Towards a collaboratively-built knowledge base of & for scalable knowledge sharing and retrieval

Philippe Martin

Towards a collaboratively-built knowledge base of&for scalable knowledge sharing and retrieval

Dr Philippe A. MARTIN (www.phmartin.info; book1_11@phmartin.info)

This document is a technical "research synthesis with incursions into some new research avenues". It is currently a slight adaptation and update of my Research Synthesis Document to obtain my Habilitation to Direct Research (HDR; *accreditation to supervise research projects*; presented and obtained on December 8th 2009)

http://www.webkb.org/doc/publications/book1_11/ ; last update: June 5th, 2011

Abstract. This document first introduces reasons why a "collaboratively-built&evaluated global well-organized secure Semantic Web" is needed to support scalable information retrieval, sharing and management processes (within an intranet or on the Web) that are both precision-oriented and completeness-oriented. With respect to that goal, current approaches for the Semantic Web and, more generally, the sharing and retrieval of knowledge – information organized in a (semi-)formal way – are insufficient. Indeed, these approaches exploit but *do not try to minimize the creation of* documents or knowledge bases (KBs) that are mostly informal or independently created and hence with few formal semantic relations between their content. In other words, they do not minimize implicit redundancies/contradictions nor support the incremental refinement, organization and evaluation of knowledge by intranet/Web users.
The main parts of this document propose various *elements of solutions* (that complement those of current approaches and that are partly or fully implemented by the *"personal"/"shared" knowledge servers WebKB-1 and WebKB-2*, both usable at www.webkb.org):

- a *web-accessible/updatable multi-source large ontology/KB* that is a loss-less integration and extension of various current top-level ontologies and which includes a transformation of WordNet into an actual lexical ontology of English with intuitive identifiers for the concepts;
- a top-level/core for an *ontology of knowledge management/sharing* (approaches, tasks, techniques, criteria for comparing tools, languages, ...) which includes a categorization of the *techniques that are proposed in this document*:
  - a cross-referencing and regular mirroring based approach between the KBs of partially competing/complementary knowledge servers so that it does not matter which KBs are queried or updated by people (this permits to combine the advantages of distributed and centralized knowledge sharing approaches);
  - a framework for a precision-oriented collaborative evaluation of the usefulness (truthfulness, originality, ...) of each piece of information and information provider;
  - KB editing protocols that keep it free of automatically/manually detected inconsistencies – and lead people to relate their knowledge/assertions/beliefs – while not forcing these people to discuss or agree on terminology and beliefs nor requiring any selection committee;
  - lexical/structural/semantic normalization rules for knowledge representation or organization;
  - various knowledge entering/search/comparison operators and KB-generated forms that extend or complement classic operators and static forms;
- *three complementary knowledge representation notations* that are more expressive, intuitive and/or concise than current common notations and whose parser(s) could be adapted to parse most current knowledge representation languages (KRLs) and allow user-specified derivations of them (to that end, an ontology of KRL structures and presentations is proposed); *a fourth language* is also proposed by WebKB-1 and WebKB-2 for permitting the mixing of these notations and the combination of their knowledge assertion/search/comparison commands.

Keywords: semantic modeling/indexation/web, knowledge/ontology/semantic based collaboration, knowledge/ontology modeling/(re-)presentation/sharing/integration/retrieval/management/tool/server, language/top-level/lexical/domain ontology, controlled languages.

Table of Contents

Depending on their embedding level, the sections of this document are, from now on, called "chapter", "section" or "subsection".

1. Introduction and High-level Summary

1.1. Selected Approach for Knowledge Management (KM)

1.1.1. Non-technical Description of the General Goal of the Selected Approach

The KM approach explored in this document is precision-oriented: it focuses on the *representation, organization, sharing, retrieval, understanding and evaluation* of individual statements, preferably undecomposable ones, not groups of statements such as whole documents. This KM approach implies giving people the possibility to

- reduce wasteful activities such as i) reading information that they already know or ii) writing information that someone has already written in "better" ways, e.g., in more precise or understandable ways,
- quickly access a *well-organized* presentation of *all* stored information *related* to a precise object that they have in mind (examples of object: concept, task, physical object, statement, ...; examples of relations: specializations, parts, corrections or evaluations of this object), and
- quickly describe and publish information in ways that permit the two previous points and *ease* the exploitation of this information for *any* kind of application (no choice restricting future exploitations should be made; the goal is general information sharing, not simply business-to-business information sharing).

Although this approach is first aimed to ease knowledge representation and sharing by knowledge engineers, progressing towards the previously mentioned goal is interesting for all persons that regularly search, read, write or evaluate documents, e.g., researchers, lecturers, students, project writers and project evaluators such as those of the European commission. As this introduction and the next chapter will show in progressively more technical ways, achieving this goal requires that a lot of people learn and use certain reading and writing techniques that at first they will find complex. However, the benefits outweigh these problems and such a process will probably begin with communities or funding organisms asking their members or applicants to follow more and more structured writing techniques.

1.1.2. General Approach or Vision

The goals listed in the bullets of the previous subsection may look like goals of most projects aiming to ease "knowledge management" in its most general sense. However, the general hypothesis of the selected approach, argued for in the following chapter, is that *genericity, scalability and efficiency* can only be achieved by an ***approach centered around interconnected knowledge servers supporting one "collaboratively-built&evaluated global well-organized secure Semantic Web" (cgosSW)***, not on loosely interconnected knowledge servers or static formal documents (e.g., documents storing knowledge bases), let alone informal documents. Each of the next paragraphs define one of the attributes of a "cgosSW".

"***Secure***" refers to the possibility for any person to hide her queries and information – or restrict operations on (and usage of) that information – from any (particular or kind of) person/agent she wants. Supporting this is not explored in this document.

"***Well-organized***" means that *most* of the concepts and statements in the information are represented in a formal way and explicitly related (and hence ordered) by semantic relations – such as temporal, spatial, mereological, corrective, argumentation and specialization relations (e.g., subprocess and physical_part for the mereological relations) – in a ***normalized way***, and hence in a non-redundant and easy to search&compare way. Modules of information grouping sets of statements (e.g., documents or long paragraphs) cannot be organized by such relations, they can mainly only be related by structural or indexation relations (e.g., inclusion, version, author, keywords, concepts, summary).

- **Ideally**, *all* paragraphs or long statements should be decomposed into interrelated ***undecomposable*** statements (a statement is "undecomposable" when removing some information, e.g., temporal information, makes it false). However, since this is time-consuming, people should be able to do this incrementally and collaboratively, when

they think the benefits will outweigh the loss of time. Similarly, all the information should ideally be represented ***formally***, that is, unambiguously, using only formal terms (identifiers for objects having a unique meaning) and a formal grammar with a logic-based interpretation. However, this is too time consuming and difficult to be asked for, semi-formal statements are interesting too. Indeed, ***semi-formal*** statements – i.e., those that would be formal if they did not use some informal terms – can often be exploited for inferencing purposes.

- **The selected approach only requires**, a ***minimal semantic organization***, one where each term and each statement has been manually – or can be automatically – connected to a formal term or statement by a logic-based generalization relation of some kind. However, the more semantically organized the information, the better. It should be noted that ***the selected approach – and hence what is meant below by "well-organized" – is only aimed to support the representation and exploitation of formal and semi-formal knowledge for "efficient" knowledge sharing, retrieval and comparison***. ***It is not aimed to support the representation or exploitation of additional information necessary for automatic problem-solving or the design of softwares***. This is why a full formalization of the knowledge (including problem-solving rules and details for executing them – information that can only be represented by trained knowledge engineers) is not necessary, why in this approach a (consistent) knowledge base can and should store diverging beliefs from different persons, why a precision-oriented KM goal could be popular in the medium or long term, and why the results presented in this document give stepping stones to achieve it. One application of the work presented in this document is to create genuinely semantic wikis ("genuinely" because current semantic wikis remain mostly "informal text"-based) that have semantic-based collaboration features, and hence to support major enhancements or complements to Wikipedia-like projects. Unlike in formal representations of domains such as in the projects CYC [www-CYC, 2009], HALO [Friedland et al., 2004] and OpenGALEN [www-OpenGALEN, 2009], not having to support problem-solving permit information providers not to "*strongly* define" certain concepts, except for explaining and solving inconsistencies between new knowledge and already entered knowledge. Thus, information providers do not have to engage in a long (and theoretically end-less) formalization of the knowledge of a domain. However, they can re-use and specialize parts of a cgosSW for automatic problem-solving.
- To illustrate what is meant above by "efficient knowledge retrieval and comparison", consider information queries such as i) "What are the arguments and objections for the use of an XML-based format for the exchange of knowledge representations?", ii) "What are all the tasks that should be done in software engineering according to the various existing 'traditional system development life cycle' models?" and iii) "What are the characteristics of the various theories and implemented parsers related to Functional Dependency Grammar and how do these theories and parsers respectively compare to each other?". Answering such queries and permitting an "efficient knowledge retrieval and comparison" requires presenting – and allowing the browsing of – well-organized semantic networks which respectively are: i) a network with argumentation, objection and specialization relations, ii) a subtask hierarchy of all the advised tasks, and iii) a network with specialization relations between the various objects or attributes related to the theories and parsers.

"***Global***" means that people would either *not have to choose* a particular node of the cgosSW (e.g., a database, Web site or community/company intranet) for making queries or additions, *or that all choices would be equivalent*: to that end, whenever useful, (parts of) the query/addition should be forwarded to all the nodes that committed to store the kind of information contained in that query/addition. That way, the cgosSW is virtually a single global Knowledge Base (KB). Subsection 2.2.4 details this point.

"***Collaboratively-built&evaluated***" means that any piece of information can be annotated – e.g., precisely evaluated, corrected or completed without risk of deletion – by any person, and this contributes to building the cgosSW. For scalability purpose, building a cgosSW should *not require* coordination/selection committees nor agreements or even discussions between people. In this document, the term "collaboration" rather than "cooperation" is used but no particular distinction is made: it does not imply any centralized task repartition and the information providers may even compete for fame or other things.

The annotations – or in other words, the whole content of the cgosSW – should then be exploitable to permit anyone to

- automatically filter out unwanted information or, in other words, find information that satisfy arbitrary complex combinations of criteria, for example i) the general criteria of truthfulness, originality and usefulness, ii) precise specifications on the way a statement has been argued for or against, or iii) specifications on what the information providers/evaluators are or think), and
- evaluate information and information providers in far less arbitrary ways than the reviewing and comparison of documents permits [AFIA, 2002].

Indeed, the writing of (informal) *documents* is necessarily affected by presentation constraints (e.g., space limits, and most importantly, the linear informal writing) as well as assumptions about the expectations of the readers. Hence, this writing results in redundant and non-explicitly organized statements. Such precise evaluations would also permit to reward -and hence encourage – "good" contributions and contributors in less arbitrary ways. Thus, the vision of a cgosSW generalizes the vision of [Hillis, 2004] about a "Knowledge Web" to which people can "add isolated ideas and single explanations at the right place" and having "mechanisms for credit assignment, usage tracking, and annotation that the Web lacks". Such a Web would for example support a much better re-use and evaluation of the work of a researcher than the current system of article publishing and reviewing.

1.1.3. The Choice of a Knowledge Representation (KR) Intensive Approach

Considering their above description, it is now clear that the selected goal and approach depart from those of projects for "KM" in its most general sense (and common sense in the industry), that is, KM based on documents or classic databases. It also departs from those of most works for "KM" in its academic sense, that is, KM based on formal knowledge representations. Indeed, research in this domain is nowadays often focused on i) automatic knowledge extraction, merging or retrieval from ***loosely interconnected formal or barely formal*** *knowledge bases (KBs)* in servers or documents, and ii) helping people create such KBs (that is, without support for strong knowledge sharing) as is the case with current ontology editors, semantic wikis and social semantic web tools [Erétéo et al., 2009].

Nowadays and in the medium term, techniques for automatic knowledge extraction, merging or retrieval are not able to understand and precisely represent the meaning of informal sentences or other data, and hence cannot create a well-organized Semantic Web. Actually, a rather safe hypothesis is that if people do not directly insert information in a cgosSW, some pieces of information in their sentences will often never be fully understood by machines or even other people and hence will often never be inserted into a cgosSW.

Another safe hypothesis is that people will always need to learn simple KR languages (KRLs) to visualize, navigate and refine parts of this semantic network in an efficient, precise or scalable way. Since people are reluctant to learn KRLs and KR best practices, and since representing knowledge is more difficult and time consuming than writing it informally, in order not to scare away potential users or to exploit already stored information, most KM projects restrict the expressiveness of the models and notations they propose or exploit. This is a safe choice for the short term but not a choice that will permit to create a cgosSW. To that end, ***an approach is to maximize this expressiveness and its exploitation while minimizing a person's effort to learn, read and manage these notations and models***. This document introduces expressive – yet rather intuitive and normalizing – notations as well as general KBs (ontologies), methodologies and tools to support and exploit these notations. In this document, ***normalizing*** means "helping to reduce and compare the various statements into which a same object can be represented by people".

The main hypothesis behind this choice for a KR intensive approach – i.e., that a sufficiently large number of researchers will sooner or later be led to create tools following such an approach, and that a sufficiently large number of persons will be led to (correctly) use such tools – will also be discussed. One of the arguments is that in the medium term some professionals or amateurs in KR or Information Technology will be led to represent their own knowledge into a cgosSW – especially researchers, lecturers and their students – and that the number of people using and extending it will then steadily grow. Although this approach is first aimed for ***manual*** knowledge modeling, it ***can also guide future advanced automatic knowledge extraction, merging or retrieval research*** by offering them i) better KBs to exploit, and ii) new guidelines about the kind of KB they should generate or contribute to for knowledge sharing purposes.

1.1.4. Quick Comparison with some Main Approaches for KM and Collaboration

The currently existing or foreseen ***Semantic Web*** (SW) – now also called Web 3.0 (Web 2.0 + Semantics) – is not a cgosSW. The SW – as described by the W3C [semArchi-Shadbolt 06] and in most current works claiming to be on this subject – is not planned to be "well-organized", "global" and "collaboratively-built&evaluated" in the above described senses. Like the current Web, the SW is most often seen as a Web of data albeit indexed by generally very lightweight semantic representations (e.g., simple categories from ontologies, thesaurus or folksonomies) stored and organized by Web users in more or less independently created KBs, hence heterogeneous and loosely interrelated KBs. (This also applies to *ontologies*, the parts of each KB that defines and organizes the formal terms it uses). Like "data" – as opposed to "genuine knowledge" (homogeneous formal semantic representations) – the information in these KBs are hard to find, match, merge and exploit in logical/relevant ways; hence, only short – and not necessary logical – chains of inferences are expected to be automatically performed on them. Current knowledge sharing approaches try to re-use independently developed ontologies (which is understandably difficult and does little to ease the work of knowledge seekers and knowledge providers) or propose a shared knowledge base with no edition protocols nor guidelines to keep it organized. Finally, the KRLs proposed by the W3C for the Semantic Web are currently designed to have good properties for inferencing purposes but are not yet expressive and normalizing enough to be adequate for the representation and sharing of non-simple kinds of knowledge. Various arguments are given for this last point in this document, including in Chapter 4 which compares various KRLs; [Kalfoglou et al., 2004] and [Patel-Schneider, 2005] give some complementary arguments.

Even in ***Semantic Grids and semantic-based Peer-to-Peer networks*** (with generally, one KB per node/peer), the partial redundancies and inconsistencies between the KBs are not made explicit and hence the replication of queries or knowledge among the KBs is restricted; thus, the search and exploitation of their knowledge is restricted too. Even in such networks, knowledge sharing, ontology evolution and collaboration are not based on "encouraging a collaboratively-built&evaluated global KB and then on the querying or filtering of these KBs by each user" but on "a more or less extended selection committee accepting or not to include/keep knowledge in a shared KB based on the assumed expectations of the users".

Similarly, current tools for direct/indirect/liquid ***e-democracy*** focus on voting-related issues, proxy chains and workflows, but do not permit their users to freely add to – and thus collaboratively build – a non-redundant well-organized semantic/argumentation network of statements (typically of interconnected hypothesis, observations, preferences and decisions) and evaluate these statements in precise ways that other users can exploit for precise search or comparison of statements and hence well-justified decisions.

More generally, as in current ***social networks***, the search and exploitation of information is limited by its lack of explicit semantic content and organization [Erétéo et al., 2009].

1.2. Selected Research Directions and their Guidelines, Hypothesis or Difficulties

The following subsections introduce the next chapters except for the last subsection which further explains the aim of this document, its content, format and decomposition into chapters. Some general ***research questions*** that these chapters aim to answer are ***what are the criteria*** for judging the quality of knowledge representations, notations and libraries, ***what kinds of techniques*** can satisfy those criteria and help *design or generate* these artifacts, and ***how to semantically organize these criteria, techniques and artifacts***. The general ***guideline*** of the research directions presented by these chapters is to ***reduce the implicit redundancies or inconsistencies*** between the knowledge objects and, more generally, increase the semantic organization of these objects. Some more specialized formulations of this guideline are: i) the objects should be represented as precisely and uniformly as possible, and ii) the objects should be as small and explicitly interconnected by semantic relations as possible, at least by identity and specialization relations. A related hypothesis is: the bigger and more organized the KBs, the easier it is for software to align/merge these KBs or guide the users in entering precise and re-usable knowledge.

1.2.1. Towards a Process-focused Ontology of KM (Processes, Structures, Tools, ...)

Nowadays, the word "ontology" refers to a set of formal terms – i.e., unambiguous identifiers for certain objects with unique meanings (concepts, statements, ...) – and to their associated formal (logic-based) or informal descriptions (partial/complete definitions, other statements) which relate the objects with respect to each other. An ontology may also include informal terms and connect them to other terms, typically via lexical relations. A ***collaboratively-extendable well-organized core ontology of KM*** (cooKM) is one necessary element to permit people (researchers, lecturers, students, engineers, ...) to index, share, organize, compare or retrieve KM information (tools, techniques, ...) relevant to their needs and in a scalable or efficient way. Then, the approach could be extended to other domains. Since this ontology would organize KM best practices and other resources, it would also be a guide for KM in general and hence for any cgosSW.

No such ontology currently exists in any domain (and the results presented below are only steps "towards" achieving it, hence the title of Chapter 2). The major works of the early 1990s on libraries of models for generic knowledge modeling tasks – in particular the library of CommonKADS [Breuker & van de Velde, 1994] – are of course very interesting resources to initialize it but, since they are focused on knowledge modeling, do not provide categories for a large percentage of important KM processes. In any domain (not just KM), process categories are represented in topic hierarchies or poorly organized ontologies of "subject areas" (topics), e.g., Yahoo's topic hierarchies and – in KM – the "Semantic Web Topics Ontology" of ISWC 2006 [www-SWTO, 2006]. In this document, the expression "topic hierarchy" refers to a list of formal or informal terms organized with only one kind of relation which is hierarchical and does not distinguish the various semantic (or lexical) relations that exist between the terms. With a topic hierarchy or a poorly organized ontology, different users are likely to insert or look for a same piece of information at different places (or represent it in different ways), thus quickly leading to implicit inconsistencies and redundancies; this decreases the organization of the hierarchy or ontology and makes knowledge sharing or retrieval progressively more and more difficult.

Chapter 2 illustrates parts of a core for a cooKM. Indeed, it i) describes and organizes research contributions *about* certain KM processes, i.e., about ***techniques, best practices and applications for knowledge sharing, modeling, comparison, retrieval and evaluation***, ii) situate them with respect to related processes or resources, and iii) follows the above cited best practices. This knowledge essentially is (or, for some parts, will be) represented in the knowledge server WebKB-2, thus allowing any Web user to navigate, query, extend or correct this knowledge.

The ***difficulties*** related to the content of Chapter 2 were to come up with theses contributions, implement them in tools, and finally represent, organize and argue for them in a scalable way. ***Among these contributions are:***

- various operators that extend classic search and comparison operators based on the projection of a query graph; these operators always give "relevant results", are efficient and can exploit statements of arbitrary expressiveness; however, they do not give results that are consistent and complete with respect to (first/Nth order) logical deductions on the KB;
- an operator to generate "scalable ontology-based entity comparison tables" and a core ontology for comparing KM tools; there is indeed an important demand for tool comparison in the KM community; this demand explains the success of Michael Denny's "Ontology editor survey" [Denny, 2004] despite the fact it was very superficial and hence rather misleading; with the presented approach, tool authors can directly and collaboratively describe and compare tools as precisely as they wish and generate comparison tables according to arbitrary complex criteria;
- an ontology of criteria – and knowledge representation/sharing best practices to achieve these criteria – to permit a scalable collaborative building of an organized semantic network, hence a network of body of knowledge that does not turn into a "knowledge soup" or a "spaghetti-like network" when it grows;
- a mechanism of partial mirroring between knowledge servers of a cgosSW in order to combine the advantages of distributed and centralized knowledge sharing approaches and thus challenge the common thinking that the large scale distribution of knowledge providers can only be achieved via the semi-independent development of partially redundant and inconsistent KBs;
- a framework to evaluate or quantify the popularity and usefulness of each information provider or piece of information, based on argumentation relations between these objects and votes or knowledge representations about their originality, usefulness or other characteristics;
- KB editing protocols that enable and encourage people to relate or integrate their knowledge into a KB while i) technically solving or avoiding manually/automatically detected semantic conflicts and redundancies, ii) not forcing the users to discuss or agree on terminology and beliefs, and iii) avoiding the classic problem of either allowing any user to modify any part of the KB (as in wikis) or having the bottleneck and restrictions associated to the existence of a selection committee.

1.2.2. Towards a General Ontology for KM

Chapter 3 presents an ontology useful *for* any KR intensive KM process. This chapter illustrates the content – as well as the rationales and techniques for the design – of an ontology created by i) transforming the noun-related part of WordNet [Miller, 1995] into a genuine (semantically correct) lexical ontology with short intuitive identifiers, and ii) integrating this lexical ontology in a loss-less manner with many top-level ontologies (e.g., DOLCE, SUMO, LIS, NSM) and some domain ontologies, including the above cited core ontology for KM. [Sowa, 2003] named the result "the Multi Source Ontology" (MSO), acknowledged it was "a necessary part of any suitable, scalable knowledge sharing effort" and recommended it as a "candidate material for a standard". It was voted as such a "candidate material" by the IEEE Standard Upper Ontology Working Group [IEEE-SUO-MSO, 2004] after having examining it. This ontology is the one proposed by the main WebKB-2 server which permits any Web user to extend it.

All ***ontology integrations in the MSO have to be "loss-less"***, that is, the categories are associated to their source ontologies (more precisely, the category identifiers include identifiers for their source ontologies) and the meaning of the categories in the source ontologies is not changed (over-interpreted) except when internal inconsistencies in the sources are detected and hence have to be fixed before integration. This is a requirement for the above cited collaboration techniques and permits the users to re-generate any source ontology or generate combinations of (parts of) the source ontologies if they wish to. This also permits to integrate subsequent versions of the source ontologies. Nowadays, (automatic or manual) *loss-less* integrations are still rare. WordNet has often been re-used in ontology related works, especially for information retrieval, but the MSO still seems to be the only work that has converted

WordNet into a genuine consistent ontology with intuitive identifiers and that has not over-interpreted it (as was for example the case when OntoWordNet [Gangemi et al., 2003] has been created).

One difficulty of the integrations in the MSO lies in making explicit the often implicit semantic relations between the concepts of the source vocabularies and finding a "right place" for these concepts into the MSO. One rather safe ***hypothesis*** behind these integrations is that different, internally consistent, ontologies do not have to be modified to be integrated into a unique consistent semantic network. A stronger hypothesis is that the categories of these different ontologies can always be inter-related into a *well-organized* semantic network, especially via specialization relations. However, it appeared that the difficult cases can be solved via "extended specialization" relations. A hypothesis that generalizes the two previously ones is that semantic conflicts can always be solved by adding more precision or making explicit how they boil down to mere "preferences", and hence that solving conflicts increases the organization of the KB. A related hypothesis is that solving conflicts (adding precision) can be done incrementally (when a new piece of information is added) and that people do not have to meet nor compromise to merge their ontologies – this is the main hypothesis behind the above cited collaboration protocols. These hypotheses will be discussed or illustrated.

1.2.3. Towards a Language Ontology for KM; Intuitive, Expressive and Personalizable Languages

Compared to many other KRLs, Conceptual Graphs (CGs) are more expressive, intuitive, concise and normalizing. These characteristics are generally the main reasons why those who adopted it did so. The first three characteristics explain why CGs are often said to be relatively "close to natural language". The last three characteristics come from its graph-based nature and from two of its textual and graphic notations (namely, CGLF and CGDF) which have similarities. The CG model – and CGIF, another textual notation for CGs – are part of Common logic (CL) [ISO/IEC 24707, 2007], an interlingua framework for logic languages based on first-order logic or subsets of it. However, the above characteristics can be improved on and it is very valuable to do so. This led to the creation of Frame-CGs (FCG), For-Links (FL) and Formalized-English (FE), ***three formal notations of complementary kinds that***

- ***have a same underlying formal model – an extension of the CG and CL models for expressiveness and collaboration purposes***; however, these notations are *not more expressive than KIF* ("Knowledge Interchange Format"; an ancestor of the CL model and its CLIF notation, as well as the *reference language* adopted in this document for logic interpretation purposes);
- ***improve on at least two of the above cited characteristics (expressiveness, normalization, intuitiveness and concision) over other notations***, for example,
  - prefixed KRL notations (in which a statement begins with its predicate or relation, as in CLIF),
  - notations in which quantifiers are explicit (as in CLIF),
  - frame-like/graph-based notations in which quantifiers are implicit (as in RDF/XML – the XML linearization of RDF – CGLF), and
  - the notations of *most* formal Controlled Natural Languages (CNLs [www-CL 2009]; FE rates well in Jonathan Pool's comparison of CNLs [Pool, 2006]).

  There are currently very few formal CNLs (FCNLs) having the expressiveness of at least the CL model. Apart from FE, one such FCNL seems to be Attempto Controlled English (ACE) [Fuchs et al., 1999] and only its expressiveness may be claimed to be improved on by FE. Another one, if (vs. when) implemented, is (vs. will be) Common Logic Controlled English (CLCE) [Sowa, 2007]. FE is a bit less readable than ACE and CLCE but has the advantage of having an explicit logical structure and the same one as FCG. Comparing FCNLs with other kinds of notations on concision or intuitiveness criteria is not really relevant because i) FCNLs are *purposely* less concise to be more intuitive for beginners, and ii) they are also less "visually structured", that is, they make it much more difficult to see if and how a certain number of things are related together.
  As shown in Section 2.1, FL is probably one of the most concise and "visually structured" possible "textual notation with the expressiveness of at least the CL model" that can be designed; however, for very *complex* statements, FL is not the most intuitive or handy notation.

Furthermore, a general textual KR notation called "For-Structuring" (FS), having all the above cited notations as sub-languages, was also created. More precisely, CGLF, FCG, FL and FE are currently included, and CLIF, KIF, CGIF and RDF+OWL/XML will also hopefully be fully included in the future, (currently, only parts of these last notations are parsed). All these notations permit to write (formal or semi-formal) logic-based statements. FS permits to use them along with query operators and procedural control structures to create queries or programs. Thus, in an FS input/output file, an ontology or KB is an "ordered" set of assertion/querying/control *commands*. Once parsed, the KB is, as usual, a set of logic formulas or functions, possibly ordered by relations between them or between the terms.

The work on notations presented in this document is an *exploration* of which notational features are needed to ease knowledge representation and sharing, which ones lack in current notations, and how to include them into a notation statically or dynamically.

- FL is the most original and useful of the introduced notations. It can have the aspect and concision of a classic frame-based notation (the most common kind of "simple" notations or of notations designed for readability purposes) but is much more expressive. Since FL is the notation mainly used in this document, it is introduced before presenting anything else in the first subsection of the next chapter (Section 2.1). Although this introduction of FL may sometimes be tedious, it acts as a reference section for the syntax and semantics of FL and permits to further introduce some terminology used in the rest of the document. Finally, it permits to list features of FL, FCG and FE that are uncommon in classic KRLs but that are needed for designing, sharing and presenting ontologies that are not small or simple. (These features were needed to design the MSO).
- Section 4.2 compares various kinds of notations – FCG, FL, FE, KIF and some Semantic Web languages such as RDF+OWL/XML and N3 – on a panel of knowledge representation cases. However, not all the features listed in Section 2.1 are re-used. Inferencing related issues such as decidability, efficiency, completeness and consistency are not discussed. Chapter 2 argues that they are not relevant for genuinely "general purpose" languages since each inference engine could automatically extract and exploit the parts of the statements corresponding to the expressiveness it can handle. For knowledge sharing purposes, knowledge providers should not do themselves such a task, i.e., restrict or bias their statements for a particular range of applications or a particular inferencing sweet-spot. Section 4.1 uses FL to organize arguments and objections about the need for a *general* KRL to have certain expressive features and be readable, and hence arguments and objections about the inadequacy of RDF+OWL as a model for such a KRL and the inadequacy of RDF/XML as a notation for such a KRL.
- Section 4.3 presents a shared LR(1) grammar for parsing FL, FCG, FE, CGLF and (for now, only parts of) CGIF and KIF. Currently, WebKB-2 has a separate parsers for each of these languages. These parsers will be merged to cover all these KRLs and hence, as explained below, most families of logic-based and graph-based notations.
- Section 4.4 introduces different kinds of data models for handling knowledge and storing them in databases, and summarizes the future data model of WebKB-2 (the need for updating the current model of WebKB-2 is presented in Subsection 2.4.2. WebKB-2 reuses the object-relational main-memory or fully-disc-based DBMS FastDB/Gigabase for persistency and transaction purposes.
- Section 4.5 presents the core ontology for components of models and notations of KRLs – in other words, a meta-model for KRLs and a model for their syntactic presentation. This ontology is intended to permit the building of ***generic parsers for families of notations, based on the values/instances of certain categories in this language ontology***, values that the end users will be allowed to use or change for specifying the components of the language they use. This approach has several advantages.
  - It will permit the building of generic KRLs that can be personalized by people to suit their own preferences or to make it look like an existing fixed notation for import/export purposes.
  - It will be more flexible and easier to use than GRDDL [www-GRDDL 2006], a knowledge extraction mechanism proposed by the W3C to permit Web page authors to use any KRL they wish for knowledge in their documents *on the condition* that they specify an algorithm (typically represented in XSLT) that transforms this KRL into RDF. Writing such an algorithm (or modifying it if some changes to the KRL are made) is difficult. Furthermore, for translating FL, FCG or FE from/to KIF or RDF+OWL (i.e., more generally, languages with lots of features for concision and readability to lower-level languages), the parsers or export procedures of WebKB-2 must often make accesses to the MSO; similar accesses to an

ontology are not possible or not efficient in a language with XSLT parsers since they are meant for direct syntax translations and do not run within a KBMS storing the ontology.
- It will permit people or applications using restricted but common (hence, de-facto "general purpose") languages not to be limited by these languages, e.g., by their poor expressiveness (as with RDFa), by their cumbersome syntax (as with XML-based notations) and by their cumbersome low-level model (as with RDF+OWL).
- It will simplify syntactic translation between languages (however, this does not address the semantic issues related to translations between languages or ontologies).

1.2.4. Towards an Ontology of Knowledge Presentation

As hinted in the previous paragraph, the above cited core language ontology is also a core for a knowledge presentation ontology. Thus, these two issues are merged in Section 4.5. Subsection 4.5.5 shows how arbitrary complex parsing/presentation directives can be built with the current model. However, this ontology does not yet include most of the knowledge presentation features of WebKB (WebKB-1 and WebKB-2):

- its ***knowledge querying/filtering/presentation options***;
- its options for generating ***cascading knowledge search/entering forms*** from definitions or general statements associated by people to concept types in the KB (the menus can be combined to guide the search or entering of knowledge; see Section 2.4.4),
- its options for generating ***scalable ontology-based entity comparison tables*** that i) permit the visual comparison of objects in a precise and scalable way (the format of the table does not have to be changed when new features/criteria or objects are added) and ii) ease the entering of knowledge; (such tables exploit the specialization and partOf relations in the ontology and can be generated from the content of the KB in answer to a comparison query);
- its future options for the above cited knowledge evaluation algorithm;
- its future options for "organizing large volume of knowledge for presentation purposes in answer to queries" and hence also for "grouping and organizing into hierarchies a long list of query results".

WebKB is composed of two complementary KB servers.

- WebKB-1 [Martin & Eklund, 2000], which was developed between 1997 and 2000], is a server with no persistent KB, hence, a "personal KB server": the user must specify which input files should be loaded into the KB before making queries. WebKB-1 offers many features to a knowledge engineer for inserting (scripts of) commands (especially, knowledge representations or queries) within Web documents, relate them to any part of any Web document, and allow end-users to exploit these relationships or commands for retrieving or generating (parts of) Web documents. WebKB-1 is not directly described in this document.
- WebKB-2 [Martin et al., 2005], which is developed since mid-2000, is a shared KB server. It offers many features for knowledge engineers to retrieve and collaboratively update a large persistent shared KB. It is ultimately aimed to support a cgosSW. It also includes most features of WebKB-1, although yet not all of them. This document describes techniques that are or will soon be used in WebKB-2.

1.2.5. About this document: its Goal, Decomposition and Amount of Formal Descriptions

The following chapters contain a *lot of information in a formal and semi-formal format rather than via informal sentences*. Both are important since, mainly due to time constraints, *not all* the important information presented *informally* in this document are currently also represented in a *(semi-)formal* way. However, this will come and the basis for this is presented. Indeed, to illustrate, follow and extend the results of the selected KM approach, one *motivation* for this document was to (re-)present, organize, justify and generalize the main ideas of the author's post-PhD research into a core for a cgosSW, thus also allowing any reader to correct or complement these ideas (via WebKB-2) in a scalable, easily retrievable way and without having to introduce redundancies by summarizing or re-

situating the new materials. As noted earlier and argued for below, traditional writing and publishing do not permit this and renders the review of information or information providers quite arbitrary since they link it to the review of documents and the way these documents are presented.

Comparison between this document and the input files for the MSO. At its inception, this document was meant to be an "onto-book", that is, a book that is both i) a "relatively" intuitive but formally structured research description that can accommodate many additions without its structure having to be changed, and ii) a file that can be used as an input file or backup file for a large ontology (in this case, the MSO) and be regenerated from it to include new additions made in the ontology. In the end, this goal had to be left for a subsequent version. However, this document contains a lot of formal descriptions and is scalable to a limited extent: there are "right places" for inserting new (representations of) ideas and categories related to those already represented in this document. Nevertheless, the modularity (and hence scalability) of this document is limited by its important informal parts and structure. Hence, it cannot be used as one of the input files of the MSO. Subsection 2.3.4 gives one approach for the scalable decomposition of input files. To sum up, the ideas and categories represented in this thesis are (or will be) also stored in actual input files for the MSO. This acknowledges the unfortunate manual duplication of formal content between this document and these files. However, like input files, this document can be seen as a static view on parts of the MSO, with additional informal information and an ordering of the information that cannot be re-generated via queries. This is one of the interests of keeping input files and permitting their access to the users while they navigate the MSO.

2. Towards a *Process-focused Ontology* of Knowledge Management

2.1. Some Top-level Concepts of Knowledge Management

2.1.1. Introduction to the Main KR Notation Used in this Document

FL, FCG and FE share many features, can be used for assertions or queries within FS and, to ease readability, can be used within each other provided that sub-statements in a different notation are embedded within their distinctive delimiters. The main KR notation used in this document is FL because it is the most concise, structured (i.e., it maximizes the possibilities to join statements and hence avoid to repeat parts of them) and often simplest to read. This is due to the following six features.

- In a FL statement where terms are linked by a binary relation, the quantifiers associated to the terms are located in the context of the relation that connect those terms. A context is a meta-statement, that is, a statement on the contextualized statement. Seen from another viewpoint, a context is a list of relations from/to each of the objects described in the contextualized statement, including the relations of this statement. A "restricting context" is a context that sets relations specifying restrictive conditions for the contextualized statement to be true, and hence specifying restrictions on the existence of the objects in the contextualized statement. In this document, such relations (e.g., temporal relations) are called "truth restricting relation from a description", while other relations from/to a statement (or a situation described by this statement) are called "truth preserving relations".
- This context can be located just before the relation term, just after it or after the destination term of the relation.
- To add a relation of the same type, the relation type need not be repeated, the new destination term simply need to be added.
- Certain contextual information (including certain quantifiers) can be left implicit.
- A concise notation is available for cardinalities and quantifiers.
- Various kinds of collections (and/or/xor, distributive/collective/..., type partition, ...) can be used.

The next titled paragraphs illustrate some features of FS an FL, mainly those that are needed to understand the representations of the next sections. The text in courier font shows a formal code accepted by the FS parser. Its translation in 1st order logic is given in Section 4.2. FL, FCG and FE are 1st-order logic *notations* (with contexts and collections): they are not dependent of any particular model (Description Logics, Conceptual Graphs, Common Logics, etc.). The next paragraphs are numbered because they are referred to in other sections of this document.

2.1.1.1. Strings, variables and code delimiters. In FS and its sub-languages, strings can be single quoted, double quoted or delimited by "$(" and ")$". This eases the embedding of strings. The escape character is '\', e.g., 'It\'s a string.' is equivalent to "It's a string.". Juxtaposed strings are concatenated, e.g., "ab" 'cd' is equivalent to "abcd" or $(abcd)$.
Variable names are prefixed by '?', '*', '$' or '^'. Variables with a '?', '*' or '^' prefix behave like variables in KIF, CGIF or CGLF (the '^' prefix is for free variables, i.e., variables that are implicitly universally quantified). Variables with a '$' prefix behave like the variables of shell scripts and can be set with a value as in shell scripts. All variables are interpreted in double quoted string or strings delimited by "$(" and ")$". With respect to variables with a '?', '*' or '^' prefix, these two kinds of strings are equivalent to KIF quoted expressions where variables are preceded with commas. Interpreting a variable with a '$' prefix means replacing it with its value. For example, the following two statements or commands are valid in FS.

```
max=8;  for n=1 to $max { print "here is a number: $n"; }
```

Apart from the interpretation of variables, the content of strings are not interpreted by the FS parser except for a string delimited by "$(" and ")$" when it is used as a delimiter for FS code within informal text. In an HTML file, FS code can also be isolated within the strings (HTML tags) "<script language='FS'>" and "</script>". This is one way to mix formal and informal information in FS.

2.1.1.2. Comments and annotations.

```
/* This is a multi-line "comment".
   //This is an in-line comment within a multi-line comment. */
```

Comments are discarded by the FS parser, unlike "annotations" which are informal notes (explicitly or implicitly) related to an object via an "annotation" relation (even though in RDFS, the type of such relations has for identifier "comment").
HTML comments are recognized and ignored: their content is not made visible by a Web browser but their content is parsed by the FS parser. More generally, the parsers of FS, FL, FE and FCG ignore HTML tags, hence they can be use within representations in input files to highlight certain parts.
Before mixing formal and informal parts via comments or code delimiters, a knowledge provider should first *consider* relating them with *precise* relations (hence preferably not annotation relations) and structure the informal parts by splitting them into shorter strings and connecting them using languages such as FL or FE. This is of course not always appropriate (e.g., this document is mostly composed of informal text).

2.1.1.3. Identifiers, names and relation from a type.

```
pm#thing  pm#name:  "something";
```

The above line is an FL assertion (i.e., a statement asserted in FL) that relates the formal term pm#thing to the string "something" by a relation of type pm#name. (In FS, both pm#name and pm#thing are predefined. This last term refers to the (non strict) supertype and type of all imaginable types or things). Thus, this line asserts that (and should be read as) "pm#thing has for name 'something'". Hence, given the semantics of pm#name, "something" becomes another name for pm#thing: "thing" and "something" are informal terms/names (which may be names for other things) while pm#thing is a unique identifier (a formal term). Since the main WebKB server (www.webkb.org) accepts conceptual queries as GET parameters, it permits to use different parameters (and hence different URLs) to query different information on the referred object. For example, the next first URL asks for all the direct relations from pm#thing in RDF/XML. The next second URL asks for the direct relations from pm#thing and all its subtypes via a recursive exploration on three levels, in a format similar to FL but relying on indentation.

```
http://www.webkb.org/bin/categSearch.cgi?categ=pm%23thing&format=RDF
http://www.webkb.org/bin/categSearch.cgi?=&categ=%23thing&recursLink=%3E&depth=3
```

In FS, and in the rest of this document, when referring to terms from an ontology, informal terms (strings) are enclosed within double quotes while formal terms are either not quoted or quoted within a single backquote and the single right quote (as with `pm#thing').*** Indeed, such quotes can be used as delimiters for terms and statements in FE. As their prefix indicates, pm#thing and pm#name have been created by `pm', some user (person or software agent) of FS (and, as it happens, an identifier in the KB of WebKB-2 for the author of this document). ***In this document, formal things which are not necessarily terms in an ontology (e.g. query operators and syntactic sugar) are single quoted.

2.1.1.4. Introduction to contexts. In the first example of the previous paragraph, the author of the relation/statement has been left implicit: ***according to the default presentation/parsing rules of FS, this author is `pm' because the source term has for author `pm'***. FL statements equivalent to the above one but showing the relation creator are:

```
pm#thing  pm#name _[pm#author: pm]:  "something" ;
pm#thing  pm#name:  "something" __[pm#author: pm];
[pm#thing pm#name:  "something"] _[pm#author: pm];
pm#thing  [_ pm#author: pm] pm#name:  "something";
[_ pm#author: pm] [pm#thing pm#name:  "something"];
"something"  [_ pm#author: pm] pm#name of: pm#thing;
"something"  pm#name of: _[pm#author: pm]  pm#thing;
"something"  pm#name of: pm#thing __[pm#author: pm];
[ [ [ [ ["something" pm#name of: pm#thing] _[pm#author: pm] ] ] ] ];
```

These equivalent statements illustrate the following three points.

- The direction of a relation can be reversed by adding the keyword "of" (a statement such as "X R of: Y" should be read "X is the R of Y").

- A statement can be enclosed by any number of balanced square brackets. It should be delimited by square brackets if is contextualized as a whole.
- A context must be delimited by '[' and ']' and can be located before or after the relation term, before or after the contextualized statement, or after the destination term, on the condition that the '[' delimiter is adequately prefixed or postfixed by one or two underscores.

As illustrated by the next FL statement, a relation may have various "believers". By default, an author is assumed to be a believer (more precisely, pm#author is a subtype of pm#believer).

```
pm#thing  pm#name:  "something" __[pm#author: pm,  believer: oc];
```

When a term has many names from the same creator, instead of explicitly using the relation pm#name, the names can be concatenated using three underscores, as in pm#thing___something which is another identifier for the concept type referred to by pm#thing.

2.1.1.5. Introduction to quantifiers, definitions and relation signatures. FL, FCG and FE uses various kinds of quantifiers and various syntactic forms for them:

- 'any' for the use of a universal quantifier to *define* properties of a term,
- 'every' for using a universal quantifier to state properties that *happen* to be true (in, may be, a specified context),
- 'a', 'an', 'some' or '?' for the existential quantifier,
- values or ranges for numerical quantifiers, e.g., '0', '1' (or 'the'), 'at least 1' (or '1..*'; except in rare cases such as within complete or ordered sets, using '1..*' is equivalent to using the existential quantifier), 'between 0 and 60' (or '0..60'), '60..*', '60%' and 'most' (which means 'at least 60%'). The "percentage" quantifiers are defined in KIF in Subsection 4.2.7.

Numerical quantifiers that are not percentages are like cardinalities in entity-relationship diagrams but they can be used in both the source node and the destination node of a relation. As with cardinalities, the default quantifier for the source node is 'any' and the default quantifier for the destination node is

- '0..*' (alias '*') when this destination node is about a type, and
- '1..*' when this destination node is about an *individual* (i.e., a formal term that does not refer to a type, e.g., a string; one may note that if wn#Paris is an instance, `a wn#Paris', `any wn#Paris' and `4 wn#Paris' are equivalent, hence '1..*' is equivalent to '1').

This default is necessary to avoid the specification of quantifiers when displaying parts of lexical ontologies such as WordNet and hence displaying these parts in a non-cumbersome way. Thus, the next statements are equivalent and, unless the signature of the relation type pm#name changes the default interpretation, they mean '*any (instance of)* pm#thing has for name "something", and "something" is name of 0 to any number of instances of pm#thing'.

```
pm#thing  pm#name: "something";
pm#thing  pm#name: "something" __[any->1..*, 0..*<-any];
pm#thing  pm#name: "something" __[any->?];
pm#thing  pm#name: "something" __[any->?, pm#author: pm];
pm#thing  pm#name: "something" __[any->? _[pm#author: pm], pm#author: pm];
pm#thing  pm#name: "something" __[any->? _[pm#author: pm] ];
pm#thing  pm#name _[any->?]: "something";
any pm#thing  pm#name: the pm#string "something";
"something"  pm#name of: pm#thing __[any->0..*, 1..*<-any];
//The order of the quantifiers is important: 'any->?' is different from '?->any'
```

However, the interpretation that one would intuitively expect is "*the type* pm#thing has for name "something", and "something" is the name of *the type* pm#thing and maybe other types". This can be expressed by using '.' in any of the following ways.

```
pm#thing  pm#name:  "something" __[.->?, .<-?];
pm#thing  pm#name:  "something" __[.<->?];
pm#thing  pm#name:  "something" __[.<->.];  //this form is to be used with caution
```

The '.' permits to state that the relation is about a category, not about its instances. For an individual it would seem that using '.' or not does not matter but Paragraph 2.1.1.10 shows that any term (including individuals and strings) may have "extended specializations". Thus, it is better to use '.' *only when* one particular *type* needs to be referred to.

To avoid forcing the use of '.', the signature of the relation type pm#name may precise that this should be the default interpretation, as in the following equivalent relation type declaration.

```
pm#name .(pm#thing . [0..*], pm#string [1..*]);
pm#name .(., pm#string [1..*]);
```

'.(' introduces a relation signature. The first identifier after '.(', here pm#thing, specifies that the source node of the relation should be of type pm#thing, i.e., that any thing may have a name (here, pm#thing may be abbreviated by '?' or left implicit since '.' is specified). The following '.' specifies that the relation applies to the object used in the source node of the relation, not to its instances. If pm#term or a 2nd order type such as pm#type is used in the signature (or an Nth-order type with N superior to 2) the '.' needs not be specified, it is the default. Thus, relation signatures very rarely need to use the '.'. Then, '[0..*]' is the reverse cardinality for the relation: any string may be the name of 0 to many types (since such a cardinality is the default, it can be left implicit). Then, the type that any destination of such a relation may have is specified: pm#string. Then, '[1..*]' is the direct cardinality for the relation: any (declared) thing has 1 to many names.

If given the above signature, `pm' still wanted to state that *any* (declared) thing *has, by definition,* "something" as a name, he would simply have to make the relevant quantifier explicit, as follows.

```
pm#thing  pm#name:  "something" __[any->?];
```

Since this last statement is a definition, it is "neither true nor false" in the sense that no one may contradict it, but it is "true by definition" in the sense that it states a relation (and a necessary condition) that `pm' associates to pm#thing and its subtypes. If another person (say, `oc') does not like this definition, he has to declare another term (say, oc#thing) and relate it to pm#thing (and, if needed, its subtypes) via some relation, e.g., a generalization relation. On the other hand, the following statement asserts that according to `pm' every (declared) thing *happens to have* "something" as a name. This following statement (which is clearly false) uses a classic universal quantification and hence may be "corrected" by other people using a relation such as pm#corrective_specialization (the supertype for all relation types having for destination an `extended_specialization' which is also a `correction').

```
pm#thing  pm#name:  "something" __[every->?, 1..*<-.];
```

To state that having the name "something" is a sufficient condition for being a thing, first by assertion and then by definition:

```
pm#thing  pm#name:  "something" __[every<-.];
pm#thing  pm#name:  "something" __[any<-.];
```

If a type was used instead of "something", `__[any<-.]' would not be a enough to state "sufficient conditions". Hence, to state them, it is safer to use the two following equivalent forms (the first one is needed because it permits to concisely express various quantifications on a same relation by different users but the first one will be used in the rest of this document because it makes the "sufficient conditions" mark easier to spot by the reader):

```
pm#thing  pm#name:  "something" __[.<=.];
pm#thing  pm#name<= "something" __[.<-.];
```

To state that pm#thing can *only* have "something" as a name:

```
pm#thing  pm#name:  "something" __[.=>.];
pm#thing  pm#name=> "something" __[.->?];
pm#thing  pm#name=> "something" __[.=>.];
```

If a type was used instead of "something" (e.g., if `[.=>1..*]' had been used), the `=>' could be translated in RDF/XML using owl#allValuesFrom (`owl:allValuesFrom' in the RDF/XML syntax).

To state that at least two things (that are not connected by an identity relation) *have* "something" as a name:

```
pm#thing  pm#name:  "something" __[2..*->?];
```

2.1.1.6. Relations and order of parameters in FL.

```
pm#Tom  pm#kind:  pm#person __[pm#author: oc]  pm#document_creator __[pm],
        pm#identifier:  oc;
```

The above line asserts that pm#Tom has for pm#kind a pm#person according to `oc', has for pm#kind a pm#document_creator according to `pm', and has for pm#identifier `oc' according to `pm'. This example shows that i) relations and relation destinations can be added without repetitions, and ii) within a context, the pm#author relation can be left implicit (in any case, the author must be a registered user). pm#kind refers to the usual meaning of an "instanceOf" relation type: the source node of the relation (the instance) must conform to (and "inherits from") the

characteristics that were associated to the type via a partial/total definition of this type or via a statement using this type with a universal quantifier. Here are FL statements that are equivalent to the above one.

```
pm#Tom  pm#kind:  pm#person __[oc]  pm#document_creator,  pm#identifier: pm;
pm#person  pm#instance: (pm#Tom  pm#kind: pm#document_creator __[oc], pm#identifier: pm);
```

These equivalent statements show that i) pm#instance is the inverse relation type of pm#kind, and ii) parenthesis can be used for attaching relations to an object within a statement without introducing a new statement (thus, unlike when square brackets are used). The use of '_', '.' or ':' before or after '(' permits to change the aspect or order of the list composed of the relation name and its parameters, which is sometimes handy. These last two points are illustrated by the fact that the following statements are equivalent.

```
pm#Philippe  pm#kind:  pm#person;      [pm#Philippe   pm#kind:  pm#person];
(pm#Philippe pm#kind:  pm#person);     [(pm#Philippe  pm#kind:  pm#person)];
pm#Philippe  _(pm#kind: pm#person);    (_ pm#kind:  pm#person)  pm#Philippe;
pm#kind _(pm#Philippe  pm#person);     (_ pm#Philippe  pm#person) pm#kind;
(: pm#kind:  pm#Philippe  pm#person);  (: pm#kind  pm#Philippe  pm#person);
(: pm#Philippe  pm#kind:  pm#person);  (: pm#Philippe  pm#person  pm#kind:);
```

The "_(", "(_" and "(:" prefixes of FL are shared by FCG and can make these notations look like and be used like lisp-based notations, which is interesting in certain cases, for example when functional relations or functions are used. Furthermore, although this has not yet been implemented in WebKB-2 but is ***prepared for via the notation/presentation ontology of Section 4.5, such prefixes and related syntactic sugar could be changed by each user (and, in certain cases be made optional) by selecting values from this ontology or changing the default values that define the selected language (Table 4.5.5.3 gives examples).*** This will be a major step to permit the re-use of the same import/export programs to parse or generate knowledge in some other notations. Having different prefixes for contexts/statements, groups of relations and relation/function parameters – e.g., by default in FL, FE and FCG, "_(", "(_" or "(:" for function calls and ".(" for relation signatures – permits FL, FE and FCG to have a Lex&Yacc parser – hence a LALR(1) grammar – and eases the reading of complex statements. The shared Lex parser of FL, FE and FCG can be made generic enough to allow small and coherent changes of the prefixes to be dynamically made by each user, within an input file. Then, the Yacc parser of FCG can be extended to parse FL, FE, CGIF, KIF and hence probably all well-known graph-based and logic-based KRLs. Section 4.3 presents a shared grammar in preparation of this. The parser for RDF/XML currently appears too different to be worth being integrated into this unique parser. The FE parser can then be independently extended to handle more English looking statements. Thus, the FS parser will continue to call different sub-parsers: the FE parser, the RDF/XML parser and another one for all the other languages. This shared parser will also take into account the selected (re-)presentation options, not just the selected language and used statement delimiters.

The current FL parser of WebKB-2 does not yet fully handle the '_', '.' or ':' before or after the parenthesis and square brackets (hence, the default order must often be followed), and expects parenthesis instead of brackets as context delimiters. Most of the other syntactic features presented in this document (and all of those presented for FE and FCG) are implemented.

2.1.1.7. User/category declarations in FL. In FL, information sources can only be declared in this rather peculiar way:

```
_user@thisKB  pm#instance:  anonymous_user@thisKB _[u]
                            spamOnly@phmartin.info _[pm pw001xyKtDq2k];
```

_user@thisKB is predefined in FS (since a type for all registered users is needed) and hence the parser interprets the context information as i) the *short identifier* for the newly declared user, followed by ii) the encryption of its password with the Unix function "crypt". Thus, the user declarations (which, like other knowledge representations, can be generated by the interfaces of WebKB-2) can be included in import/export files in a (at least minimally) secure way. Except in this special case, user identifiers must be declared before they are used.

If an identifier such as pm#thing has been declared, FS accepts `thing\pm' as an alternative identifier, and conversely. In both cases, `pm' must have been declared as a user. By default, the FS parser only accepts identifiers that are either i) predefined, ii) a registered user, iii) prefixed or postfixed as above indicated, or iv) common URIs (typically, URLs and email addresses; these identifiers have no author). By default, to prevent some lexical errors and then semantic problems, the FS parser does not accept identifiers to refer to different objects that are not related by an identity relation, and it does not accept the use of yet unknown identifiers, unless they are in a statement that declares them.

Declarations are made by relating the new identifier to an existing one by a relation of type pm#equivalent_object or pm#extended_specialization (and hence also any subtype of it, e.g., pm#instance or pm#subtype).

2.1.1.8. Some predefined types of relations from types.

```
pm#supertype_or_equal .(pm#type, pm#type)
  pm#supertype:  pm#relation_from_type  pm#generalizing_category,
  pm#subtype:  pm#supertype___strict_supertype  pm#direct_supertype___direct_strict_supertype,
  pm#inverse:  pm#subtype_or_equal,
  pm#equivalent_object:  rdfs#subClassOf  owl#subClassOf  ontolingua#subclass-of;
```

The Multi Source Ontology (MSO; the default ontology of WebKB-2) relates categories from many ontologies, including language ontologies such as RDFS, OWL and the Frame Ontology of Ontolingua [www-Ontolingua-FO 1994]. The above statement *defines* pm#supertype_or_equal as being equivalent to rdfs#subClassOf, owl#subClassOf and ontolingua#subclass-of, and *asserts* that these last three types are equivalent according to `pm' (before doing so, pm#type was defined as being equivalent to rdfs#Class, owl#Class and ontolingua#Class). These equivalence relationships may not be strictly correct if pm#equivalent_object is interpreted as the identity or equivalence relations of certain logics but the MSO can be complemented by other users of WebKB-2 to make this explicit. The use of relations such as pm#supertype_or_equal is discouraged by WebKB-2 because the "equal" part reduces possible validations. In FL, FE and FCG, many relation types such as the following ones are predefined to allow inferences or validations and have abbreviations:

1. '=' for pm#equivalent,
2. '<' for pm#greater_number_or_measure, pm#supertype and pm#generalizing_statement (the types of the connected objects permit to distinguish the relevant relation type),
3. '=<' for pm#greater_or_equal_number_or_measure and pm#generalization_or_equal,
4. '>' for pm#lower_number_or_measure, pm#subtype and pm#specializing_statement,
5. '>=' for pm#lower_or_equal_number_or_measure and pm#specialization_or_equal,
6. '=>' for pm#implication___logical_deduction___necessary_condition (supertype of pm#generalizing_statement),
7. '<=' for pm#sufficient_condition (supertype of pm#specializing_statement),
8. ':=>' for pm#definition_of_necessary_condition,
9. ':<=' for pm#definition_of_sufficient_condition,
10. ':<=' for pm#definition_of_necessary_and_sufficient_condition,
11. '-' for pm#inverse,
12. ':' for pm#instance, '^' for pm#kind, '!' for pm#exclusion and '/' for pm#closed_exclusion.

Only the first eleven abbreviations will be used in this document and the abbreviation of pm#inverse (i.e., '-') will only be used in Chapter 3. The relation type pm#definition will be used for relating a term to an informal definitions of it.

```
pm#definition .(pm#term, pm#description)
  >  pm#definition_of_necessary_condition  pm#definition_of_sufficient_condition,
     pm#definition_of_necessary_and_sufficient_condition;
```

2.1.1.9. The pm#supertype relation defines necessary conditions for the source term and sufficient conditions for the destination term. This is a consequence of i) the definition of the "supertype" relation with respect to the "instance" relation (in FE: ` `a pm#type *x has for pm#supertype a pm#type *y' iff `any (pm#thing that is pm#instance of *x) is pm#instance of *y' '), and ii) the meaning of the "instance" relation (any definition or universal statement associated to a type applies to its instances).

In WebKB-2, this consequence holds as long as no conflict is (manually or automatically) detected, even if the creator of the pm#supertype relation is not the same as the creator of the source and destination term. If a conflict is automatically detected between statements belonging to a same creator, the last entered statement (the one causing the conflict) is rejected. If a conflict is automatically detected when a creator connects a pm#supertype relation from/to at least a term that she has not created, this relation is rejected. If, when the creator of a term (say, pm#X) associates a definition or universal statement to pm#X, a conflict is automatically detected between the statements associated to

pm#X and statements associated to another term (say, pm#Y) from another creator (say `oc'), X or Y is "cloned", that is, its identifier is automatically changed (e.g., into oc#X) but its previous relations are kept (in order to keep its meaning) and the clone is connected to its source (e.g., oc#X becomes a supertype of pm#X). Core techniques for this cloning (i.e., in the previous example, for selecting an appropriate oc#X or oc#Y and connecting it to pm#X or pm#Y) are described in Annex 2 of [Martin, 1996] and are not repeated in this document. However, the conflict solving protocols (which rely or not on such a cloning) will be described in Subsection 2.2.5.
In WebKB-2, these rules apply to all the subtypes of the pm#extended_generalization relation type and hence also, symmetrically, to all the subtypes of the pm#extended_specialization relation type.

2.1.1.10. The pm#extended_specialization relation type. This type is predefined in FS and refers to relations of *"specialization" in its general sense* between terms (e.g., strings and category name/identifiers) or statements. This general sense of "specialization" is that *the destination node of this relation includes more information than the source node*. Extended specialization relations can be manually set. The 'ext-spec' graph matching operator (described in Section 2.4) permits to discover extended specialization relations between many kinds of (semi-)formal statements. The extended_specialization relation is a supertype of the (logical) generalization – the inverse of a specialization – which corresponds to a logical deduction. For example, generalizing an existential conceptual graph (or a logical formula) can be done by cutting any of its branches or generalizing any of the categories it contains. In WebKB-2, the 'gen' operator looks for recorded statements that generalize a query graph or, if the parameter is not a graph but a category, looks for its types and supertypes. Its inverse operator is 'spec'. The 'ext-spec' operator does not just look for types/statements that are more *constrained* (i.e., have more information) in a logical sense but in a general sense. For example, it considers that any context (e.g., using modalities or temporal relations) is a constraint, that the universal quantifier is more constraining than the existential quantifier, and that the identifier of a type is more constrained than a name of this type (since categories may share names but not identifiers). More generally, 'ext-spec' can also take into account manually set pm#extended_specialization relations between formal or informal terms. Table 2.1.1.10.1 shows some subtypes of pm#extended_specialization.
Here are examples of use. (The meaning of the '{(' and ')}' delimiters is explained in the next paragraph).

```
"animal related concept"
  .>  ("animal right"
        .>  (pm#right_of_an_animal  // < pm#right,  //commented since redundant with next line
               :<=> (pm#right  owner=> some pm#animal),
               //pm#owner=>  a pm#animal,  //implied by the definition of the previous line
               .>  (pm#right_of_every_animal
                      pm#owner=>  every pm#animal,
                      .>  (pm#right_of_any_animal  owner=>  any pm#animal)
                          (pm#right_of_Garfield  owner=>  Garfield)
                   ) ) );

pm#term
  >  {( (pm#informal_term  < pm#string)  (pm#formal_term  .> pm#informal_term) )}
     {( (pm#term_for_an_individual  .> pm#individual,
           >  {(pm#informal_term_for_an_individual  pm#formal_term_for_an_individual)} )
        (pm#term_for_a_type  >  {(pm#informal_term_for_a_type  pm#type)} )
     )};
     pm#individual  >  {(pm#formal_term_for_an_individual  pm#statement)};
       pm#formal_term_for_an_individual  >  pm#formal_term_for_a_statement;
```

Table 2.1.1.10.1. Some subtypes of pm#extended_specialization

```
pm#extended_specialization___extended_strict_specialization .(., .)
  pm#abbreviation:  '.>',
  pm#inverse:  (pm#extended_generalization  pm#abbreviation:  '.<'),
  >  (pm#extended_specialization_from_formal_or_informal_term .(pm#term, ?)
        >  (pm#type_specialization .(pm#type, pm#formal_term)  >  pm#instance   pm#subtype)
           (pm#term_specialization .(pm#term, ?)
              pm#inverse:
                (pm#term_generalization .(pm#term, pm#term)
                   > (pm#formal_term_generalization .(pm#formal_term, pm#term)
                       >  (pm#type_name .(pm#type, pm#informal_term)   <  pm#name) )
                     pm#string_generalization .(pm#string, pm#string) ))
                       //A string X generalizes a string Y if X is a regular expression that
                       //  i) describes Y if Y is not a regular expression, or
                       //  ii) describes more strings than Y if Y is a regular expression.
                       //The other case for a string X to be considered as a generalization
                       //  of a string Y if X has the same content as Y except for some spaces
                       //  at the beginning or end of Y (the user may decide which characters
                       //  should be considered as spaces; see Subsection 4.5.2 for details)
           pm#definition //.(pm#term, pm#description)
           pm#subdomain //.(domain, domain)  //declared in next subsection
     )
     (pm#extended_specialization_from_statement .(pm#description, pm#description)
        >  (pm#specializing_statement
              pm#abbreviation:  '>'  '<=',
              pm#inverse:  (pm#generalizing_statement  pm#abbreviation:  '<'  '=>') )
           (pm#corrective_specialization___corrective_restriction
              >  pm#corrective_existential_specialization)
           pm#overriding_specialization
     );
```

2.1.1.11. Introduction to collections.

```
pm#thing
  >  {(pm#situation  pm#entity)}       {(pm#thing_playing_some_role  sowa#independent_thing)}
     {(sumo#physical  sumo#abstract)}  {(pm#indivisible_thing  pm#divisible_thing)}
     {(pm#individual  pm#type)}        {(dolce#particular  dolce#universal  dolce#world)}
     {(sowa#continuant  sowa#occurrent)}           {(cyc#partially_tangible  cyc#intangible)}
     {(cyc#temporal_thing  pm#non-temporal_thing)} {(cyc#partially_intangible  cyc#tangible)}
     {(pm#domain  pm#thing_that_is_not_a_domain)}  {3D#thing 4D#thing},
  pm#closed_exclusion:  owl#nothing,
  =  owl#thing  cyc#thing  akts#thing  sumo#entity  sowa#entity  rdfs#resource;
```

In FL, FCG and FE, curly brackets are used for delimiting collections. By default, their type is pm#exclusive_AND-set (the members of a set of that type are "exclusive" according to the pm#subtype relation and the pm#equivalent_object relation; exclusive types cannot have common subtypes or instances, and other exclusive objects cannot be identical or equivalent; the type pm#part_exclusive_AND-set is needed to specify disjoint types according to the pm#part relation, i.e., to specify that the objects cannot have common parts). Thus, given the default parsing rules, the next two statements are equivalent.

```
  pm#thing >  {3D#thing,  4D#thing};  //the "," is optional (see Table 4.3.7)
  pm#thing >  {3D#thing __[pm]   4D#thing __[pm]}_[pm#kind: pm#exclusive_AND-set _[pm]]
```

By default, a collection is "distributive", i.e., each of its (implicit or explicit) members is an individual source/destination of the relations connected to the collection. The distributive, collective or cumulative interpretation of a collection may be specified using the keywords coll and cuml in the context of the relation. For example, in the

next statement the relation of type pm#collection_size can be used because a cumulative interpretation of the set is specified.

```
{3D#thing  4D#thing}_[pm#collection_size: 2 __[. cuml -> ., * <- .]];
```

Thus, by default, a set of types refers to a (non necessarily complete) type partition. To precise that a set is complete with respect to a certain relation, the keyword `complete' can be used in the cardinality/quantifier part of the context of this relation. Alternatively, if within a statement (i.e., until its final ";") the set is complete with respect to all the relations it is connected to, the "{(" and ")}" delimiters can be used as an abbreviation. As an example for this, the next two statements are equivalent.

```
pm#thing  > {pm#situation  pm#entity} __[. -> . complete];
pm#thing  > {(pm#situation  pm#entity)};
  //Note: when the destination set of a relation of type pm#subtype or pm#part is complete
  //      this set represents the type of the source node of the relation.
```

By default, the creators of the relations to the types inside a partition are the same as the creator of the partition but different creators may also be explicitly specified. The creator of the partition is the creator of the exclusion relations between the types in the partition. In a subtype partition of a type X, at least one type must be a direct subtype of X. Two partitions of X can group nearly the same types but can still be both required to express the specializations of X according to two different viewpoints. This viewpoint can be expressed in the context of the partition, as in the following example:

```
pm#thing
  >  {(pm#indivisible_thing  pm#divisible_thing)}
     _[pm#relation_on_which_the_source partition_is_based: pm#part __[. cuml -> .];
```

2.1.1.12. Introduction to some top-level concept types. The above subtype partitions of pm#thing lists some important top-level distinctions:

- ***situations/entities***, i.e., states or processes vs. anything else; this distinction is more general – and hence less precise but easier to use – than DOLCE's distinction between occurrents and perdurants; it is also closely related to – but not a generalization of – John Sowa's distinction between occurrents and continuants and Matthew West's distinction between 3D and 4D things,
- ***role-types/natural-types***,
- ***individuals/types***, i.e., 0-order categories vs. 1st/2nd/.../Nth-order categories; the distinction between particulars and universals can be seen as a generalization of this individuals/types distinction but from the viewpoint of a world-based modal logics.

In this document, the terms "objects" or "categories" refer to types (concept/relation types) as well as individuals (including statements and collections). The term "node" refers to the syntactic structure composed of an object and its associated quantifier if it has one (examples of concept nodes in FE and with `wn' referring to WordNet, `some wn#cat', `at least 2 wn#cat', `at least 2 wn#cat', `the wn#cat "Tom"' and `wn#cat').

2.1.1.13. Situations (states or processes) and descriptions. A (real or imaginary) situation is described (represented) by a statement (description, i.e., a definition, a belief or a preference). In theory, a type of "relation from a situation" (i.e., a relation type having a signature which specifies that a relation of that type can only have for source a node of type pm#situation) cannot have for source a description, and conversely, a type of "relation from a description content/medium/container" cannot have for source a situation. In both cases, one should theoretically explicitly use an intermediary relation such as pm#descr to connect the situation node to the description node, and indeed one must do so in most languages, e.g., Conceptual Graphs. Since this is tedious, does not bring any information and leads to bigger representations, ***such relations can be left implicit in all sub-languages of FS, and (formal or informal) statements do not have to be typed***: indeed, when needed, for each relation connected to the statement, the relevant type (situation or description) can automatically be inferred from the signature of this relation.
As shown by the next statements, in the MSO there are types for description content (e.g., belief, narration), description mediums (e.g., languages, abstract data types) and description containers (e.g., documents), and one type that generalizes all of them. The main reasons for that type are: i) there are many types of relations that apply to description content as well as description mediums or containers, ii) distinguishing the actual type of such objects is tedious, sometimes difficult and hence leads users to make different representation choices (thus, this is not a task that people should have to do), and iii) many ontologies do not distinguish between these different types. However, as detailed in the next chapter, the MSO also has relations specific to each type.

```
pm#situation  pm#description_content/medium/container:
                                    pm#description_content/medium/container __[*<->*];
pm#description_content/medium/container  >  {pm#description  pm#description_container};
  pm#description  >  pm#description_content  pm#description_medium;
```

2.1.1.14. Things that can be seen as relations. In all sub-languages of FS, certain concept types can be used in binary relation nodes as if they were binary relation types, as illustrated by the above statement about pm#situation. This is a rare but not exceptional feature (e.g., it has been used in Ontoseek [Guarino et al., 1999]) since it is handy and, as later detailed, avoids to "duplicate concept types" in the relation type hierarchy and thus eases knowledge sharing. In this document, the expression "duplicating types" (duplicating types elsewhere in the ontology) means creating other types with similar names and with structurally similar relationships. To remind the special meaning of this expression, the word 'duplicate' will always be single quoted in this document. The conditions for a concept type X to be used as a relation type are that: i) X is declared as subtype of pm#thing_that_can_be_seen_as_a_relation (which is predefined in FS), ii) the destination node is of type X and, iii) if X has an associated signature, the source node must conform to that signature. The following statements show some of the subtypes of pm#thing_that_can_be_seen_as_a_relation and an example of a relation signature associated to a concept type (although this was not necessarily in this particular case). When exporting to languages that do not have such a feature, for each subtype of pm#thing_that_can_be_seen_as_a_relation, the declaration of a corresponding relation type (including the relation signature) can be generated (see Subsection 4.2.13 for a specification of this in KIF). This corresponding relation type may also be explicitly declared, e.g., in this document, pm#definition is declared as a relation type in Paragraph 2.1.1.10 (since its supertype is a relation type) while pm#Definition is declared as a concept type in Table 2.1.3.3 (since its supertype is a concept type). In such a case, a relation of type pm#manually_set_corresponding_relation_type must be set between the concept type and the corresponding relation type. A relation type cannot be given a signature different to the one given to its corresponding concept type, and conversely. If, in a statement, the signature of the used relations are consistent with the use of both a concept type and its corresponding relation type, by default the concept type is selected. A way to lift the ambiguity is to use more specific relations with this used type.

```
pm#thing_that_can_be_seen_as_a_relation
  <  pm#thing_playing_some_role,
  >  pm#thing_that_can_be_seen_as_a_function  pm#attribute_or_quality_or_measure
     pm#description_content/medium/container  pm#entity_playing_some_role
     wn#relation;
pm#description_content/medium/container .(situation, pm#description_content/medium/container);
```

2.1.1.15. Signatures of processes in order to make certain relations implicit. Types subtypes of pm#process cannot be subtypes of pm#thing_that_can_be_seen_as_a_relation but can have some kind of signature to allow their use in relation nodes. Here is an example of declaration (note: '*x' and '?x' are equivalent ways to declare or refer to a variable named "x" and hence to an unknown object which, if no quantifier is explicitly given, is existentially quantified; in FL, FCG and FE, the scope of a variable covers the whole statement in which it has been declared, including its context; in other words, the scope ends with the ';' ending the statement):

```
wn#indexing .(pm#input: ?x, pm#output: ?y)
  input:  pm#thing ?x,  //anything, not just a description_content/medium/container
  output:  pm#description ?y;  //an indexing output (an index) is a description
```

This declaration makes the following two statements equivalent and permit the first one to be accepted:

```
"Tom is on a mat"  wn#indexing:  "Tom" "mat";
"Tom is on a mat"  input of:  (a wn#indexing output: "Tom" "mat");
```

The default signature for processes is ".(pm#object: *x, pm#result: *y)". Since pm#input is subtype of pm#object and pm#output is subtype of pm#result, the above signature could have been omitted, and the following two statements would have been equivalent.

```
"Tom is on a mat"  wn#indexing:  "Tom" "mat";
"Tom is on a mat"  object of:  (wn#indexing  result:  "Tom" "mat");
```

2.1.1.16. Special keywords in relation nodes/contexts: `more'/`less', `main', `1st', `2nd', ..., `last'. Here are self-explanatory equivalent examples about the use of `more'/`less'.

```
pm#informal_term  has less pm#constraint_on_meaning than: any pm#formal_term;
pm#informal_term  less pm#constraint_on_meaning than: any pm#formal_term;
pm#formal_term  more pm#constraint_on_meaning than: any pm#informal_term;
```

Here are self-explanatory equivalent examples about the use of `main'.

```
pm#process_with_a_process_as_main_input  main pm#input:  process;
pm#process_with_a_process_as_main_input  pm#input:  process __[main];
```

Here are self-explanatory equivalent examples about `1st', `2nd', ..., `last' and the use of a variable as a member index (here, the variable is `?Nth'; unless quantified elsewhere, such variables are implicitly existentially quantified).

```
pm#changing_a_tire .(pm#agent: *x, pm#object: {wn#tire *t1, wn#tire *t2})
  1st  pm#subprocess:  pm#unscrewing_all_screws_of_a_tire .(pm#agent: *x, pm#object: *t1),
  (?Nth != last) pm#subprocess:  pm#disposing_of_a_tire .(pm#agent: *x, pm#object: *t1),
  last pm#subprocess:  pm#screwing_all_screws_on_a_tire .(pm#agent: *x, pm#object: *t2);
pm#changing_a_tire .(pm#agent: *x, pm#object: {wn#tire *t1, wn#tire *t2})
  pm#subprocess: { pm#unscrewing_all_screws_of_a_tire .(pm#agent: *x,pm#object: *t1)__[?ct->?]
                   (pm#disposing_of_a_tire .(pm#agent: *x, pm#object: *t1)
                      (?Nth != last) pm#member of: ?subprocesses ) __[?ct->?]
                   pm#screwing_all_screws_on_a_tire .(pm#agent: *x, pm#object: *y) __[any->?]
                 }_[pm#kind: pm#sequence] __[any ?ct->?subprocesses];
  //note: the use of ?ct is optional (destinations in a collection must have the same source)
```

The use of these keywords avoids the need to define relation types such as pm#main_input or pm#less_constraint_on_meaning_than. The type pm#constraint_on_meaning is actually a subtype of wn#precision which is an indirect subtype of pm#attribute_or_quality_or_measure which is a subtype of pm#thing_that_can_be_seen_as_a_relation. Relations such as `less pm#constraint_on_meaning than' permit to order types of descriptions (e.g., terms and statements) with respect to how well-defined their meaning are. This permits to order indexing processes (creating such descriptions) with respect to precision thanks to the rule "the more precise the indexation, the more precise the indexing process that produces it". The representation of this rule in FL and FE will be given later.

2.1.1.17. The pm#part relation type. This type refers to various kinds of binary mereological relations. It only connects objects of the same type. Hence, it is not necessary to declare a specific mereological transitive relation for every concept type: pm#part can be used directly. Below are some definitions. Table 3.1.13.2 categorizes additional mereological relation types.

```
pm#part .(*x, *y)
  pm#relation_source:  (*x  pm#kind:  *t __[.<->?]) __[.->?],
  pm#relation_destination:  (*y  pm#kind:  *t __[.<->?]) __[.->?],
  := [ [?x direct_part ?y] or: [?x part: ?y] ];  //or:
//:= [?x (direct_part: a description_medium)* direct_part: ?y];
  <  pm#part_or_equal   //the next 3 lines will be explained later
     (pm#relation_instance_of_transitiveProperty_unless_directly_overrided
        kind:  owl#transitive_property,
        type:  pm#type_instance_of_a_certain_second_order_type_unless_directly_overrided),
  >  {pm#sub-situation  pm#spatial_part  pm#sub-attribute  pm#sub-description  pm#subdomain}
     pm#sub-collection  (pm#member  >  pm#partner);
```

Example of subtypes:

```
pm#sub-situation .(pm#situation, pm#situation)
  >  pm#subprocess  pm#substate;
pm#spatial_part .(pm#spatial_object, pm#spatial_object)
  >  {(pm#physical_sub-area .(pm#physical_entity, pm#physical_entity)
       pm#non-physical_sub-area .(pm#spatial_object, pm#spatial_object) )}
     (pm#physical_part .(pm#physical_entity, pm#physical_entity)
        >  pm#matter___stuff
           (pm#physical_sub-area
             >  pm#attached_physical_component  pm#removed_physical_piece) );
```

The relation types pm#relation_source and pm#relation_destination are predefined in FL, FCG and FE. The above listed examples of subtypes of pm#part already include more distinctions than Rogers and Rector mention in [Rogers & Rector, 2000] to have found in the literature: component, stuff, portion, area, member, partner and piece.

2.1.1.18. Mixing FL, FCG and FE.

```
[_ pm, pm#language: pm#FCG _[@<->?]] [2 wn#cat, pm#place: (a wn#mat, wn#color: some wn#red)];
```

The above formal sentence begins with a context which asserts that the contextualized statement has `pm' for author and is written in FCG. The above '@' quantifier specifies that the pm#language relation has for source the *abstract/concrete model* (a description medium) of the contextualized node, not the description content of this contextualized node. Here are equivalent declarations of pm#language with a signature that permits to omit the _[@<->?] above part since the '@' is specified:

```
pm#language .(pm#description_content/medium/container @ [0..*], pm#description_medium [1]);
pm#language .(pm#description_content/medium/container @ -> pm#description_medium);
                                                        //'->' indicates a functional relation
```

This facility also applies to the content of strings, e.g., the content of the following string is declared as being interpretable in Perl. In this example, the content is a double quoted that contains a regular expression. This expression is declared as being interpretable in Perl.

```
$("<script .* language='[A-Za-z0-9/_-]*'.*>")$ _[pm#language: Perl]
  //this Perl expression matches the content of the string "<script language='FS'>"
```

Here are equivalent FE statements for the FL statement at the beginning of this paragraph:

```
[_ pm, pm#language: pm#FE] [2 wn#cat have for pm#place a wn#mat which has for
                                                              wn#color some wn#red)];
[_ pm] `2 wn#cat have for pm#place a wn#mat with wn#color some wn#red';
`2 wn#cat have for pm#place a wn#mat which has for wn#color some wn#red'_[pm];
`2 wn#cat have for pm#place a wn#mat with wn#color some wn#red'_[with pm#author pm];
```

These examples illustrate that FCG, FE and FL can be mixed, that they have common features, that FCG and FE have a similar structure but different syntactic sugar and that the delimiter of FCG statements are square brackets while those of FE statements are the single backquote and the single right quote.

FE and FCG can use multiple types in a concept node. The objects referred by such a node must conform to each of these types. As illustrated by the next example, an exception is that when at least one of the types is not subtype of pm#attribute_or_quality_or_measure the objects do not need to conform to those types that are subtypes of pm#attribute_or_quality_or_measure. Instead, in such a case, the objects must be considered as being related to instances of those types by an pm#attribute_or_quality_or_measure relation. The following example of equivalent FCG statements illustrate this. The rationale for this rule is given in Paragraph 2.3.1.2.

```
[a wn#red wn#mat];
[a wn#mat, pm#attribute_or_quality_or_measure: some wn#red];
[a wn#mat, wn#color: some wn#red];  //since wn#color is a supertype of wn#red and an attribute
[a wn#mat, wn#red___redness: some wn#red];
```

2.1.1.19. Parsing indications/directives.

```
[_ parsing][(pm#formal_term  pm#part of: pm#relation_node) pm#default_creator: pm __[any->?]];
[_ parsing][(pm#formal_term  pm#part of: pm#concept_node)  pm#default_creator: pm wn];
[_ parsing][pm#term_being_declared  pm#default_creator:  pm];
[_ parsing][pm#new_term  pm#default_creator:  pm];
[_ pm#kind: pm#parsing_macro][pm#statement  pm#default_creator: pm];
```

The above three statements are indications to the FS parser that (respectively)

- any formal term that is found in a relation node (i.e., in a place where a relation type is expected) has for default creator `pm' and hence whenever in such a place the parser encounters a formal term that has no explicit creator (i.e., not a string, a URI, a prefixed/postfixed identifier or a predefined term), it should check if `pm' has created a category which has such a term as key name (i.e., forming an identifier when prefixed/postfixed by `pm') and which can act as a relation type; if the parser has found such a category, it should use it, otherwise it should deliver an error message;
- any formal term found in a concept node with no explicit creator has `pm' or then `wn' for creator (a term from `pm' should first be tested to see if it conforms to the constraints associated to this concept node, e.g., the constraints set by the signatures of the relations connected to this concept node; then, if no adequate term from `pm' can be found, a term from `wn' should be tried);
- any new term (formal or not) being the subject of a category declaration/definition and without explicit creator should be given `pm' as creator (only one default creator can here be specified as destination of the pm#default_creator relation);
- any new term (formal or not) without explicit creator should be given `pm' as creator and declared if it has not yet been declared; informal terms can have associated creators too, e.g., pm#"cat" and pm#"this is an informal term/statement";
- any statement without explicit creator is from `pm'.

All the terms in the above statements are predefined in FS; indeed, the FS parser needs to know their special meanings to take the parsing indications into account. These terms are part of the presentation/parsing ontology that is proposed in Chapter 4. The keyword `parsing' is an abbreviation for "pm#kind: pm#parsing_macro". Parsing indications are particular kinds of defeasible statements, i.e., statements that can be overrided by other statements. The FS parser accepts statement overriding only if those statements have been marked as defeasible (e.g., via the keyword `defeasible' or `parsing' in their context; properties associated to a type "X" subtype of pm#type_instance_of_second_order_type_unless_directly_overrided can also be overrided by the subtypes of "X" as in [Y supertype: X __[.->0]]). Overriding a statement does not override any other statement, even the more specialized statements. This is why the above 4th parsing indication does not override the above 2nd parsing indication. To ease the reading and writing or generation of informal statements, lexical parsing indications/abbreviations are also taken into account in the content of strings that are double quoted or delimited by "$(" and ")$", as for the expansion of variables. The current FS parser of WebKB-2 does not yet accept parsing indications written in FL but accepts and takes into account many predefined abbreviations for them.

2.1.1.20. Relations from/to quantified categories.

```
  cat  place:  (mat color: red __[?m<->?]) __[2->?m]; //a non-free variable in a cardinality is
  cat  place:  (mat color: some red) __[2->?];        //  by default existentially quantified
  cat  place _[2->?]:  (mat color: red);
  2 cat place: (a mat color: some red);
```

The above equivalent statements are translations in FL of "there are 2 cats on a red mat", with the previous parsing indications taken into account, hence without the 'pm#' and 'wn#' prefixes. Since the concept node of type 'mat' is existentially quantified and given the variable ?m due to `__[2->?m]' for the `place' relation, there is no need to repeat these existential quantifier and variable for the `color' relation.

Assuming that the user `oc' has stated that "any cat is on a red mat" and hence has used '_[any->?]' instead of '_[2->?]', the two statements could be merged into any of the following equivalent statements:

```
  cat  place:  (mat color: some red) __[2->?, any->? _[oc]];
  cat  place:  (mat color: some red) __[any->? _[oc], 2->? _[pm]];
  cat  place:  (mat color: some red) __[oc, 2->? _[pm]];
```

Expressing that "there are 2 cats on a mat and that any mat may be on a floor" can be done in the following equivalent ways.

```
  cat  place:  (mat place: floor __[any->?]) __[2->?];
  cat  place:  (. mat place: floor __[any->?]) __[2->?];
  cat  place:  (. mat place: a floor) __[2->?];  //here, the '.' is not optional
  2 cat  place:  (. mat place: a floor);
  2 cat  place:  a (. mat place: a floor);
//2 cat  place:  a (mat place: a floor);  //this one uses a lambda-abstraction and states that
                                          //  2 cats are on a mat-on-a-floor
```

The `.' permits to specify that a concept node does not have the quantifier or implicit/explicit variable that it was given by a previous relation. This feature permits to avoid specifying quantifiers that are the default ones. Subsection 4.2.1 presents complementary examples.

These last examples show how FL permit to express in one concise statement what would need to be expressed in several statements (and sometimes long statements) in other notations. This eases knowledge comparison, especially knowledge from various users.

2.1.1.21. Contextual relations from/to (descriptions of) processes. Consider the following FCG statements.

```
  [a sitting, agent: a cat, time: 21/01/2009];
  [ [a sitting, agent: a cat], time: 21/01/2009];
```

In WebKB-2, these statements are considered equivalent. This can be seen as an extension of the above described feature for i) non-distinguishing contexts on situations from contexts on descriptions, and ii) allowing the scope of a variable to be usable until the end of the whole statement. Without this extension, the first statement asserts that there exists a sitting cat and that it is sitting on the 21/01/2009. With this extension, like the second statement, it asserts that on the 21/01/2009 there exists a cat and that it is sitting. This extension means considering that "truth restricting relations from/to a context" connected to a process also apply to all the objects directly or indirectly connected to this process within the same context (i.e., in FCG, within the nearest pair of square brackets). Without this extension the two above statements would not be automatically comparable.

2.1.1.22. Complex definitions. Below are three equivalent representations (one in FE and two in FL) of the rule "the more precise the indexation, the more precise the indexing process that produces it". For clarity purpose, the authors of certain categories are made explicit even though the content of the MSO and the above parsing indications permit to leave them implicit.

```
`any `wn#indexing *i with result a pm#description *y' has less wn#precision than
 any `wn#indexing *i2 with result a pm#description that has less wn#precision than *y';
wn#indexing
  ouput:  pm#description ?y,
 less wn#precision than: any (wn#indexing output: (a pm#description less wn#precision than: ?y);
wn#indexing .(pm#input: ?x, pm#output: ?y)
 less wn#precision than: any (wn#indexing output: (a pm#description less wn#precision than: ?y);
```

Given this rule and the following statements in FL, the conclusion in FE is `any pm#indexing_by_an_informal_term has less pm#precision than any pm#indexing_by_a_formal_term'.

```
pm#constraint_on_meaning  <  wn#precision;
pm#informal_term  has less pm#constraint_on_meaning than:  pm#formal_term __[any<->any];
pm#indexing_with_a_formal_term  result:  a pm#formal_term;
pm#indexing_with_an_informal_term  result:  a pm#informal_term;
```

However, WebKB-2 is not yet able to draw such conclusions. Similarly, it cannot see the logical equivalence between the above FE representation of the rule and this one:

```
` `a pm#description *y has more wn#precision than a pm#description *y2' has for pm#implication
  `a wn#indexing with output *y has more wn#precision than a wn#indexing with output *y2' ';
```

Unlike formulations via definitions or using universal quantifiers, formulations using pm#implication between statements do not have a structure that can be directly translated into a frame-like notation such as FL and hence be accepted by frame-based systems or description logic based systems. On the other hand, theorem provers would have no problem translating between the two formulations. Hence, for knowledge sharing purposes, it seems better for information providers not to use the last kind of formulation (in other words, the first kind can be seen as the normalized formulation).

2.1.1.23. Syntactic sugar for collections, to ease the reading of argumentation structures. FL can be used for representing argumentation structures or "semi-formal discussions": formal or informal sentences created by various persons and related by argumentation relations, specialization relation and corrective relations. It permits to clearly distinguish "a relation Y on the destination sentence of a relation X" from "a relation Y on a relation X" (in the last case, nothing is said on the destination of the relation X). Few argumentation systems make such a distinction and, more generally, accept meta-statements. ArguMed [Verheij, 1999] is one of the exceptions. Hence, most of them force incorrect representations of argumentation structures. This is clearly true of systems restricted to the famous argumentation schema of Toulmin, as for example noted by [Newman & Marshall, 1992]. Even fewer provide a textual notation that is not XML-based. Such a notation is nonetheless necessary whenever the use of an XML parser, editor or viewer is impossible or not desirable (this is for example the case in many text-based email editors, in text-based browsers, and in PDF or HTML documents). Finally, FL is not restricted to argumentation structures nor informal sentences. It permits to use them only when it is not handy to use more precise representations. It is also not restricting the user to a predefined ontology of only a few types of concept or relations.

To ease the reading of "joint arguments/objections" (e.g., a rule and its premise), instead of using the normal syntactic sugar for a collective AND-set ("{x, y, ...} __[coll]") dashes are used as shown by the next example. This example mentions three sources: `pm', `fg' and `tbl' (for Tim Berners-Lee). The identifier `pm/tbl' refers to `tbl' as interpreted by `pm': `pm' is the creator but `tbl' is the initial source. Thus, '__[pm/tbl]' is an abbreviation for '__[pm, source: tbl]'. Such an abbreviation will also be used in later subsections. The informal sentences in this example have no associated creator.

Table 2.1.1.22.1. Small semi-formal discussion about
the interest of XML-based languages for knowledge sharing

```
"knowledge_sharing_with_an_XML-based_language is advantageous"
   .<  ("knowledge_sharing_with_an_XML-based_language is possible"
           .<  knowledge_sharing_with_an_XML-based_language  __[pm]
       ) __[pm],
   argument: – "XML is a standard" __[pm]
             – ("knowledge_management_with_classic_XML_tools is possible"
                  corrective_specialization:
                    "syntactic_knowledge_management_with_classic_XML_tools is possible"__[pm]
               )__[pm],
   argument: "the use of URIs and Unicode is possible in XML"
             __[fg, objection: "the use of URIs and Unicode can easily be made possible in
                                most syntaxes" __[pm/tbl]
               ],
   objection: – ("the use_of_XML_by_KBSs implies several tasks to manage"
                    argument: "the internal_model_of_KBSs is rarely XML" __[pm]
                 )__[pm],
              – ` "an increase of the number of tasks *t to_manage" has for consequence
                   "an increase of the difficulty to develop software to manage *t" '_[pm]
                 __[pm],
   objection: – "knowledge_sharing_with_an_XML-based_language will force many persons
                  (developers, specialists, etc.) to learn how to understand
                  complex_XML-based_knowledge_representations" __[pm]
              – ("understanding complex_XML-based_knowledge_representations is difficult"
                   argument: "XML is verbose" __[pm]
                 )__[pm];
```

2.1.1.24. Category naming style and lexical facilities. For readability purposes and to ease translations into FE or other kinds of controlled English, the users of WebKB-2 are discouraged (e.g., via warnings) to declare new identifiers that do not follow the English spelling and capitalization of the words used in the identifiers. The "loss-less category naming style" is to re-use those words as such and separated by '_' or '-'. Unfortunately, the currently most commonly used naming style – which is called the "W3C category naming style" in this document – is to follow the Intercap style (as in the identifier rdfs#subClassOf) and use an uppercase character for the first letter of the identifier if it is not for a relation type (as in rdfs#Resource). Indeed, this last style became a well-followed convention when it was adopted in the RDF examples given by the W3C and, much earlier, in the Ontolingua library [www-Ontolingua-library 1994]. However, when this naming style is used, an English dictionary is needed for recovering the exact case for the first letter of the English words composing an identifier and also for isolating these words when several uppercases are used consecutively. RDFS proposes the rdfs#label relation to permit knowledge providers to specify the source English words but this is understandably rarely used and not used in a uniform way.
For the MSO to have a coherent naming style, each new identifier following the "W3C category naming style" is converted into the "loss-less category naming style" whenever i) there is a way to quickly convert it back to the "W3C category naming style" for knowledge export purposes, and ii) this does not lead to a lexical conflict with another already declared category from the same source/user. By default, for search purposes, WebKB-2 first converts the given words or category identifiers in lowercase and ignore the '_' or '-' characters as well as the final 's'; if this leads to lexical ambiguities (i.e., several possible identifiers), the given spelling and capitalization are used. In an expression or statement in FE or FCG, it is possible to add a final 's' to an unprefixed category identifier when it is used with a numerical quantifier, as in `75% of cat**s**' and as long as there is no lexical ambiguity. However, the use of this last facility is generally avoided in this document.

2.1.1.25. Generation of categories for organization purposes. Consider the sentence "governments should enforce animal rights" and its normalized representation in FE: `any `enforcement with object `any `right with owner an animal' ' ' should have for agent any government'. In a KB storing knowledge about animals or rights, several statements may be recorded about notions such as "animal rights", "enforcement by government", "enforcement of animal rights" or "enforcement of animal rights by governments". Then, for knowledge organization purposes – and hence for knowledge sharing, comparison and retrieval purposes – it is interesting to have such notions inserted in the specialization hierarchy and such statements accessible from these represented notions. Ideally, the representation of the sentence and the insertion of the notions would be (semi-)automatic but WebKB-2 does not yet perform such tasks. However, it can propose identifiers for such notions and the MSO includes the pm#extended_specialization relation type which permits to organize them and relate them to their associated statements. Here are examples.

```
"right"
  .> (wn#right
       > (right_with_an_owner
           > (right_with_owner_an_animal
               .> (`enforcement with object `any `right with owner an animal'''
                           and with agent any government'
                     .> `any `enforcement with object `any `right with owner an animal'''
                           should have for agent any government') ) ) );
"enforcement"
  .> (wn#enforcement
        > (enforcement_with_an_agent
            > (enforcement_with_agent_a_government
                .> (enforcement_with_agent_every_government
                     .> (enforcement_with_agent_any_government
                          .> `enforcement with object `any `right with owner an animal'''
                                      and with agent any government') ) ) )
          (`enforcement with object some thing'
            >  (`enforcement with object some right with owner an animal'
                  .> (`enforcement with object any `right with owner an animal''
                       .> `enforcement with object `any `right with owner an animal'''
                                  and with agent any government') ) );
```

The use of a unique relation type such as ".>" is handy for certain presentation purposes but more specialized relation types can be used, e.g., the pm#statement_directly_using_this_term relation type to connect the type
`enforcement with object `any `right with owner an animal'" and with agent any government'
to the statement
`any `enforcement with object `any `right with owner an animal'" should have for agent any government'.
Which relation type is used for presentation purposes depends on presentation options.
For organization purposes, the most important and easiest-to-generate categories in these examples are pm#right_with_an_owner, pm#enforcement_with_an_agent and `enforcement with object some thing'. Indeed, such categories are based on basic types of relations, permit to partition the specializations of a type, and subtyping them is only needed if there are many specializations.

2.1.2. Representing and Avoiding to Represent or Organize Subject Areas

After discussing various topic hierarchies (subject hierarchies) for document classification purposes (e.g., the Dewey Decimal System, LC, COLON, Bliss classifications, thesauri, subject headings, taxonomies), [Welty & Jenkins, 1999] concludes that placing a topic (subject area) into a topic hierarchy is quite ***arbitrary (in the sense that it depends on each person's goals, knowledge and preferences)*** and that the subtopic relation of such hierarchies is not a specialization relation (and is not even always transitive) but can be seen as a union of various relations. To bring a bit more determinism to the manual process of classifying documents (or other things such as persons' interests or competencies) and/or their related topics while still only relying on relations between topics, i.e., without representing (some of) the statements that the books contains or that the topics covers, Welty & Jenkins propose to replace the classic (and rather meaningless) subtopic relation by a "located-under" relation based on five mereo-topological relations that should be distinguished: "contains" (a transitive and specialized kind of "partOf" relation between topics), "overlaps", "borders", "near" and "far". They give axioms to relate these six relations.

Welty & Jenkins see topics as individuals. Indeed, relations between "shoe" and "heel" or "shoe-making" are clearly not subtype relations. However, Welty and Jenkins but do not precise the nature of these individuals. The MSO categorizes topics as "fields of study" (e.g., such as Physics) and, more generally, as particular kinds of sets (like theories and ontologies) which include i) categories for processes and things related to them, and ii) descriptions associated to these categories, processes and other things. Indeed, with such an interpretation, if these statements were represented it would be possible to derive the "contains" relations from the inclusion relationship between the sets and the specialization relations between the things contained in those sets. As a reference to this mix of partOf and specialization, in the MSO the "contains" relation (which has for identifier pm#subdomain) has for abbreviation ">part".

```
subdomain___true_subdomain (domain, domain)
  abbreviation:  ">part",
  <  subdomain_or_equal  part;
   //extended_specialization //this was already stated as a subtype in the previous subsection
```

Most modern representations for topic hierarchies do not specify whether a topic is an individual or a type. For example, SKOS (Simple Knowledge Organization System [www-SKOS, 2009]), the ontology recommended by the W3C "to represent thesauri, classification schemes, subject heading systems and taxonomies within the framework of the Semantic Web" defines the type skos#Concept (topics must be instances of that type but nothing prevents topics to be themselves types) and relations such as skos#broader and skos#related which connect instances of skos#Concept. Another example is the "Semantic Web Topics Ontology" of ISWC 2006 [www-SWTO 2006] which defines and uses relations such as "topic_subtopic", "topic_requires", "topic_relatedTo" and "topic_relatedProjects" between instances of types "Topic", "Technique" and "Project".

In any case, representing topics and subtopic relations between them – instead of representing relations between processes and the things they use or that used them – is detrimental for knowledge sharing and retrieval (KS&R). First, because it leads to partially 'duplicate' the representations of those processes, things and relations into those of topics and their relations (these two kinds of representations have similar names but are unconnected). Second, the subtopic relation is semantically extremely poor and hence placing a topic into a topic hierarchy is quite arbitrary and cannot be much checked or guided – the slightly more restricted "contains" relation of Welty & Jenkins does not really change this point.

To sum up, since "true" fields of study (e.g., Physics) – or subkinds of them (e.g., Knowledge Engineering which may be seen as a mere set of processes) – exist, they should be represented as such but, for KS&R purposes, ***other things should not be represented as topics or fields of study*** (in the MSO, the term for such "areas of interest" is pm#domain). A relation such as pm#domain_object is needed to connect a domain to the things it includes (relation types such as "topic_relatedProjects" are not needed). Since (descriptions of) processes permit to relate most of those

things and since specialization and subprocess relations between process categories are far less arbitrary than subtopic relations, ***the backbone for the representation of the content of a domain should be an ontology of processes in this domain, and it is not necessary (and hence, to avoid 'duplications', not advised to) relate a domain category to more than a few top-level process categories*** ("top-level" according to the specialization and subprocess relations).

However, what to do to integrate existing ontologies that use topics? For example, how to extend the MSO with the ReSIST ontology [www-ReSIST, 2006]? This ontology describes top-level concepts related to dependable and secure computing by reusing the Reference Ontology of the AKT project [www-AKT, 2006]. AKT is a very well funded Semantic Web project aimed to ease knowledge acquisition and retrieval but its Reference Ontology is small (a hundred of categories), not normalized and includes types for some fields of computer sciences and permits to reuse those of the ACM (Association for Computing Machinery). In accordance with the principles described in this document, to integrate the ReSIST ontology, the MSO i) integrates the (useful part of the) AKT Reference Ontology, ii) represents the ACM thesauri (about 1500 categories), iii) integrates the ReSIST ontology while only correcting the semantically incorrect representations (e.g., the MSO represents the fields of research as individuals instead of types and replaced their subtype relations by pm#domain relations), and iv) generalizes most of the new types from ReSIST, AKT or ACM with existing types in the MSO, essentially types from the WordNet ontology. Regarding the categories from ReSIST, the result is described in the MSO input file titled "The ReSIST ontology" [www-ReSIST-in-FL, 2008]. The following table gives a typology of domain-related concept/relation types that were used for these three integrations. (Note: the relation types that are instances of pm#non-directly_usable_type cannot be used with directly in relation nodes, only their subtypes can).

Table 2.1.2.1. Typology of some domain-related concept/relation types

```
domain  <  non-spatial_collection___collection_of_categories_or_statements,
  >  wn#field_of_study  wn#theory,
  =  aktp#Generic-Area-Of-Interest,  //aktp: the AKT Portal part of the AKT Reference Ontology
  closed_exclusion:  thing_that_is_not_a_domain___object,
  domain_object:  wn#knowledge_domain;


domain_related_relation___domain-related-thing .(?,?)
  kind:  non-directly_usable_type,
  <  relation_playing_a_special_role,
  >  {(relation_from_a_domain  relation_from_a_thing_that_is_not_a_domain)}
     {(relation_to_a_domain  relation_to_a_thing_that_is_not_a_domain)};

    relation_from_a_domain___thing_in_domain .(domain, ?)  kind:  non-directly_usable_type,
      >  subdomain  domain_object;

       //subdomain___true_subdomain .(domain, domain);  //already stated above

         domain_object .(domain, thing_that_is_not_a_domain)  <  member  object_relation;

    relation_from_a_thing_that_is_not_a_domain .(thing_that_is_not_a_domain, ?)
      name:  "thing_in_domain_of_object",
      kind:  non-directly_usable_type,
      >  object_subdomain  object_in_domain_of_object;

         object_subdomain___subdomain_of_object .(thing_that_is_not_a_domain, domain)
           inverse:  domain_object;

         object_in_domain_of_object .(thing_that_is_not_a_domain, thing_that_is_not_a_domain)
           kind:  non-directly_usable_type;

    relation_to_a_domain___sub-domain_or_object-subdomain .(?, domain)
      kind:  non-directly_usable_type,
      >  subdomain  subdomain_of_object;

     relation_to_a_thing_that_is_not_a_domain .(?, thing_that_is_not_a_domain)
       name:  "domain-related-object",
       kind: non-directly_usable_type,
       >  domain_object  object_in_domain;
```

WordNet is nowadays commonly used as an ontology: its synsets are directly interpreted as conceptual categories, even though types and individuals are not distinguished. The integration of WordNet into the MSO included the transformation of the noun-related part of WordNet into a genuine "lexical ontology" by removing internal inconsistencies, giving short intuitive identifiers to the categories and making the distinction between types and individuals (about 3000 individuals were isolated). This meant distinguishing the hyponym relationship (the main specialization relationship in WordNet) into subtype, instance and subdomain relations. These last relation types were declared as subtypes of pm#hyponym for the change to be a refinement of WordNet and hence minimize the number of changes inconsistent with the source.

WordNet uses hyponym relations as well as part relations between its categories for fields of studies. These part relations were kept and these hyponym relations were transformed into subdomain relations. More precisely, the very general kinds of fields of studies – those with annotations beginning by "**any** of the branches of ..." – have been represented as types but their direct subdomains (which have annotations beginning by "the discipline ...") have been represented as their instances. From these instances, most of the other categories for fields of studies (with annotations beginning by "the branch of ...") are reachable via "subdomain" relations.

The next two examples shows direct relations from wn#field_of_study and wn#computer_science. These relations come from WordNet or the MSO input file titled "Fields of Study". (All MSO input files are accessible from http://www.webkb.org/kb/). The source/user/category `is' represents "Information Sciences". A term such as pm/is#software_engineering_science is a term created by `pm' but that "belongs to" (or "has for source" or "has been stated by") `is' *according to `pm'*. Given the previously stated presentation/parsing indication that `pm' prefixes are not shown, all the occurrences of `pm/is' are abbreviated into `is'.

```
wn#field_of_study___discipline___subject_area___subject___subject_field___field___study
  name:  "bailiwick"  "branch_of_knowledge",  //these relations are implicitly from `wn'
  annotation: 'a branch of knowledge; "in what discipline is his doctorate?";
               "teachers should be well trained in their subject";
               "anthropology is the study of human beings"',
  //<  pm#domain __[pm], //already stated
  >  wn#major.field_of_study  wn#frontier.field_of_study  wn#ology
     wn#scientific_discipline  wn#humanistic_discipline
     aktp#Research-Area __[aktp]  aktp#Business-Area __[aktp],
  instance:  wn#allometry  wn#bibliotics  wn#architecture.field_of_study  #numerology
             #engineering_science  wn#theology.field_of_study  wn#military_science;

wn#computer_science___computational_science
  annotation:  "the branch of engineering science that studies
                computable processes and structures",
  subdomain:  wn#artificial_intelligence
              is#software_engineering_science __[is]  is#database_management_science __[is]
            acm#field_of_study_in_the_ACM_classification __[is],
  subdomain of:  wn#engineering_science,
  part:  wn#information_theory,
  part of:  wn#information_science;
```

Below are two tables respectively showing extracts from the beginning of the representations in the MSO input file titled "Classification of the ACM" [www-ACM-in-FL, 2008] and then from the beginning of the MSO input file titled "The ReSIST ontology" [www-ReSIST-in-FL, 2008]. The first table shows that in addition to representing the ACM classification, the MSO structured it and also indexed it with WordNet categories. To ease the understanding of who created what, the prefix "pm#" is often used even though it could have been left implicit.

Table 2.1.2.2. Top-level representations from the MSO input file "Classification of the ACM"

```
pm#field_of_study_in_the_ACM_classification
  kind: wn#field_of_study,
  subdomain of:  wn#computer_science,
  subdomain:  pm#ACM_domains_from_the_hierarchy_with_the_A_to_K_categories_at_the_top
              pm#ACM_domains_from_an_hierarchy_with_general_terms_at_the_top;

    pm#ACM_domain_from_the_hierarchy_with_the_A_to_K_categories_at_the_top
      definition:  "union of the classic ACM hierarchies with the A to K
                    categories at the top, e.g., the 1998 ACM hierarchy"
      subdomain: acm#A  acm#B  acm#C  acm#D  acm#E  acm#F  acm#G  acm#H  acm#I  acm#J  acm#K;

    pm#ACM_domains_from_an_hierarchy_with_general_terms_at_the_top
      annotation:  "hierarchy based on 41 general terms, including the officially
                    listed 16 general terms",
      subdomain:  pm#ACM_situation_related_domain  pm#ACM_entity_related_domain;

       pm#ACM_situation_related_domain
         subdomain:  pm#ACM_state_related_domain  pm#ACM_process_related_domain;

           pm#ACM_state_related_domain
             subdomain:  acm#Security;

               acm#Security  domain_object:  wn#security __[main],
                 subdomain:  acm#Reliability;

                   acm#Reliability  core_domain_object: wn#dependability,
                     subdomain:  acm#Verification; //developed later in the original file

           pm#ACM_process_related_domain
             subdomain:  pm#Design  pm#Experimentation  pm#Management  pm#Measurement
                         pm#Performance  pm#Standardization
                         pm#Verification  //note: a subdomain of it is pm#Reliability
                         pm#ACM_Recovery  pm#ACM_method  pm#ACM_simulation
                         pm#ACM_evaluation  pm#ACM_development  pm#ACM_optimization
                         pm#ACM_analysis  pm#ACM_synthesis  pm#ACM_communication
                         pm#ACM_education  pm#ACM_application;

               pm#Design  domain_object:  wn#designing  wn#design,
                 subdomain:  pm#ACM_Redundant_design  pm#ACM_model;

                   pm#ACM_Redundant_design
                     subdomain:  acm#B.1.3.2  acm#B.2.3.2  acm#B.3.4.2  acm#B.4.5.4
                                 acm#B.5.3.2  acm#B.6.2.2  acm#B.7.3.2;
                 //These domains are developed later in the original file; examples:
                 //  acm#B.1.3.2__Hardware_Control_Structure_Redundant_design
                 //  acm#B.2.3.2__Hardware_ARITHMETIC_AND_LOGIC_STRUCTURES_Redundant_design

                   pm#ACM_model
                     subdomain:  acm#B.1.2.1  acm#B.3.3.0  acm#B.4.4.0  acm#C.0.2
                                 acm#C.2.0.1  acm#C.4.3  acm#D.2.4.4  acm#D.2.9.6
                                 acm#D.2.13.2  acm#D.4.8.1  acm#E.4.2  acm#F.1.1
                                 acm#F.3.2.4  acm#F.4.1.6  acm#H.1  acm#H.2.1.0
                                 acm#H.3.3.4  acm#H.5.5.1  acm#I.2.7.2  acm#I.2.10.3
                                 acm#I.3.5  acm#I.5.1  acm#I.6;
```

Table 2.1.2.3. Top-level representations from the MSO input file "The ReSIST ontology"

```
resist#ReSIST_ontology  <  ontology,
  part:  resist#Dependability-And-Security  resist#Entity;

     resist#Dependability-And-Security
       kind:  aktp#Research-Area __[pm],
       domain_object:  pm#dependency_or_information_security_attribute __[pm]
                       pm#dependency_or_information_security_related_process __[pm];

          pm#dependency_or_information_security_attribute
            >  resist#Dependability __[pm]   resist#Security __[pm],
            <  pm#security_attribute;

               resist#Dependability___High-Confidence___Survivability
                 definition: "ability to avoid service failures that
                              are more frequent and more severe than is acceptable",
                 >  resist#Dependence  resist#Trust  resist#Attribute-Of-Dependable-Systems,
                 <  wn#dependability __[pm];
```

Systems of logics can be seen as "fields of studies" or as "theories". Given the selected interpretation of "fields of studies", there are little differences. As above illustrated, wn#field_of_study and wn#theory were represented as subtypes of pm#domain. Thus, the pm#subdomain relation can apply to them. The next table shows examples. The source/user/category `km' represents the "Knowledge Management" domain. `pm/km' is used for `km' concepts as interpreted and represented by `pm'. Thus, '__[pm/km]' is an abbreviation for '__[pm, source: km]'. The represented categories (and most of the relations that were associated to them) are informally described in the Wikipedia pages indicated for these categories. The represented relations offer a precise, structured and browsable presentation of some of the informal descriptions found in these Wikipedia pages or in the Wikipedia taxonomy (which is not semantically structured and sometimes contains cycles).

Table 2.1.2.4. Extracts from the MSO input file "Systems of logics"

```
   //from now on, `pm/km' is the first default creator in all given examples
   //  except in relation nodes where `pm' stays the first default creator
   //  (new parsing directives will be specified in Chapter 3)
[_ parsing][(pm#formal_term  pm#part of: pm#concept_node) pm#default_creator: pm/km pm wn];
[_ parsing][pm#term_being_declared  pm#default_creator:  pm/km];
[_ parsing][pm#new_term  pm#default_creator:  pm/km];
[_ pm#kind: pm#parsing_macro][pm#statement pm#default_creator:  pm/km];


semantics_of_logic
  subdomain of:  wn#semantics,
  subdomain:  model-theoretic_semantics  proof-theoretic_semantics,
  url:  http://en.wikipedia.org/wiki/Semantics_of_logic;


wn#logic.philosophy
  annotation:  "the branch of philosophy that analyzes inference"
  subdomain:  inductive_reasoning_system  deductive_reasoning_system
              binary_logic  multi-valued_logic
              informal_logic  formal_logic,
  url: http://en.wikipedia.org/wiki/Logic;

     formal_logic
       subdomain:  philosophical_logic  mathematical_logic
                   {(non-modal_logic  modal_logic)}
                   {(classical_logic  non-classical_logic)}
                   type_theory  term_logic  dialectical_logic,
       url:  http://en.wikipedia.org/wiki/Formal_logic;

          non-classical_logic___intuitionist_logic
            subdomain:  substructural_logic  paraconsistent_logic  natural_deduction
                        intuitionistic_logic;

               substructural_logic
                 annotation: "one system of propositional calculus that is
                              weaker than the conventional one",
                 subdomain: relevance_logic  linear_logic,
                 url: http://en.wikipedia.org/wiki/Intuitionistic_logic;

                      relevance_logic
                        annotation: "a kind of system developed to avoid the paradoxes
                                     of material and strict implication",
                        url: http://plato.stanford.edu/entries/logic-relevance/;
```

Indexing and organizing KM resources (techniques, projects, conferences, ...) ***is the focus of several projects.*** The simplest ones use static Web pages, e.g., Peter Clark's Web page titled "Some Ongoing KBS/Ontology Projects and Groups" [Clark, 2008]. There exists several wikis related to this subject (e.g., semanticweb.org) and Wikipedia also has many pages related to it. On the other hand, there is no directory about or close to this subject in Yahoo's Directory nor in the Open Directory Project (Google's Topics Directory) [www-ODP, 2009]. The ACM thesaurus has about 90 categories related to "Artificial Intelligence" (a good part of which are related to KM, e.g. acm#I.2.6.4__Knowledge_acquisition and acm#I.2.4__Knowledge_Representation_Formalisms_and_Methods) and about 50 categories which are related to "information interfaces and presentation" and hence also to KM. The KA2 project [Benjamins et al., 1998] proposed a predefined ontology for some KM domains, asked KM researchers to register their Web pages and annotate some of the content using categories from this ontology, and automatically imported this indexed content into a Web-accessible knowledge base. However, the main part of this ontology was composed of a hierarchy of only 37 Knowledge Acquisition (KA) domains (the names of which also alluded to tasks; thousands of well-organized KM process categories would have actually been needed to permit a useful indexation of the works of researchers). E.g., this hierarchy included:

```
reuse_in_KA  >  ontologies  problem_solving_methods;
  problem_solving_methods  >  Sysiphus-III_experiment;
```

Despite the large publicity that surrounded this project and the good reception it received, few researchers actually indexed their Web page content. The reason may be that even simple indexations are not something that researchers are actually prepared to do but a more probable reason is that the approach was unscalable and hence quite discouraging for the users. Indeed,

- as the above example shows, the relations in the hierarchy were often arbitrary (and neither specialization relations not part relations),
- the ontology was very small and, even more importantly, the end-users could not directly extend it (they had to ask for extensions by email), and
- the notation to use was poorly expressive and thus, like the ontology, very constraining.

The "Semantic Web (SW) Topics Ontology" of ISWC 2006 [www-SWTO, 2006] has about 250 categories about SW related topics, techniques and projects but it too is unscalable and an inefficient support for Information retrieval (IR), even for document indexation, since it is static, arbitrary (there is no "right place" to find or insert a new concept) and does not follow knowledge representation/sharing best practices.
Ontopedia [www-Ontopedia, 2009] is a project whose purpose is to represent and relate subjects of information and knowledge management using Topic maps. As noted in the Web page of the Ontopedia server, only "a small number of subject identifiers" have so far been entered.

An efficient support for IR in KM would need to have an ***initial well organized core of hundreds of categories, be updatable in a scalable way by people and, as above noted, have an ontology of processes for backbone***. Then, the question becomes: ***how to collaboratively organize processes and their related elements in a scalable and easy-to-retrieve way?***

- Section 2.2 presents collaborative techniques.
- Section 2.3 presents lexical, structural and semantic best practices (and normalization principles) followed in the MSO, e.g., "minimizing the number of relation types and the use of `instance' relations" and "maximizing the number of relations between types (especially exclusion relations and transitive relations), the number of spatial/temporal/... constraints on statements, and the use of singular nouns for concept/relation types".
- Chapter 3 shows how in the MSO the WordNet categories (among which hundreds are related to Information Sciences) are organized under intuitive top-level distinctions (see Table 3.1.1.1 for an overview) for each of which all their usual basic relations are proposed and organized in a scalable way (see Table 3.1.1.2 and Table 3.1.1.3 for an overview). Besides these relations, WordNet categories that are subtypes of pm#thing_that_can_be_seen_as_a_relation can be used. Relations "typically used between concepts" are specified as such to permit the generation of menus that guide knowledge representation or retrieval. Some of the distinctions offer ways to organize categories at any level of depth, e.g., see Subsection 3.1.4 for distinctions among processes based on their inputs/outputs.

- An efficient support for IR in KM should also take into account that people may find interesting to organize processes (or find processes organized) according to many non-exclusive viewpoints (e.g., their inputs/outputs of these process, the types of objects they use, the types of techniques they use, and their roles). These viewpoints should then be inter-related via shared subtypes. To that end, ***a basic relation-based "generation of categories for organization purposes" mentioned in the previous subsection is important to generate such an organization without imposing such a work to the users*** (who would anyway not do it in a systematic way) nor cluttering the ontology with "categories for presentation purposes only" that cannot easily be filtered out or shown in non-obtrusive ways. In WebKB-2, this feature is not yet available but prepared for. For the (semi-)automatic classification of the generated categories, it is important that most of the existing categories are defined using basic (and hence also domain independent) relation types. ***Although important, such relations and such generated categories are rarely shown in the examples of this document to avoid cluttering the examples with representations that are not relevant to the illustrated points. Alternatively, as done in Subsection 2.1.4, better names are chosen for the categories; this is not a problem for a (semi-)automatic classification as long as the underlying basic relations are defined.***

The next subsection organizes types of description objects used by KM processes.

2.1.3. Some Description Content/Mediums/Containers in KM

The type pm#description_content/medium/container and its rationales have been introduced in Paragraph 2.1.1.13. Reminder:

```
pm#description_content/medium/container
  >  {(pm#description  pm#description_container)};
```

The type pm#description has also been introduced. Below are more details. pm#data is defined as any pm#description that is not a pm#formal_or_semi-formal_well-formed_statement. Informal statements (e.g., natural language sentences), informal terms and numbers are data.

From now on in this document, ***a category identifier is in bold characters when specialization relations from the category are presented in a subsequent table***, i.e., when the category is really presented. For emphasis purposes, bold italic characters are used, not plain bold characters.

```
pm#description___information
  annotation:  "description (content/medium) of an entity or a situation",
  >  pm#description_content  pm#description_medium
     {(pm#data  pm#formal_or_semi-formal_well-formed_statement___knowledge)};
```

Document elements, the units of information considered by structured document editors and hypertext systems, are description containers:

```
pm#document_element___document___DE
  <  (pm#description_container
        annotation:  "e.g., file, image, ... but not a disk or a piece of paper"),
  definition:  "part of a document or whole document",
  >  wn#document;
```

Words such as "models" and "statements" refer to concepts about description content and to other concepts about description mediums. To avoid representing both interpretations and hence avoid 'duplications', since the description medium interpretations seem to have more "interesting to represent" relations with other description medium categories than the description content interpretations, only the description medium interpretations have so far been represented in the MSO. This is not a very important modeling choice since there is no exclusion relation between pm#description_content and pm#description_medium but this prepares the work for a future formal distinction of the two aspects using more specialized exclusive categories. The next tables show important description medium types in Information Sciences and Knowledge Management.

Table 2.1.3.1. Some top-level categories of description mediums

```
pm#description_medium
  annotation:  "e.g., a syntax, a language, a script, a structure, a term",
  >  {(pm#non-symbolic_description_medium  pm#symbolic_description_medium)},
     wn#structure
     sumo#content_bearing_object; // >  (sumo#linguistic_expression  >  sumo#language)

     pm#non-symbolic_description_medium  >  pm#connexionist_structure;

     pm#symbolic_description_medium
       >  {(pm#atomic_symbolic_description_medium  pm#divisible_symbolic_description_medium)}
          {(pm#fully_formal_description_medium  pm#non-fully_formal_description_medium}
          pm#abstract_data_type  wn#symbolic_representation  wn#language_unit;

          pm#atomic_symbolic_description_medium
            > pm#term
              (pm#atomic_abstract_data_type  <  pm#abstract_data_type,
                                             >  sumo#number  pm#boolean);

          pm#divisible_symbolic_description_medium
            <  pm#non-spatial_collection,
            >  (pm#structured_abstract_data_type  <  pm#abstract_data_type,
                  >  sumo#list  pm#array  pm#queue  pm#stack  pm#keyed_collection_ADT
                     pm#graph_ADT  wn#lattice  pm#number_container  xmls#block_set
                     owl#data_range)
                {(pm#fully_formal_structure  pm#non-fully_formal_structure)}
                pm#statement;

                pm#graph_ADT
                  >  pm#graph_ADT_storing_knowledge-representations;

          pm#fully_formal_description_medium
            >  {(pm#formal_term  pm#fully_formal_structure)};

                pm#fully_formal_structure
                  definition: "mathematically or logically defined structure",
                  >  pm#well-formed_formal_statement;

          pm#non-fully_formal_description_medium
            >  {(pm#informal_term  pm#informal_or_partially_formal_structure)};

          wn#symbolic_representation
            >  (pm#knowledge_representation
                  >  {pm#formal_term pm#formal_or_semi-formal_well-formed_statement});
```

Table 2.1.3.2. Formal and informal terms

```
pm#term
  definition:  "sign (character string, icon, sound, ...) that has been given at least one
                meaning, and hence that refers to one or several types or individuals",
  > {(pm#informal_term  pm#formal_term)})  wn#symbol;

    pm#informal_term  //already stated:  <  pm#non-fully_formal_description_medium,
      definition:  "term without known creator or having several meanings for its creator",
      >  term_from_informal_hierarchy_of_terms_such_as_the_DMOZ_topic_hierarchy
      less pm#constraint_on_meaning than:  pm#formal_term __[any<->any];

    pm#formal_term  //already stated:  <  pm#fully_formal_description_medium,
      description of:  1 pm#thing __[1..*<-any],
```

Table 2.1.3.3. Top-level kinds of statements

```
pm#statement
  > {(pm#semantically-atomic_statement  pm#non-semantically-atomic_statement)}
    {(pm#statement_having_a_meaning_for_at_least_someone
      pm#statement_having_a_meaning_for_nobody)}
    {( (pm#assertable_statement
         > {( (pm#Definition  pm#manually_set_corresponding_relation_type: pm#definition)
              (pm#belief
                > (pm#Annotation  pm#manually_set_corresponding_relation_type: pm#annotation)
                  pm#observation  pm#preference
                  (pm#interpretation  >  pm#deduction  pm#assumption)
                  (task_description  description of:  task __[*<->*])
                  (domain_description  description of:  domain __[*<->*])
              )
           )} )
       (pm#non-assertable_statement  >  pm#query) )},
  member:  1..* term   0..* statement;
           //statements are particular collections of terms and, possibly, of sub-statements

    pm#semantically-atomic_statement
      > {pm#OR-set_of_statements  pm#XOR-set_of_statements
         pm#universally_quantified_statements  pm#definition
         pm#statement_with_a_restrictive_context}
        pm#semantically-atomic_formal_or_semi-formal_statement;

    pm#non-semantically-atomic_statement
      >  (pm#AND-set_of_statements_without_restrictive_context_on_the_set
           > (pm#PCEF_statement___positive_conjunctive_existential_formula
                wn#expressiveness=> some PCEF_logic
             ) ); //a pm#PCEF_statement is an AND-set of relations (atomic statements)
```

Table 2.1.3.4. Statement having a meaning for at least someone

```
pm#statement_having_a_meaning_for_at_least_someone
  definition: "set of quantified terms having at least one logical or semantic meaning",
  member:  1..* pm#term,
  pm#description_content:  1..* pm#description_content __[*<-any]),
  >  {pm#well-formed_formal_statement  pm#partially_formal_statement_with_some_meaning}
     pm#AND-set_of_statements_having_a_meaning_and_with_or_without_context_on_the_set
     {(pm#semantically-atomic_statement_with_some_meaning
       pm#non-semantically-atomic_statement_with_some_meaning)}
     pm#formal_or_semi-formal_well-formed_statement;

     pm#formal_or_semi-formal_well-formed_statement___knowledge___knowledge_statement
       >  pm#semantically-atomic_formal_or_semi-formal_statement
          pm#well-formed_formal_statement;

          pm#well-formed_formal_statement  //already stated:  < pm#fully_formal_structure,
            <  pm#formal_or_semi-formal_well-formed_statement,
            definition:  "statement using only formal terms and with a formal grammar
                          having a logic-based interpretation",
            member=>  1..* pm#formal_term, //"only formal terms"
            >  rdf#statement  pm#PCEF_statement;

     pm#partially_formal_statement_with_some_meaning
       >  {pm#fully_informal_statement_with_some_meaning
           pm#semi-formal_statement_with_some_meaning},
       less pm#constraint_on_meaning than: pm#well-formed_formal_statement __[any<->any];

          pm#fully_informal_statement_with_some_meaning
            definition: "statement not using formal terms but having a
                         logical or semantic meaning for at least someone",
            member:  0 pm#formal_term __[0<-?],
            less pm#constraint_on_meaning than:
                              pm#semi-formal_statement_with_some_meaning; __[any<->any];

          pm#semi-formal_statement_with_some_meaning
            <  pm#formal_or_semi-formal_well-formed_statement,
            definition: "statement that would be formal if it did not use at least
                         one informal term",
            member:  1..* pm#formal_term   1..* pm#informal_term,
            > pm#non-semantically-atomic_semi-formal_statement;
```

Table 2.1.3.5. Knowledge bases, ontologies and models

```
pm#AND-set_of_statements_having_a_meaning_and_with_or_without_context_on_the_set
  >  {pm#AND-set_of_formal_or_semi-formal_well-formed_statements
      pm#AND-set_mostly_composed_of_formal_or_semi-formal_well-formed_statements
      pm#AND-set_of_mostly_informal_statements_having_a_meaning_for_at_least_someone}  model,
  member:  1..* pm#statement_having_a_meaning_for_at_least_someone;

     pm#AND-set_of_formal_or_semi-formal_well-formed_statements
       >  ontology //default creator for concept nodes: `pm/km'
          pm#AND-set_of_statements_in_FS;

          ontology  //this is the interpretation of an "ontology" by `pm'
            annotation: "an ontology is a set of formal terms which can be considered
                         as a pm#non-semantically-atomic_statement' statement asserting that
                         i) the terms are formal terms, and that
                         ii) they have certain associated statements (axioms/definitions)",
            member:  1..* pm#formal_term;

     pm#AND-set_mostly_composed_of_formal_or_semi-formal_well-formed_statements
       >  pm#AND-set_only_composed_of_formal_well-formed_statements  KB_or_KBS,
       member<=  most pm#formal_or_semi-formal_well-formed_statements;

          pm#AND-set_only_composed_of_formal_well-formed_statements
            >  fully_formal_knowledge_base
               (pm#AND-set_of_formal_well-formed_statements_in_FS
                  <  pm#AND-set_of_statements_in_FS,
                  member:  1..* pm#well-formed_formal_statement_in_FS),
            member:  1..* (pm#well-formed_formal_statement
                             > (pm#well-formed_formal_statement_in_FS  pm#language: a pm#FS) );

          KB_or_KBS
            > (knowledge_base  //reminder: according to `pm' and for KM
                >  fully_formal_knowledge_base  knowledge_base_at_least_minimally_organized,
                subset_or_equal<=  1 ontology;

                   knowledge_base_at_least_minimally_organized
                     annotation:  "KB where each term and each statement has been manually
                                  – or can be automatically – connected to a formal term or statement
                                  by a logic-based generalization relation of some kind",
                     member: 1..* (pm#knowledge_representation_with_generalization
                                     <  pm#knowledge_representation,
                                     pm#extended_specialization of<=
                                                                  1..* pm#knowledge_representation))
               (knowledge_base_system
                  subset:  1..* knowledge_base);

     pm#AND-set_of_mostly_informal_statements_having_a_meaning_for_at_least_someone
       member<=  most pm#fully_informal_statement_with_some_meaning;

     model
       >  {conceptual_model_for_a_KB_or_KBS  design_model_for_a_KB_or_KBS}
          {generic_model  instantiated_model}
          {task_description  domain_description},
       subset_or_equal:  task_description __[*<->*]   domain_description __[*<->*];
```

The next table represents some supports of description in the Knowledge Acquisition methodology KADS (referred to via the source/user `kads'). They are referred to in some tables of the next subsection.

Table 2.1.3.6. Some description mediums used by KADS

```
description_medium_in_KADS
  <  pm#description_medium,
  >  description_with_KADS_inference_structure  kads#role;

     description_with_KADS_inference_structure
       definition:  "dataflow graph of 'inferences' (tasks)
                     the inputs/outputs of the inferences are described by 'roles'",
       <  structured_abstract_data_type,
       description of:  1..* kads#inference,
       part:  1..* kads#role;

     kads#role
       >  kads#hypothesis  kads#observable  kads#finding  kads#complaint  kads#norm
          kads#difference  kads#discrepancy_class  kads#diagnosis_result
          kads#parameter  kads#system_model  kads#historical_data;
```

2.1.4. Top-level Processes of Knowledge Management and Acquisition

Table 2.1.4.1. Knowledge management

```
knowledge_management
  acronym:  "KM",
  definition:  "set of processes aimed to design, maintain or exploit a `KB' or `KBS'",
  part of:  wn#information_science,
  subdomain of:  wn#engineering_science,
  subdomain:  knowledge_engineering,
  domain_object:  creating_or_exploiting_knowledge __[.<->any];

     knowledge_engineering
       definition:  "part of `knowledge_management' aimed to
                     design or maintain a `KB' or `KBS'",
       subdomain:  knowledge_acquisition;

     creating_or_exploiting_knowledge
       <  is#information_sciences_process,
       >  {creating_or_selecting_knowledge_from_data_or_knowledge
           associating_data_or_new_knowledge_to_knowledge}
          language/structure_specific_knowledge-based_process
          creating_or_exploiting_knowledge_for_a_non-knowledge-engineering-goal,
       object:  KM_structure __[.<->any];

          creating_or_selecting_knowledge_from_data_or_knowledge
            <  process_creating_copies/versions/selections_of_its_main_inputs,
            >  knowledge_search/retrieval  knowledge_extraction/modeling/representation
               knowledge_importing/exporting/translation
               knowledge_distribution/sharing/merging
               knowledge_inference/reasoning/generation  knowledge_classification
               (storing_knowledge_assertions_or_queries_in_a_document
                  subprocess:  knowledge_extraction/modeling/representation);

          associating_data_or_knowledge_to_knowledge
            <  process_associating_values_to_its_main_inputs,
            >  knowledge_comparison   knowledge_annotation  knowledge_based_indexation
               knowledge_evaluation;

               knowledge_evaluation
                 > knowledge_validation;

               knowledge_based_indexation/annotation .(pm#input: *x, pm#output: *y)
                 <  wn#indexing,
                 input:  pm#description_content/medium/container *x,
                 output:  pm#knowledge_representation *y,
                 subprocess:  knowledge_extraction/modeling/representation .(*x, *y);

          language/structure_specific_knowledge-based_process
            <  process_with_a_main_input/object/output_of_a_particular_kind,
            >  {Formal_Concept_Analysis_specific_process  Conceptual_Graph_specific_process};

          creating_or_exploiting_knowledge_for_a_non-knowledge-engineering-goal
            <  process_that_is_input_or_output_of_another_process,
            >  teaching_knowledge_management;
```

Table 2.1.4.1 represents and relates four domains: Knowledge Management, Information Sciences, Knowledge Engineering and Knowledge Acquisition. The supertype of all processes in these domains is pm#creating_or_exploiting_knowledge, a subtype of is#information_sciences_process. Each of these domains may also be seen as a collection of processes: a representation as a domain is compatible with that interpretation, e.g., pm#domain_object is subtype of pm#member. The direct subtypes of pm#creating_or_exploiting_knowledge specialize the "distinctions among processes based on their inputs/outputs" presented in Subsection 3.1.4 and thus propose three (intuitive and easy to distinguish) kinds of decomposition, two of which being "according to goals" and "according to languages or data structures". The following sections of this chapter introduce specializations for these subtypes. As in the previous subsections, bold characters are used for the identifiers of categories that are specialized in a subsequent table.

The next tables represent Knowledge Acquisition (KA) and top-level processes in it. In the description of common KA methodologies (e.g., KADS [Breuker & van de Velde, 1994]) such processes are not explicitly represented and organized. Furthermore, the subprocess relations are here distinguished from specialization relations. If a process type "X" specializes a process type "Y" then "X is one particular way to do Y"; WordNet uses such patterns for the classification of types of processes and, in WordNet terminology, the "troponym" relation is the "hyponym" (specialization) relation between types of processes. If a process "X" is a subprocess "Y" then "X is a part of Y" and, generally, "the duration of X is a strict part of the duration of Y". Making a choice between a specialization relation or a subprocess relation is not always easy. For example, should kads#analysis_in_knowledge_acquisition_with_KADS be the source node of subprocess relations or subtype relations to kads#environment_analysis_with_KADS and kads#problem_analysis_with_KADS ? In the MSO, subprocess relations have been used. This permits to see the source node of these relations as the sum of all analysis tasks in KADS rather than a supertype for each of them.

KADS refers to tasks rather than processes and distinguishes tasks from goals, problem definitions and problem solving methods (PSMs). A long analysis of their differences and relationships can be found in [Breuker, 1994]. A one-line summary of this analysis could be: "a task permits to solve a well defined problem by applying PSMs". However, this long analysis does not fully permit the reader to ascertain what tasks and PSMs really are with respect to classic distinctions: are they

- processes (they often seem to have all their characteristics),
- descriptions (like problem definitions), or
- arbitrary structures (sentences such as "a task consists of a goal, a problem definition and a PSM" are hints to such a conclusion but they are probably just over-concise sentences)?

Because representing them as descriptions would lead to either complicated representations or a duplication of many of the basic relations associated to processes, the MSO represents pm/km#task and pm/km#problem_solving_method as subtypes of pm#problem_solving_process (on the other hand, the "inference structures" of PSMs clearly are descriptions). This has the additional advantage of not requiring many duplications between types for tasks and types for PSMs.

Table 2.1.4.2. Knowledge acquisition

```
knowledge_acquisition
  acronym:  "KA",
  definition: "part of knowledge engineering aimed to design a `KB' or `KBS'",
  domain_object:  designing_a_KB_or_KBS;

     designing_a_KB_or_KBS
       >  {KBS_design_via_quick_prototyping  model_directed_design_of_KB_or_KBS}
          analysis_in_knowledge_acquisition
          designing_a_KB_or_KBS_with_a_particular_methodology,
       subprocess:  collecting_data_for_building_a_KB_or_KBS;

          collecting_data_for_building_a_KB_or_KBS
            input:  knowledge_source  environment_of_system_to_build,
            output:  data_for_building_a_KB_or_KBS;

          KBS_design_via_quick_prototyping
            input:  data_for_building_a_KB_or_KBS,
            output:  executable_KB_or_KBS;
```

Table 2.1.4.3. Model directed design of a KB or KBS

```
     model_directed_design_of_KB_or_KBS
       subprocess:  building_a_conceptual_model_for_a_KB_or_KBS
                    building_a_design_model_for_a_KB_or_KBS
                    operationalizing_a_KB_or_KBS_from_its_design_model,
       input:  data_for_building_a_KB_or_KBS,
       output:  KB_or_KBS;

          building_a_conceptual_model_for_a_KB_or_KBS  <  knowledge_modeling,
            subprocess:  compose_and_instantiate_generic_task_models
                         compose_and_instantiate_task_independent_models
                         map_instantiated_task_model_and_task_independent_model,
            input_material:  data_for_building_a_KB_or_KBS
            input_parameter:  knowledge_modeling_principle,
            input-output:  generic_conceptual_model_for_a_KB_or_KBS,
            output:  instantiated_conceptual_model_for_a_KB_or_KBS;

          building_a_design_model_for_a_KB_or_KBS
            input:  data_for_building_a_KB_or_KBS,
            input_parameter:  KB_design_principle,
            output:  design_model_for_a_KB_or_KBS;

          operationalizing_a_KB_or_KBS_from_its_design_model
            input:  design_model_for_a_KB_or_KBS,
            output:  KB_or_KBS;
```

Table 2.1.4.4. Knowledge acquisition with the methodology KADS

```
kads#knowledge_acquisition_with_KADS
  <  designing_a_KB_or_KBS_with_a_particular_methodology,
  subprocess:  kads#real_life_process  kads#generic_process;

     kads#real_life_process_in_knowledge_acquisition
       >  kads#analysis_in_knowledge_acquisition_with_KADS  knowledge_engineering,
       description:  (kads#real_life_process_description  part:  kads#generic_model);

          kads#analysis_in_knowledge_acquisition_with_KADS
            <  analysis_in_knowledge_acquisition,
            subprocess: {kads#environment_analysis_with_KADS  kads#problem_analysis_with_KADS
                         kads#task_analysis_with_KADS  kads#function_analysis_with_KADS
                         kads#implementation_analysis_with_KADS};

     kads#generic_process
       >  {kads#problem_solving_method  kads#problem_solving_task  kads#inference},
       description:  (kads#generic_model  <  generic_model);

          kads#problem_solving_method___PSM
            <  (problem_solving_method___PSM  < pm#method  pm#problem_solving_process),
            input:  kads#problem_solving_task,
            output: inference_structure;
```

Table 2.1.4.5. KADS problem solving tasks

```
          kads#problem_solving_task___PST___generic_PST  <  task  problem_solving_process,
            >  {(kads#non-primitive_PST  kads#primitive_PST)}
               kads#system_synthesis  kads#system_modification  kads#system_analysis,
            subprocess:  kads#inference,
            input:  kads#role,  //subtypes given in the previous subsection
            output:  kads#role;

               kads#non-primitive_PST
                 subprocess:  0..* kads#problem_solving_task;

               kads#primitive_PST
                 >  kads#assignment  kads#planning_and_reconstruction  kads#modeling
                    kads#design  kads#modeling  kads#prediction  kads#assessment
                    kads#correction  kads#monitoring  kads#diagnostic  kads#repair,
                 subprocess: kads#inference;

/*                  //Relations of succession between KADS primitive PSTs:
                    kads#assignment
                      succ of: (planning_and_reconstruction succ of: modeling)
                               (design  succ of: modeling)
                      succ: (prediction succ: (assessment succ: correction)
                                               (prediction succ: (monitoring succ:
                                                                  (diagnostic succ: repair)))
                            );
*/
               kads#system_synthesis  < creating_knowledge,
                 comment: "operation without known system model"
                          "multi-dimensional and generally involves 'space'",
                 >  kads#modeling  kads#design  kads#planning/reconstruction;

               kads#system_modification  <  modifying_knowledge,
                 >  kads#assignment;

                    kads#assignment
                      >  kads#scheduling  kads#configuration;

               kads#system_analysis
                 <  exploiting_knowledge_without_modifying_the_main_inputs,
                 comment: "operation on a known system model",
                 >  kads#prediction  kads#monitoring  kads#diagnosis  kads#assessment;
```

Table 2.1.4.6. The most primitive generic tasks of KADS

```
kads#inference___primitive_non-problem-solving_task
  >  kads#collect  kads#generalize  kads#specialize  kads#compare  kads#modify
     kads#determine_truth_or_relevance  kads#explore_option  kads#reduce_working_set;

     kads#generalize___finding_a_kind_or_supertype
       >  (kads#classify___finding_a_kind_or_supertype_in_a_given_type_hierarchy
            >  kads#abstract_class___finding_a_supertype_in_a_given_type_hierarchy
               kads#identify___finding_a_type_from_an_individual)
          kads#cluster___discovering_types_from_examples;

     kads#specialize___finding_subtypes_a_subtype_or_an_individual
       >  kads#refine___finding_a_subtype_for_a_type
          kads#instantiate___finding_an_individual_for_a_type
          kads#select___finding_a_subset_of_things_in_a_set_of_things;
     kads#compare
        >  kads#compare_values  kads#match___compare_structures;
     kads#modify
       >  kads#assign___assigning_a_value_to_an_attribute_of_something
          kads#valuate___producing_a_concept_regarding_the_structure_of_something
          (kads#transform >  kads#sort  kads#restructure)
          kads#assemble
          kads#decompose;
     kads#determine_truth_or_relevance
       >  kads#establish  kads#cover  kads#verify;
     kads#explore_option
       >  kads#make_decision___select_decision   kads#propose_solution___generate_solution;
     kads#reduce_working_set
       >  kads#anticipate  kads#prefer   kads#rule_out;
```

2.2. Knowledge Sharing (KS): Modularization, Indexation, Distribution, Collaboration, ...

Information sharing approaches can be partitioned according to the kind of information they permit to share (e.g., data versus knowledge) and the kinds of processes they rely on (e.g., "(semi-)independent creation of resources to be indexed and possibly merged" versus "collaborative edition of a same resource"). Table 2.2.1.1 show how such distinctions can be categorized based on the description concept types introduced in Subsection 2.1.3. Since this categorization is rather superficial, it is is mainly i) an illustration or initial guideline for the categorization of knowledge sharing approaches and processes they rely on (e.g., indexation processes), and ii) a way to categorize the advocated approaches with respect to other approaches.

Table 2.2.1.1. Main top-level kinds of information sharing

```
is#information_sharing   <  wn#sharing,
  >  {(is#asynchronous_information_sharing  is#synchronous_information_sharing)}
     is#information_sharing_based_on_the_creation_and_indexation_of_multi-statement_resources
     {is#information_sharing_based_on_the_semi-independent_creation_of_resources
      is#information_sharing_based_on_the_collaborative_edition_of_a_same_resource}
     {(knowledge_sharing_based_on_at_least_one_formal_or_semi-formal_knowledge_base
       is#information_sharing_based_on_at_least_one_non-formal-or-semi-formal_resource)}
     knowledge_sharing,
  object:  1..* pm#description,
  subprocess:  is#information_retrieval;

     is#information_sharing_based_on_the_creation_and_indexation_of_multi-statement_resources
       >  {is#information_sharing_based_on_the_indexation_of_mostly_informal_resources
           knowledge_sharing_based_on_the_creation_and_indexation_of_mostly_formal_resources}
          is#info_sharing_based_on_the_indexation_of_mostly_independently_created_resources,
       subprocess:  1..* (is#indexing_of_multi-statement_resources  <  wn#indexing,
                          object: 1..* pm#non-semantically-atomic_statement);

          is#information_sharing_based_on_the_indexation_of_mostly_informal_resources
            subprocess:  1..* (is#indexing_of_mostly_informal_resources  <  wn#indexing,
                               object: 1..*
                 pm#AND-set_of_mostly_informal_statements_having_a_meaning_for_at_least_someone);

     knowledge_sharing
       <  knowledge_distribution/sharing/merging,
       >  (knowledge_sharing_based_on_at_least_one_formal_or_semi-formal_knowledge_base
            >  knowledge_sharing_based_on_the_indexation_of_mostly_formal_resources
               knowledge_sharing_based_on_a_knowledge_base_for_problem_solving_purposes
               sharing_of_a_knowledge_base_that_is_physically_distributed_or_not),
       object:  1..* knowledge_base,
       subprocess:  supporting_knowledge_sharing_between_KBs
                    supporting_the_valuation_of_knowledge_or_knowledge_authors;

          sharing_of_a_knowledge_base_that_is_physically_distributed_or_not
            >  (sharing_of_a_knowledge_base_at_least_minimally_organized
                  object:  1 knowledge_base_at_least_minimally_organized) ),
            subprocess:
               (supporting_loss_less_knowledge_sharing_between_KBs
                 <  supporting_knowledge_sharing_between_KBs,
                 subprocess: integrating_all_published_information_specified_as_parameter)
               (supporting_the_collaborative_building_of_the_KB_specified_as_output
                 subprocess:
                    supporting_collaborative_knowledge_editions_in_a_shared_KB
                    (supporting_a_precise_valuation_of_knowledge_statements_or_authors
                      <  supporting_the_valuation_of_knowledge_or_knowledge_authors) );
```

Based on these distinctions, the next three subsections argue that almost all current approaches for the sharing of information are not scalable. The argumentation has not yet also been represented or organized in a semi-formal way. However, most of the arguments are directly or indirectly derived from

- relations of type pm#constraint_on_meaning that can be set between current or future specializations of the type pm#description (as illustrated in Table 2.1.3.2 and Table 2.1.3.3), and
- application of rules such as "the more precise the indexation, the more precise the indexing process that produces it" which has been represented in Paragraph 2.1.1.22.

The type pm#constraint_on_meaning is a subtype of wn#precision, and knowledge precision is important for the scalability of knowledge sharing. Here is what the next three subsections argue for:

```
sharing_of_a_knowledge_base_at_least_minimally_organized
  more wn#scalability than:
     is#information_sharing_based_on_the_creation_and_indexation_of_multi-statement_resources
     knowledge_sharing_based_on_a_knowledge_base_for_problem_solving_purposes;
```

After these next three subsections, the three subsequent subsections present an approach for the following kinds of processes:

- supporting_loss-less_knowledge_sharing_between_KBs,
- supporting_collaborative_knowledge_editions_in_a_shared_KB, and
- supporting_a_precise_valuation_of_knowledge_statements_or_authors.

Only asynchronous information sharing is considered here since its techniques underly (and are more scalable than) those supporting the exchange of information between co-temporal users of a system.

Here are some reminders or precisions on two terms used in the next three sections (and at some other places in this document):

- a "resource" is a stand-alone collection of several statements (e.g., an ontology, a database, a document, a section or a paragraph), and
- "metadata" is a set of one or several numerical values or other objects used for relating or indexing one or more statements, typically those of a resource.

Some metadata related to some resource or created by some person(s) can also be considered as a resource.

2.2.1. Unscalability of KS Approaches Based on the Indexation of Resources

The more statements a resource contains, and the more resources there are, the more these resources contain similar and/or complementary pieces of information, and hence the less the metadata for each resource can be useful: queries will return lists of resources that are partially redundant or complementary with each other and that need to be manually searched, compared or aggregated by each user.

The more statements a resource contains, the more its metadata have to be information selective, and hence the less such metadata are representative of the contained pieces of information and the more the indexation methods and usefulness are task/user/domain dependent and somewhat arbitrary.

Finally, the more statements some resources contain, and the less formal the statements are (or the more context-dependent their interpretations are),

- the less any similarity measure between these resources can have any intuitive or semantic meaning, and
- the less it is possible to relate these resources meaningfully by semantic relations (e.g., specialization relations, rhetorical relations or argumentation relations).

For example, the statement "some animal sits above some artifact" is a generalization of both "Tom (a cat) sits on a blue mat" and "any animal sits above some artifact" because all the objects and quantifiers of the first statement are identical or generalize those of the second and third statements (such relations can be automatically inferred if the statements are formal or semi-formal). However, such relations rarely hold between two collections of statements, and especially between any two documents.
Statistical similarity/distance measures between documents, ontologies or metadata, are useful for many purposes [Euzenat et al., 2009] but, like all statistical measures, have no *intuitive "semantic"* meaning: they are experimentally or mathematically designed to be of some help for some *specific kinds of data, tasks or users*. For example, Knowledge Zone [Lewen et al., 2006] allows each of its users to i) rate ontologies with numerical or free text values for criteria such as "usage", "coverage", "correctness" and "mappings to other ontologies", ii) rate other users' ratings, and iii) uses all these ratings to retrieve and rank ontologies. This approach compounds several problems:

- whole ontologies are rarely genuinely/intuitively comparable (indeed, given two randomly selected ontologies, it is very rare that one fully includes or specializes the other),
- giving numerical values for such criteria is rather meaningless and hence arbitrary,
- textual values for each of such criteria cannot be automatically organized into a semantic network,
- two sets of criteria are rarely comparable (one set rarely includes all the criteria of the other set and, at the same time, has higher values for all these criteria), and
- similarity measures on criteria only permit to retrieve *possibly "related"* ontologies: the work of understanding, comparing or merging their statements still has to be (re-)done by *each* user.

To sum up, however sophisticated, techniques that index resources are inherently limited in their possibilities and usefulness for information seekers. Furthermore, since they do not provide re-use mechanisms, they force information providers to repeat or re-describe information that is described elsewhere and thus add to the volume of redundant data that information seekers have to sift through. Yet, techniques to index data or people form the bulk of Learning Object retrieval/management techniques and Semantic Web related techniques, for example in the Semantic Learning Web [Stutt & Motta, 2004] and the Educational Semantic Web [Devedzic, 2004]. Although the number and seeming variety of these techniques is important, our definitions permit to categorize most of them in the following few categories (with, in each of the two groups, the listed kinds of techniques or tools being more or less ordered by increasing indexation precision).

- As annotation tools permitting their users to index or relate resources or metadata i) by informal terms (e.g., folksonomy tools and topic map based tools), ii) by terms from a small predefined small list such as the Dublin Core metadata or argumentation relations as in ScholOnto [Buckingham-Shum et al., 1999], iii) by terms from an informal hierarchy of terms such as the DMOZ topic hierarchy, iv) by terms from a lexical database such as WordNet, v) by terms from a semantically organized ontology such as the SUMO, vi) by terms from an ontology that can be dynamically updated by users, as in WebKB-2 [Martin, 2003a], vii) by attribute-value pairs

with textual or numerical values, viii) by restricted kinds of knowledge representations (e.g., semantic wikis), or ix) by expressive knowledge representations, as in WebKB-2 which uses Conceptual Graphs and Formalized-English.

- As tools automatically indexing or relating resources or metadata i) by terms from a predefined small list, ii) by informal terms automatically organized into a hierarchy via techniques such as Latent Semantic Indexing, Formal Concept analysis or terminological analysis, iii) by terms from lexical databases via natural language parsing (NLP) techniques, iv) by attribute-value pairs with textual or numerical values, v) by a measure of similarity between resources and/or their metadata, vi) by informal sentences (e.g., summarizing tools) using statistical/NLP techniques, or vii) by restricted kinds of knowledge representations (e.g., question-answering tools which index sentences in documents but are not able to represent most of the semantic content of different sentences and hence organize it) via NLP techniques or ad-hoc Web site wrappers. [Shadbolt et al., 2006] acknowledges that current "Semantic Web"-like applications still use ad-hoc wrappers from particular Web documents or databases.

Illustration with learning objects (LOs) [www-LO 2009]. Many researchers in (e-)learning focus on techniques to create, index, retrieve or manage LOs (learning materials aimed to be "re-used and combined" for creating a course, or browsed for learning purposes). It is now well accepted by such researchers that the smaller and less contextual the "LOs available for re-use" are, and the more precisely indexed or inter-connected via metadata they are, the more easily they can be semi-automatically retrieved and combined to create i) "LOs to teach with" that are adapted to particular course objectives or kinds of users, and hence ii) "context-dependent LOs" [Downes, 2001] [Hodgins, 2006]. However, in current LO repositories [www-LOR, 2009], a LO is almost never a "semantically meaningful LO", i.e., either a formal term or an "un-decomposable semantic statement" (an instance of pm#semantically-atomic_statement_with_some_meaning; typically, one semantic relation between semantically meaningful LOs, with some formal information about the context of this relation, such as its creator and temporal, spatial or modal constraints on its validity). For example, a typical LO about Java is an "Introduction to Java" listing some features of Java and giving an example of code, instead of being a relation between Java and one of its features. Since LOs are not atomic knowledge representations and since LO repositories are not cgosSWs, retrieving and combining them is difficult.
Current LO related standards [www-LOS, 2009] (e.g., AICC, SCORM, ISM, IEEE WG12) and projects (e.g., CANDLE, GEODE, MERLOT, VLORN) essentially focus on associating simple meta-data to whole documents or big parts of them (e.g., author, owner, terms of distribution, presentation format, and pedagogical attributes such as teaching or interaction style, grade level, mastery level and prerequisites). Each of such LOs cannot actually be a "truly re-usable LO" but is a package of objects selected and ordered to satisfy a certain curriculum. Although such packages are useful for pedagogical purposes and ease the task of most course designers since they are ready-made packages, they are *black-box packages*, that is, their decomposition into objects of a cgosSW has not been made explicit and hence they cannot be easily modified nor compared or efficiently retrieved: i) they can only be retrieved via keywords, not via arbitrary complex conceptual queries on the objects they contain, and ii) from a browsing viewpoint or a conceptual querying efficiency viewpoint, they cannot be organized into a lattice (partial order) according to the objects they combine.
More details on the interest of a cgosSW for LOs and (e-)learning can be found in [Martin & Eboueya, 2008]. Details on the use of WebKB-2 related techniques for (e-)learning can be found in [Martin, 2009]. [Yessad et al., 2009] introduces an ontology-based semantic distance for detecting the relevance of LOs.

2.2.2. Unscalability of KS Approaches Based on Either Fully Formal or Mostly Informal Resources

Some information repository projects use formal KBs, e.g., the OpenGALEN project which created a KB of medical knowledge, the QED Project [www-QED, 1995] which aimed to build a formal KB of all important, established mathematical knowledge, and the Halo project [Friedland et al., 2004] which has for very long term goal a system capable of teaching much of the world's scientific knowledge by preparing and solving test questions for students according to their knowledge and preferences. Such formal KBs permit to support problem solving but they are not meant to be directly read or browsed, and designing them is difficult even for teams of trained knowledge engineers, e.g., the six-month pilot phase of Project Halo was restricted to 70 pages of a chemistry book and had encouraging but far-from-ideal results. Hence, such fully formal KBs are not adequate for scalable information sharing or retrieval.

Informal documents (articles, emails, wikis, etc.), that is, documents mainly written using natural languages such as English, as opposed to knowledge representation languages (KRLs), do not permit objects to be explicitly referred and interconnected by semantic relations. This forces document authors to summarize what has been described elsewhere and make choices about which objects to describe and how: level of detail, presentation order, etc. This makes document writing a time consuming task. Furthermore, the lack of detail often makes difficult for people or softwares to understand the precise semantic relations between objects implicitly referred to within and across documents. This leads to interpretation or understanding problems, and limits the depth and speed of learning since retrieving or comparing precise information has to be done mostly manually. The automatic indexation of sentences within documents permits to retrieve sentences that may contain all or parts of some required information (this process is often called "question answering"; tools supporting it, e.g., WeboPedia and Ask Jeeves, are evaluated by the TREC-9 workbenches) but the lack of formalization in the sentences often does not permit to extract and merge their underlying objects and relations.

Cognitive maps [www-COM, 2009] and concept maps [www-CM, 2009] [Novak, 2004] – or their ISO version, topic maps [www-XTM, 2009] – have often been used for teaching purposes (e.g., many examples in biology can be found in [Leung, 2005]). However, they are overly permissive and hence do not guide the user into creating a principled, scalable and automatically exploitable semantic network. For example, they can use relations such as "of" and nodes such as "other substances" instead of semantic relations such as "agent" and "subtask", and concept names such as "non-essential_food_nutrient". Thus, concept maps are often more difficult to understand or retrieve, aggregate and exploit than regular informal sentences (from which, unlike deeper representations, they can currently be automatically generated); [Sowa, 2006] gives commented examples of many shortcomings of concept maps.

Similarly, the modeling of the preferences and knowledge of students or other people is often very poor, e.g., a mere keyword for each learned subject or LO (e.g., "Java") and a learning level for it (e.g., "advanced"). This is for example the case with the CoAKTinG project [Page et al., 2005] which aims to facilitate collaboration and data exchange during or after virtual meetings on a semantic grid, and the Grid-E-Card project [Gouardères et al. 2005] which manages a model of certification for each LO and each student on a grid to facilitate the learning and insertion of this student within relevant communities. For efficacy and scalability purposes, it is necessary to use a more fine-grained approach in which all the statements for which a student has been successfully tested on are recorded.

Some reasons why more knowledge-oriented solutions are not developed can be listed as follow:

1. most people, including many tool developers, have little or no knowledge about semantically explicit structures,
2. many tool developers fear that people will be "scared away" by the looks of such structures or by having to learn some notations,
3. precise and correct knowledge modeling is complex and time-consuming,
4. KB systems are not easy to develop, especially user-friendly ones supporting collaboration between their users,
5. there currently exists a lot of informal legacy data but very little well-organized explicit knowledge.

Point 2 was the reason given by many creators of "knowledge-oriented" hypermedia systems or repositories to explain the limited expressiveness of the formal features or notations proposed to the users. This was for example the case for the creators of SYNVIEW [Lowe, 1985], AAA [Schuler & Smith, 1992], ScholOnto [Buckingham-Shum et al., 1999] and the Text Outline project [Sanger, 2006]. [Shipman & Marshall, 1999] notes that the restrictions of

knowledge-based hypermedia tools often lead people not to use them or to use them in biased ways. Although this fact appears to be presented as an argument against knowledge-based tools, it is actually an argument against expressiveness restrictions set by tool developers to ease their tasks (especially for designing graphical interfaces) and, supposedly, to avoid confusing the users. As in [Shipman & Marshall, 1999], it can be concluded that annotation tools should provide users with generic and expressive structuring features but also convenient default options, and that the users should be allowed to describe their knowledge at various levels of details, from totally informal to totally formal so that they can invest time in knowledge representation incrementally, collaboratively and only when they feel that the benefits out-weight the costs.

Although the points 1 to 5 of the previous paragraph are valid, it has been previously argued that effective or scalable knowledge sharing and retrieval cannot be achieved without a "global virtual KB" (a cgosSW) which, to a large extent, is collaboratively updated by the information providers themselves. Although using this cgosSW requires the learning and use of graphical or textual notations for representing information precisely, *in the long term* this probably won't be a problem. Here are some rationales.

- The need for programming languages and workflow or database modeling notations is already well accepted and more and more students learn them. Many persons also learn languages which, like Perl or many XML-related languages (e.g., RDF+OWL/XML, XML Schema and XPATH), are less regular than the notations presented in this document and hence at least as difficult to learn.
- There are many projects aiming to support collaboratively-built large repositories in any domain (e.g., Wikipedia, Truly Open Directory, OntoWiki and Freebase which is also a regular ontology server) or in a given domain (e.g., one of the goals of the Mizar project [www-Mizar, 2009] is to formalize the entire body of informal proofs of mathematics). These projects have many enthusiast contributors despite the fact that the "need for recognition" of the contributors is not particularly exploited since the sources/authors of the contributions are often not displayed in the text.
- Many persons spend time doing intellectual games or other complex things as hobbies (e.g., hackers and Linux contributors). If the advantages of using formal notations were better known and if using them became a popular game or challenge, there would not be a lack of contributors to global virtual KBs. This is not a new idea: it is used in [Witbrock et al., 2003] and [Siorpaes & Hepp, 2007] notes that there are several "games [such as OntoGame] masquerading core tasks of weaving the Semantic Web behind on-line, multi-player game scenarios". This description directly applies to semi-formal discussions (see the examples in Table 2.1.1.22.1 and Section 4.1).

For the following reasons, a global virtual KB is likely to be first adopted by (e-)learning researchers:

1. the need of using very small learning objects is now well recognized by the e-learning research community,
2. the economy of time and resources brought by the use of truly re-usable learning objects will be understood by more and more e-learning/university teachers and administrators,
3. more and more teachers are involved in e-learning,
4. it is part of the roles of teachers and researchers to (re-)present knowledge in explicit and detailed ways,
5. a cgosSW permits a better evaluation of the knowledge and analytic skill of the students than less precision-oriented approaches, and
6. providing the semantic organization of the content of teaching materials (instead or in addition to these materials) help students find, compare and memorize the information scattered in these materials.

[Martin, 2009] describes an experiment to validate the sixth point. The most important concepts and statements of learning materials of three courses at Griffith University were represented into three semantic networks in FL (this did not require making quantifiers explicit). For each course, its related network was presented to the students and, as a replacement for an informal learning journal, the students were invited to extend the network (first separately and then collaboratively) by adding not-yet-represented concepts and statements from the informal supports for the courses. The input files for the semantic networks are (like all Information Sciences related input files of WebKB-2) directly or indirectly accessible from http://www.webkb.org/kb/it/. The sixth point was acknowledged by many of the students after they had learned how to read the semantic networks.

2.2.3. Unscalability of KS Approaches Based on Mostly Independently Created Resources

Like the previously presented distributed knowledge sharing strategies, the W3C's strategy is minimal: the W3C only proposes a low-level KRL (RDF+OWL and the Rule Interchange Format) and some optional rudimentary "best practices" [Swick et al., 2006], and envisages the Semantic Web to be composed of many small KBs (RDF documents) which are more or less independently developed and hence partially redundant, competing and very loosely interconnected since the knowledge provider is expected to select, import, merge and extend other people's KBs into her own [Hendler, 2001] [Rousset, 2004]. This approach which relies on "formal documents" has problems that are analogue to those above listed for "informal documents":

- finding relevant KBs, choosing between them and combining them is difficult and sub-optimal even for a knowledge engineer, let alone for softwares,
- a knowledge provider cannot simply add one object "at the right place" and is not helped nor guided by a large KB (and a system exploiting it) into providing precise and re-usable objects that complement the already stored objects, and
- as opposed to normalized insertions into a shared KB which directly or indirectly guide all other related insertions, creating new ontologies actually increases the amount of poorly interconnected information to search, compare and merge by people or software agents.

Most of current Semantic Web related approaches focus on supporting the manual setting or automatic discovery of relations between formal terms from different ontologies. They are quite understandably imperfect but can be sufficient for certain applications and are necessary to re-use existing ontologies. Comparisons of ontology matching tools are yearly done by the "Ontology Alignment Evaluation Initiative", e.g., see [Euzenat et al., 2005] and [Caraciolo et al., 2008]. Although more interested in "ontology matching" [Euzenat & Shvaiko, 2007], Euzenat has acknowledged the interest of (semi-)formal KB servers that let both people and software agents directly exploit and save new knowledge or object alignments, that is, query, complement, annotate and evaluate the existing objects, guided by large, shared and well-organized KBs.

The next subsections and sections present some avenues to support an approach that would be efficient and scalable for knowledge sharing and retrieval on the Internet or within large intranets: the collaborative creation of a cgosSW, hence a global virtual well-organized (semi-)formal KB without redundancies nor implicit inconsistencies.

2.2.4. Supporting Knowledge Sharing Between KBs (or: Combining the Advantages of Centralization and Distribution)

A cbwoKB lifts some governance and scalability issues but not the dependence of the users with respect to the owner(s) of these servers. Furthermore, one cbwoKB server cannot support knowledge sharing for all communities. For scalability purposes, the cbwoKB servers of different communities or persons should be able to interact for creating a ***global virtual cbwoKB (gv-cbwoKB) without a central brokering system, without restrictions on the content of each KB, and without the individual cbwoKB servers necessarily having to register to a particular super-community or peer-to-peer (P2P) network***. For several cbwoKB servers to be collectively seen as such a gv-cbwoKB, it should not matter which KB a user or agent chooses to query or update first. Hence, object additions/updates made in one KB should be replicated into all the other KBs that have a scope which covers these objects. Similarly, when relevant, a server should forward (parts of) a query to other servers in order to give a more complete answer.

With respect to these specifications, *current approaches for distributed querying amongst a few KB servers (e.g., [Lee et al., 2010]) or for collaboration between KB servers/owners* seem insufficient. Examples for such collaboration approaches are those of [Casanovas et al., 2007] [Noy & Tudorache, 2008] which are based on integrating changes made in other KBs, and those of [Palma et al., 2008] which also use a workflow system. Indeed, even in Semantic Grids and semantic-based P2P networks, both kinds of approaches are based on *partial descriptions* of the content of each KB or on *predefined roles* for each user, and the r*edundancies or inconsistencies between the KBs are not made explicit*. This often makes difficult for people or systems i) to find relevant KBs to search or update, and ii) to integrate query results. Furthermore, no semantic-based Peer-to-Peer architecture (e.g., see [p2p-Pitoura 2006] [p2p-semanticQuerying 2006]) is yet flexible enough to permit the semantic querying – or (cross-)indexation of knowledge among peers – required by a gv-cgosSW.

As in the previous sections, a solution is to let the knowledge indexation and distribution be made at the object level instead of the file/KB/community/owner level. More precisely, to satisfy the above cited specifications, ***for a cbwoKB server*** to be part of one or many gv-cbwoKBs, the ***requirement*** is that ***for every term T it stores in its KB, this server*** must ***either***

- ***have a Web-accessible formal description specifying that it is committed to be a "nexus" for T***, i.e., that i) it stores any statement S on T (if S is inserted in another KB of this gv-cbwoKB, it is also inserted in this KB), or ii) it associates to T the URLs of cbwoKB servers permitting to find or store any statement on T, ***or***
- not be a "nexus" for T, and hence ***associate to T*** either i) the URLs of all cbwoKB servers that have advertised themselves to be a nexus for T, or ii) the URL of at least one server that stores these ***URLs of nexus servers for T***.

Thus, via forwards between servers, all objects using T can be added or found in each nexus for T. This *requirement* is an adaptation and refinement of the 4th rule of the Linked Data approach [Bizer et al., 2010]: "link things to their related ones in some other data sets". Indeed, to obtain a gv-cbwoKB, the data sets must be managed via cbwoKB servers and there must be at least one nexus for each term. A consequence is that when the scopes of two nexus overlap, they share the same common knowledge and there is no inconsistencies nor *implicit* redundancies between them. Thus, the gv-cbwoKB has a unique ontology distributed on the various cbwoKB servers. This approach would work with cbwoKB servers on the Web but also in a P2P network (or semantic grid) where each user has his own KB server. The main difference is that a P2P network permits to implement systematic push/pull mechanisms instead of relying on KB servers to regularly check the KBs of other servers and integrate new updates.

For the owners of a cbwoKB server wanting it to be part of one or several gv-cbwoKBs, the difficult task is to integrate the ontology of their server into the global one(s). More precisely, for each term T of their ontology they must first find via a Web/ontology search engine if some other server has advertised itself as a nexus for T. Once they have found a nexus for one of the terms, by navigation between nexus, finding a nexus for other terms may be easier. This search is somewhat similar in its approach to the search of a "right place" to insert a new term in a cbwoKB. When all the nexus are found, via the forwarding of updates between them, the protocol described in the previous section could be applied at the scale of the whole gv-cbwoKB constituted by this network of nexus. An integration

task is at the core of most knowledge sharing or re-use approaches. Their methods can be re-used for the loss-less integration of the ontology of a cbwoKB into the one of a gv-cbwoKB. An important point is that this integration of an ontology has to be done only by the creators of this ontology (or owners of the cbwoKB) who know what the objects in this ontology mean. Once this integration is made, regularly and (semi-)automatically integrating new knowledge from other nexus is much easier since a common ontology is shared. Thus, it can be envisaged that one initial cbwoKB server be progressively joined by other ones to form a more and more general gv-cbwoKB.

The cornerstone of the approach is the formal commitment of *being a nexus for a term* (and hence of *being a cbwoKB* since direct searches or updates by people must be allowed). This formal commitment ***should be advertised in a standard way.*** To do so, the media and KRL to use is not an issue: a public RDF+OWL/XML file should at least be used (since this is what is recommended by the W3C and hence is de-facto standard) even though, as illustrated in Chapter 4 (e.g., in Subsection 4.2.6), RDF+OWL and RDF/XML are respectively a low-level model and syntax which, whenever the reference collections are not simple to describe, leads knowledge engineers to come up with ad-hoc representations that are difficult to compare, merge or exploit.
What is an issue is that there is no standard term to describe this scope and commitment. The W3C does not recommend any formal term to assert that a particular server commits to continually i) search for other resources (e.g., database/KB servers or static files) that provide knowledge relevant to its scope, and ii) include this knowledge in its KB. Ideally, the specification should also indicate the list of discovered relevant resources, the kind of knowledge that could or could not be copied from these sources or integrated in the KB, and the polling period. The Dublin Core relation type named "Coverage" is often used for specifying the content of a resource but is not meant to also permit to specify that the resource is in a certain sense "complete" with respect to the described scope. This is why the MSO proposes the following process type.

```
integrating_all_published_information_specified_as_parameter
  .(agent: *x, parameter: ontology *y)
  //subprocess of:  supporting_loss-less_knowledge_sharing_between_KBs, //already stated
  agent:  1..* pm#causal_entity *x,
  parameter: (KB_reference_collection  < pm#ontology //a set of formal terms and definitions
             ) __[any->1..* *y],
  output:  1..* pm#description_content/medium/container,
  input:  1..* pm#description_content/medium/container,
  period:  1 pm#time_measure;
```

Any other type of "relation from/to a process" can be used from a node of this type. Here are two examples of its use:

```
pm#Ray_White_real_estate_in_Southport_Queensland_Australia
  url:  http://www.raywhite.realestate.com.au/southport,
  agent of:
     (any pm#Ray_White_integration_of_all_real_estate_information_in_Southport_Queensland
        <  integrating_all_published_information_specified_as_parameter,
        parameter:  { (wn#real_property  place: QLD#Southport) },
        output:  http://www.raywhite.realestate.com.au/southport,
        input:  every (. pm#real_property_agency_in_southport  < #enterprise,
                         agent of:  1..* (. wn#selling_real_estate_in_Southport_Queensland
                                            <  wn#selling,
                                            location:  QLD#Southport,
                                            object:  1..* wn#real_property
                                         ) ),
        period:  every pm#Thursday
     );
pm#Ray_White_real_estate_in_Southport_Queensland_Australia
  integrating_all_published_information_specified_as_parameter:
                                                  { pm#real_property_agency_in_southport };
```

The actual coverage of a particular domain by a knowledge server X can be checked by automatically exploring other servers related to this domain and see which percentage of all their "knowledge relevant to the specified domain" are also in the knowledge server X. Such information may for example be published by competitors. In a business environment, it is in the interest of a competitive company to check what its competitors or related companies offer

and either integrate (and hence compare) their public information in its public files (Web pages, database, KB) and/or refer to Web-accessible Web pages. It is also in its interest to refer to the most comprehensive KBs of its related companies.

Furthermore, ***if a formal term such as pm/km#integrating_all_published_information_specified_as_parameter was made a recommendation by the W3C, it would be in the interest of a competitive company to create such an integration and use such a term to advertise this integration (and hence be listed prominently by Web Search engines), thereby advertising its competitiveness and thus attracting clients.*** The next two subsections and, more generally, all the chapters of this document, show how these clients could be allowed to contribute information or feedbacks to such an integration, while keeping it well ordered and easy to filter (by end-users or the company) according to certain viewpoints. Thus, it may be in the interest of a *competitive* company to also allow this. Indeed, many companies not only survey what people write about them in their blogs but also offer people a place on their official Web sites to blog about them, even if this often means that the company offers a well-visited platform for people to complain about its practices or products. Supporting the techniques described in the next subsections permit the feedbacks to be semantically ordered and collaboratively evaluated (and hence retrievable, comparable and reducing the impact of spams or dishonest messages) instead of being mostly long unordered lists of messages. Thus, supporting such techniques would likely ease the gathering of information and increase the trust that people have in the collected information. To distinguish servers who allow a collaborative building of their KB from other servers, the MSO proposes the following process type.

```
supporting_the_collaborative_building_of_the_KB_specified_as_output
  .(agent: *x, output: KB *y)                 //the next line has already been stated
  //subprocess of:  sharing_of_a_knowledge_base_that_is_physically_distributed_or_not,
  agent:  1..* pm#causal_entity *x,
  output:  1 KB;
```

Here is an example of its use:

```
official_WebKB
  kind:  WebKB,
  url:  http://www.webkb.org,
  supporting_the_collaborative_building_of_the_KB_specified_as_output:
        (MSO_of_WebKB-2___MSO___Multi-Source_Ontology_of_WebKB-2
           <  ontology,  subset of=>  WebKB __[?mso->any]
        ) __[.->?mso];
```

To conclude, if the W3C recommended a small ontology that included the last two introduced process types along with the common basic types of relations associated to processes or to situation (see Table 3.1.3.3 to Table 3.1.3.5), this would likely go a long way towards popularizing techniques similar to those described in this subsection and hence towards much better knowledge sharing.

The approach described in this section seems the simplest knowledge distribution approach because

- the approaches used in distributed databases would not work since KBs do not have any fixed conceptual schema (they are composed of large, explicit and dynamically modifiable conceptual schemas), and
- a fine-grained classification or ontology for all the objects is necessary since classifying servers according to fields or domains is far too coarse to index or retrieve knowledge from distributed servers; for example, some knowledge about "neurons" or "hands" in some domains may be relevant to many other domains.

This approach would work with servers on the Web but also in a peer-to-peer (P2P) network (or semantic grid) where each user has her own KB server: the main difference is that a P2P network permits to implement systematic push/pull mechanisms instead of relying on KB servers to regularly check the KBs of other servers and integrate new additions.

None of the current P2P architectures (e.g., [p2p-semanticRouting 2004] [p2p-semanticDrivenHashing 2004] [p2p-semantic 2005] [p2p-semanticQuerying 2006]), even RDF-oriented P2P architectures [p2p-Pitoura 2006], are flexible enough to permit the semantic querying – or the (cross-)indexation of information from the knowledge sources – required by a cgosSW. However, two complementary approaches seem possible to ensure the cross-indexation of knowledge among peers. One would be to implement the replication mechanism described in this subsection. A more restricted but complementary approach would be to redesign the Distributed Hash Table (DHT) routing/indexation algorithm to use formal terms (semantic categories) – and their specialization relationships – instead of informal terms (words). One idea is to encode the specialization hierarchy of the virtual ontology into a DHT structure with nodes also representing concepts and node neighborhood relationships representing specialization relations between these concepts. This idea is an extension of the idea used in [p2p-semanticRouting 2004].

Integrating knowledge from other servers of large KBs is not easy but it is easier than integrating dozens or hundreds of (semi-)independently created small KBs. Furthermore, since in the proposed approach the first integration from a server is loss-less, the subsequent integrations from this server are much easier. A more fundamental obstacle to the widespread use of this approach is that many industry-related servers are likely to make it difficult or illegal to mirror their KBs; however, this problem hampers all integration approaches.

The above described replication mechanism complements works on the distributed querying of KBs (e.g., [Gandon et al., 2008]). It is also a way to combine advantages commonly attributed to "distributed approaches" and "centralized approaches". Indeed, distribution and centralization only have opposite meanings when applied to actions and physical things, not when applied to easy-to-copy information. Decentralizing actions (and in particular, decisions), i.e., distributing manpower and decision power to individual persons, is generally a good thing. Centralizing information only implies gathering and relating information; this can be done without centralized storage and management and, at least under those conditions and when security and privacy are ensured, is also generally a good thing: this is how data is progressively transformed into knowledge. More generally, every standard is a centralization tool; the Internet and the Web currently work because they make everyone use very few protocols (TCP/IP, HTTP, ...) and languages (HTML, XML, RDF+OWL, ...). Yet, most Semantic Web related researchers seem to believe that a Web of "mostly independently created resources" is the only possible way to achieve the distribution of actions and information. This section shows (hopefully) that this is not the case. The next two subsections highlight other related misconceptions. They show that "collaboratively editing a same KB" (i.e., centralization) does not imply that the users have to agree or even discuss terminological issues or beliefs, nor that a committee making content selection or conflict resolution for the users is necessary.

2.2.5. Supporting Collaborative Knowledge Editions Within a KB

Most knowledge servers support concurrency control and users' permissions on files/KBs but only two knowledge servers seem to have special protocols to support collaboration between users: Co4 [Euzenat, 1996] and WebKB-2. (Freebase, Ontolingua, Ontosaurus, Ontowiki, DBpedia ... have no collaboration protocols; Wikipedia is not a knowledge server and has no such protocol either). More generally, WebKB-2 seems to be the only knowledge server having editing protocols that permit, enforce or encourage people to interconnect their knowledge into a *shared KB*, without having to discuss and agree on terminology or beliefs, and while keeping the KB consistent. Co4 had knowledge sharing protocols based on peer-reviewing for finding *consensual* knowledge; the result was a hierarchy of KBs, the uppermost ones containing the most consensual knowledge while the lowermost ones were the private KBs of contributing users. Starting from a shared KB where each statement has many associated creators or believers, a similar hierarchy of KBs could also be generated. The fact that a protocol exploits many KBs or one shared KB is only an implementation issue. It can be changed without changing the spirit of the protocols. However, implementations are likely to be easier and more efficient in a shared KB. Protocols used in other knowledge servers [Lausen et al., 2005] or in knowledge oriented approaches in peer-to-peer networks [Rousset, 2004] or Semantic Grids [Page et al., 2005] focus on managing the integration of a source KB into a private/shared target KB: these protocols are not permitting the users of the two involved KBs to tightly interconnect their knowledge.

The cbwoKB editing protocol used in WebKB-2 are not tied to any particular knowledge representation (KR) language or inference mechanism (hence, in this document, no comparison is made on such mechanisms). This protocol only requires that *conflicts between knowledge representations – i.e., partial redundancies or inconsistencies between terms or statements* – are detected by some inference mechanism or by people. (Hence, the protocol also works with informal pieces of knowledge as long as they can be inter-related by semantic relations). The more conflicts are detected, the more the KB is kept organized and hence exploitable. Section 2.2.5.1 reminds and summarizes the language model for the KB editing protocol used in WebKB-2. Section 2.2.5.2 details this protocol, i.e., the approach used for letting people edit the shared KB. Section 2.2.5.3 quickly compares the approach with some other ones. Section 2.2.5.4 illustrates some technique used in WebKB-2 detecting conflicts and, more generally, comparing statements.

2.2.5.1. Generic Language Model for the KB Editing Protocol

The *model for the protocols* – i.e., their view on a KB (whichever KR language it actually uses) – is a set of *objects* which are either *terms* or *statements*. Every object must have at least one associated source (e.g., creator, interpreter or source ontology) represented by a formal term. A formal object is one that has a unique meaning according to its source. Any non-contextual identifier for a formal object must include an identifier for its source. A *formal term* refers to either a concept type ("class" in RDF), a relation type ("property" in RDF) or an "individual" (an instance of a first-order type, e.g., a particular set, relation, statement or concept). A *formal statement* has an interpretation in some logic and hence has a logic-based syntax. A statement that has a logic-based syntax but includes at least one informal object is *semi-formal*. *Every asserted statement is either a definition or a belief*. Here are some examples in FE, an English-like but formal notation used in WebKB-2. `pm#bird` and `wn#bird` are formal terms for concept types respectively created by `pm` and `wn`, two formal abbreviations (contextual identifiers) in the default KB of WebKB-2 for respectively the author of this document and WordNet 2.0. The informal terms `"bird"`, `en#"bird"` and `pm#"bird"` respectively refer to the string "bird", the English word "bird", and one or several concept types created by `pm` and having for informal name "bird". `u1#u2#"birds fly"` is an informal statement from `u2` that is represented by `u1`. `u1#`any u1#bird is pm#agent of a pm#flight´` is a formal statement and *partial definition* by `u1` of `u1#bird`: it states that to be a bird, a necessary condition is to (constantly) fly. `u2#`every u1#bird is agent of a flight´` is a semi-formal *belief* by `u2` that "every u1#bird is flying". `u3#` `every daylight ?t `every u1#bird is agent of a flight with time ?t´ ´ = u3#edbfdi = u3#every_daylight__birds_flies_during_it´` is a definition by u3 of the terms `u3#edbfdi` and `u3#every_daylight__birds_flies_during_it` as referring to a non-asserted statement representing in a semi-formal way that "every day, every bird is flying during all daylight". Finally, `u3#` `u3#edbfdi is a pm#corrective_specialization of u2#`every u1#bird is agent of a flight´ ´` is a semi-formal belief by `u3`

that his previous statement is a correction and a specialization of u2's belief. In KIF [Genesereth, 1998], a knowledge representation language (KRL) with a second-order notation that can be interpreted in first-order logic, these example statements can respectively be represented as follows.

```
(pm#creator_of  u1 '(defconcept u1#bird (?b) :=> (exists ((?f pm#flight)) (pm#agent ?f ?b))))
(pm#believer_of u2 '(forall ((?b u1#bird)) (exists ((?f flight)) (agent ?f ?b))))
(pm#creator_of  u3 '(=  u3#edbfdi  u3#every_daylight__birds_flies_during_it
   '(forall ((?t daylight)((?b u1#bird))
      (exists ((?f flight)(?t daylight)) (and (agent ?f ?b) (time ?f ?t))
(pm#believer_of u3 '(pm#corrective_specialization
   '(forall ((?b u1#bird)) (exists ((?f flight)) (agent ?f ?b)))    u3#edbfdi) )
```

u3#edbfdi specializes u2's belief. A statement Y specializes a statement X if it uses more specialized terms or structurally contains more information (and hence *either contradicts it or makes it redundant*). Thus, negating a statement is also specializing it. Hence, "generalization between formal statements" does not solely refers to "logical implication". The general "specialization" relation type (for which one identifier is pm#specialization) is applicable to organize all kinds of objects, formal or not, and has many subtypes, including the classic subtype relation type.

As illustrated in Section 2.2.5.4, there are efficient (polynomial) – but incomplete – ways to test if there exists a relation of specialization between two statements, whatever their expressiveness [Mugnier & Chein, 1992] [Chein & Mugnier, 1997]. Searching a specialization hierarchy can be done in $O(N^3)$ comparisons, where N is the number of nodes in the compared statements [Levinson & Ellis, 1992].

RDF/XML (the W3C recommended linearization of RDF) and OWL (the W3C recommended language ontology) are currently not particularly well suited for the cbwoKB editing protocol or, more generally, for the representation or interconnection of expressive statements from different users in a same KB.

- They offer no standard way to associate a believer, creator or interpreter to every object in an RDF/XML file. Since 2003, RDF/XML has no bagID keyword, thus no way to represent contexts and hence believers or beliefs. XML name-space prefixes (e.g., u1:bird), Dublin Core relations and statement reification do not permit to do this. This is likely a temporary only constraint since many RDF-related languages or systems extend RDF in this direction: Notation3 (N3), Sesame, Virtuoso, ...
- RDF and OWL – like almost all description logics – do not permit their users to distinguish definitions from universal quantifications. More precisely, they do not offer a universal quantifier. N3 does (Turtle, the RDF restricted subset of N3, does not). The distinction is important since, as noted in the documentation of KIF (http://logic.stanford.edu/kif/dpans.html#5.3), a universally quantified statement (belief) may be false while a definition cannot. A definition may be said to be "neither true nor false" or "always true by definition". A user u1 is perfectly entitled to define u1#cat as a subtype of wn#chair; there is no inconsistency as long as the ways u1#cat is further defined or used respect the constraints associated with wn#chair. A definition may be changed by its creator but then the meaning of the defined term is changed rather than corrected. This distinction is important for a cbwoKB editing protocol since it leads to different conflict resolution strategies: "term cloning" and "loss-less correction" (Rule R4 and Rule R8 of the next subsection).
- Many natural language sentences are difficult to represent in RDF/XML+OWL or N3+OWL, since they do not yet have various kinds of numerical quantifiers, contexts, collections, modalities, ... (FE has concise syntactic sugar for the different kinds). However, at least N3 might soon be extended.
- Like most formal languages, RDF/XML and N3 do not accept – or have a special syntax for – the use of informal objects instead of formal objects. KRLX does and this permits WebKB-2 to create one specialization/generalization hierarchy categorizing all objects. More precisely, this is an "extended specialization/generalization" hierarchy since in WebKB-2 the classic "generalization" relation between formal objects (logical implication) has been extended to apply to informal objects too.

2.2.5.2. Loss-less Knowledge Integration Protocol

A shared KB is "*ideally well organized with respect to the objects it includes*" if, given any two objects in this KB, given the meaning of each of the two objects in its creator's head and given their various possible relationships, i) all the relationships that can be represented using only already represented objects are represented, and ii) to that end, all already represented objects relevant for representing these relationships are directly or indirectly used. Thus, the most precise objects (e.g., terms) have to be used and there must exist a specialization hierarchy (alias, generalization hierarchy) organizing all the objects. This also entails that the bigger such a KB is, the more precise the objects tend to be. By supporting and guiding knowledge providers in representing knowledge as explicitly as they are willing to, the best practices of Section 2 are an help in aiming towards that ideal KB.

The KB editing protocol described below is meant to keep a shared KB ***at-least-minimally-well-organized*** via the next two "minimal requirements": i) each term (resp. statement) – except the most general one, which may be generated if it is not given – must be *explicitly* related to at least another term (resp. "belief", i.e., asserted statement that is not a definition) via inferred or manually-set relations of generalization or equivalence as well as exclusion (resp. equivalence, argumentation or correction, and if possible, corrective_generalization or corrective_specialization), and ii) manually or automatically detected inconsistencies or partial/total redundancies are prevented or made explicit via these relations. Indeed, these two requirements imply that every object of the KB has a unique place in the global, fully connected, specialization hierarchy of the KB and in the network composed of relations of the above cited kinds. This "unique place" in the specialization hierarchy is a minimal requirement for knowledge insertion and retrieval to be done in a scalable way in this hierarchy and thus in the KB of which it is a backbone [Dromey, 2006]. Rejecting an action introducing an *implicit* partial/total redundancy is also important because this often permits the author of the action to detect a mistake, a bad interpretation or a lack of precision (on his part or not). At the very least, this reminds the users that they should check what has already been represented on a subject before adding something on this subject.

This protocol is not tied to any particular language or inference mechanism. However, it requires the language to support the previously described language model and hence, for example, to distinguish between beliefs and definitions. Regarding inference mechanisms, this protocol only requires that i) the source of each object is explicitly represented in some way, and ii) at least one inference engine tells it if, when an object addition, modification or removal is made in the KB, this creates a *conflict* in this KB and if this conflict is *implicit* or not. *A **conflict** is an inconsistency between statements or a partial/complete redundancy between statements that are **not both** definitions*. Indeed, a detected redundancy between terms or term definitions is not a conflict, it is an inferred equivalence or specialization relation. Apart from this case, if a statement Y is detected as a specialization of a statement X and if the main object of Y is not an instance of the main object of X, there is a detected conflict. ***Implicit*** means *not explicitly represented by an inferred or manually set relation which* i) between beliefs, may be a relation of equivalence or correction, and ii) between definitions, and hence between terms too, may be a relation of equivalence, specialization or exclusion. Since the integration is loss-less, there is no reason why the order in which users enter knowledge should affect conflict detection and hence should be of importance.

Table 2.2.5.2.1 gives the high-level algorithm of the protocol in an object-oriented Java-like syntax (for clarity purposes). Then, explanations are given via a list of informal rules. Table 2.2.5.2.2 summarizes the cases related to conflict detections. This algorithm checks on a user's attempt to remove or add a statement, and rejects the action ("return false") or accepts it. It may perform some automatic repair step before accepting the action. It only allows statement removal and addition since i) an update is considered as a removal followed by an addition (WebKB-2 and the informal rules also consider direct updates), ii) reading or re-using an object is always accepted (privacy control is not dealt with in this document), and iii) term removal or addition must be made via the removal or addition of a statement (Rule R2 below). Like Rule 2, Rule 12 is implicitly enforced because the algorithm only uses the generic knowledge model described in Section 1.1. From now on, the word "user" is used as a synonym for "source". Furthermore, u1 and u2 are used as examples of different users.

Table 2.2.5.2.1. Algorithm for a cbwoKB editing protocol

```
//The informal rules enforced by some steps of the following functions are referred to via comments

boolean statement.removal_by (User agent)     //true is returned to accept the removal
{ if (this.creator != agent) return false;                                                    //rule R1
   if (agent.created_statements_without(this).are_conflicting()) return false;                 //R3
   if (KB.statements_without(this).are_conflicting())
   { if (this.is_definition_of_existing_term_created_by(agent))
      { s= KB.statements_of_other_users_using_this_term_and_conflicting_with_this(agent,term,this);
         s.clone_this_term_for_and_in_these_statements(this.defined_term());                  //R4
      }
      else
      { s= KB.statements_of_other_users_using_this_belief_and_conflicting_without_it (agent,this);
        s.clone_this_belief_for_and_in_these_statements(this);                                //R9
      }
   }
   KB.remove(this);   return true;
}

boolean statement.adding_by (User agent)    //true is returned to accept the addition
{ if (agent.created_statements_with(this).are_conflicting()) return false;                    //R3
   if (KB.statements_with(this).are_conflicting())
   { if (this.is_definition_of_existing_term_created_by(agent))
      { s= KB.statements_of_other_users_using_this_term_and_conflicting_with_this(agent,term,this);
        s.clone_this_term_for_and_in_these_statements (this.defined_term());                  //R4
      }
      else if (! KB.statements_with(this).have_only_explicit_conflicts_between_beliefs() ) )
            return false;                                                          //R5, R6, R7 or R8
   }
   else if (this.is_definition_of_a_new_term())
   { if (!this.is_connected_to_at_least_another_term_via_a_relation_of_one_of_the_following_types
                                        ("pm#equivalence", "pm#generalization", "pm#exclusion")
        return false;                                                                         //R11
   }
   else if (this.is_a_belief())
   { if (!this.is_connected_to_at_least_another_belief_via_a_relation_of_one_of_the_following_types
                                         ("pm#equivalence", "pm#correction", "pm#argumentation")
        return false;                                                                         //R10
   }
   KB.add(this,agent);   return true;
}
```

Table 2.2.5.2.2. Summary of the conflict related cases
(with s1 and s2 (resp. t1 and t2) being statements (resp. terms) respectively created by u1 and u2)

Action by u1	*Conflicting with*	**Rule**
add/delete s1	a statement of u1 → reject	R3
add/delete definition of already existing t1	s2 → clone t1 for u2 (and in s2 if needed)	R4
add definition to new t1	s2 → reject	R5
add definition to t2	s2 → reject	R6
add belief b1	definition of t2 → reject	R7
add belief b1	s2 being a belief → reject if the conflict is implicit	R8
delete belief b1	s2 → clone b1 for u2	R9

Here are the informal rules enforced by this algorithm.

R1 (object update only by its creator). Any user may add and use any object, and hence may for example add a relation between objects he has not created, but *an object may only be modified or removed by its creator*.

R2 (operations on terms via operations on statements). *Adding, modifying or removing a term* is done by adding, modifying or removing at least one statement (e.g., one relation) that uses this term.

R3 (no conflict between statements of a user). *If a user adds, modifies or removes a statement that introduces a detected conflict between statements believed by this user*, this *action is rejected* by the system. Thus, in the case of an addition, the user must refine his statement before trying to add it again or he must first modify at least one of his already entered statements.

R4 (cloning of updated terms used by other users in a way not consistent with the new definition). *If the addition, modification or removal of a statement defining an already existing term u1#T* by a user u1 *introduces an inconsistency involving statements directly or indirectly re-using u1#T and created or believed by other users* (i.e., users different from u1), *u1#T is automatically cloned* to solve this conflict and ensure that the original interpretation of u1#T by these other users is still represented. Indeed, such a conflict reveals that these other users had a more general interpretation of u1#T than u1 had or now has. Assuming that u2 is this other user or one of these other users, the term cloning of u1#T consists in creating u2#T with the same definitions as u1#T except for one, and then replacing u1#T by u2#T in the statements of u2. The difficulty is to choose a relevant definition to remove for the overall change of the KB to be minimal. In the case of term removal by u1, term cloning simply means changing the creator's identifier in this term by the identifier of one of the other users (if this generated term already exists, some suffix can be added). In a cbwoKB server, since statements point to the terms they use, changing an identifier does not require changing the statements. In a global virtual cbwoKB distributed on several servers, identifier changes in one server need to be replicated to other servers that use this identifier. [Djedidi & Aufaur, 2009] proposes an ontology pattern using term cloning for knowledge integrations that are not loss-less.

R5 (no new term with a definition conflicting with statements of other users). *If the addition of a new term u1#T* by a user u1 *introduces an inconsistency with statements of other users*, this *action is rejected by the system*. Indeed, such a conflict reveals that u1 has directly or indirectly used – and misunderstood – at least one term from another user in his definition of u1#T. Hence this conflict reveals an incorrect belief of u1 and thus, as in R8, this conflict should be rejected.

R6 (no new definition to another user's term if the definition conflicts with another statement). The *addition by a user u2 of a definition to u1#T implies a belief of u2* about the meaning of u1#T. Thus, this addition should be *rejected if it*

conflicts with another statement (from u1 or not since no term cloning is not worth to done here: u2 can simply create and/or use another term).

R7 (no conflict between a new belief and a definition). *If adding or modifying a belief introduces an implicit conflict with a definition*, this action is *rejected*, whether or not the definition is about a term created by the author of the definition. Indeed, as in the previous rule, no term cloning is worth to done here.

R8 (no implicit conflict when adding a belief). *If adding a belief introduces an implicit conflict involving beliefs created by other users*, this action is *rejected*. However, a user may still represent his belief (say, b1) – and thus "loss-less correct" another user's belief that he does not believe in (say, b2) – by *explicitly* connecting b1 to b2 via a *corrective relation*. This has already been illustrated but here are two other examples of "corrections to solve conflicts": i) `u2#` u1#`every bird is agent of a flight´ has for pm#corrective_specialization u2#`every healthy flying_bird is able to be agent of a flight´ ´`, and ii) `u2#` u1#`every bird can be agent of a flight´ has for pm#corrective_generalization u2#`75% of bird can be agent of a flight´ ´`. In the first case, u2's belief specializes u1's belief and corrects it. In the second case, u2's belief generalizes u1's belief and corrects it.

R9 (cloning of a belief if its update conflicts with other users' statements). *If modifying or removing a belief introduces an implicit conflict with other users' statements, the belief is cloned: its associated creator becomes one of these other users.*

R10 (no addition of a belief without at least one relation of equivalence, correction or argumentation to another belief). This is particularly important for informal beliefs since conflicts between them and the rest of the KB are difficult to detect. It is preferable to use relations of type `pm#corrective_generalization` or `pm#corrective_specialization` than non-transitive correction relations, and preferable to avoid using argumentation relations whenever possible. A most general belief is predefined: `` `a pm#thing´ `` ("there exists something").

R11 (no addition of a term without at least one relation of equivalence or generalization as well as exclusion to another term). If these two relations can be automatically inferred based on the definitions of the new term, there is no need to add them manually. A most general term (`pm#thing`) is predefined.

R12 (no other asserted objects than beliefs or term definitions). If a set or collection is asserted, it should be via the definition of an instance to a collection type (e.g., `pm#set`). A query – and even a query operator – can be described via a term definition. The purpose of this rule is for the protocol to cover all objects of the KB, for it to be *at-least-minimally-well-organized*, for the same mechanisms to work on all objects and for knowledge to be represented in an explicit way, even if some objects are represented informally.
In particular, evaluations by individual users on the interest (originality, importance, veracity, ...) of other user's objects should be done via meta-statements. Thence, these evaluations can be exploited in user-defined queries to filter knowledge when browsing the KB or when generating modules for an application. Section 2.2.6 gives a framework for possible "default evaluation measures".
More generally, methods to measure the average interest of an object based on these evaluations – or the interest of a knowledge provider based on the average interest of its statements – should be represented in an explicit way, ideally in a declarative way. Thus, each user can easily create measures tailored to his needs by deriving them from other measures (e.g., the default measures proposed by a cbwoKB) and these measures can be organized into the specialization hierarchy (they are function definitions, hence term definitions).
This approach may be seen as the beginning of a technical vision for the very general "model of discursive practice" of [Brandom, 2000]. These measures do not have to be in a rule-based language, they may be in a functional language or in a command-based script language. WebKB-2 proposes the language FC (For Control) which permits users to combine query/assertion commands via shell script like control structures (loops, pipe, procedure declaration). The arguments of its query/assertion operators can be knowledge objects in any of the accepted notations (FE, FL, FCG, KIF, ...).
Ideally, the knowledge presentation used by the above cited measures or by the query operators of a cbwoKB should be specifiable by the user. This is partly the case in WebKB-2: users can use FC to change the value of predefined "presentation variables".

2.2.5.3. Notes and Comparison with some other approaches

The approach of this protocol leads to a loss-less integration of knowledge, based on the interconnection of objects, preferably undecomposable ones. Relating multi-statement objects has not been forbidden by a rule in order to ease incremental knowledge refinement but this constraint could be added. In any case, an object can be decomposed objects whenever correction relations or other meta-statements have to be specified about a statement that is part of this object. Thus, unlike approaches based on physically separated multi-statement modules, this approach does not have to deal with the problems associated to the use of different versions for such modules. This approach may be used by current knowledge integration methods to save their results into a cbwoKB instead of creating new KBs partially redundant with the ones they use as sources. Conversely, these methods can guide loss-less integration.

The approach followed by the protocol is unrelated to the approaches of defeasible logics based works.

This approach also solves some governance and scalability problems since the KB stays "at least minimally well organized" without restricting users on content or terminology choice. Previous sections have presented problems that current shared KB servers (semantic wikis included) suffer from due to not having such a protocol.

The approach of the protocol assumes that all beliefs can be argued against and hence be "corrected". This is true only in a certain sense. Indeed, among beliefs, one can distinguish "observations", "interpretations" ("deductions" or "assumptions") and "preferences". Although all these kinds of beliefs can be false (their authors can lie, make a mistake or assume a wrong fact), most people would be reluctant to argue against self-referencing beliefs such as `u2#"u2 likes flowers"` and `u2#"u2 authored this sentence"`. The editing protocol of WebKB-2 relies on this reluctance to argue against such beliefs that should generally not be argued against.

Before browsing or querying the shared KB, a user can set "filters for certain objects (categories or statements) not to be displayed". These filters may set conditions on statements about these objects or on the creators of these objects. Filters are useful whenever the KB is not sufficiently organized for a user to avoid being overwhelmed by the large amount of information on a certain point.

The above described editing protocols – especially via the specialization and corrective relations that they lead users to set – encourage or enforce a minimal re-use, precision or connectivity between objects: they ensure the "minimal semantic organization" described in Subsection 1.1.2. They also permit a loss-less information integration approach. Interesting aspects of this approach is that it works for semi-formal KBs and is incremental:

- Even if WebKB-2 cannot detect an inconsistency or a redundancy, the users can (and are encouraged to) set corrective or specialization relations between formal or informal statements.
- When an addition, modification or removal action is performed, the creator of a cloned term, statement using a cloned term, specialized term or statement, or corrected statement, is warned via email if she is not the one having performed the action and if she has authorized such warnings. This can lead her to modify or remove some of the objects she created.

This approach can be seen as a precise asynchronous dialog between the knowledge providers. The summarizing expression used at the beginning of this subsection is now more understandable: "editing protocols that permit, enforce or encourage people to interconnect their knowledge into a shared KB, without having to discuss and agree on terminology or beliefs, and while keeping the KB consistent".

The next subsection complements this one by allowing the exploitation of argumentation relations (e.g., corrective relations) in valuations of creators and created statements. Thus, the remarks on the interest and possible applications of the above described approach also apply to – and are re-inforced by – the (results of the) techniques of the next subsection. Since the approach described in this subsection and the next one works on semi-formal KBs, it is a solution to some problems that most shared information repositories have, e.g. wikis.

- These repositories lack semantic organization and hence not only include numerous implicit redundancies and inconsistencies but have their scope restricted because of presentation related issues (this includes avoiding users

to experience information overload by restricting the scope instead of structuring the content). For example, the 1-article-1-subject approach of Wikipedia does not restrict its number of subjects but restricts each article to be relatively short and led the creators of Wikipedia to restrict it to be of "encyclopedic nature only" hence "avoiding content of technical or research nature" and "using prose rather than tables". This has for example led the selection committee of Wikipedia to remove some articles on the ground that they were "too technical" or used "too many tables", even when tables were the most suited informal structure, e.g., when they were used for compared objects (e.g., CG tools) according to some of their characteristics.

- For similar reasons, even if an object is within the scope of the repository, the selection committee may chose to discard it on the ground that it may not be of of interest to most users of the repository. In the approach described in this subsection and the next one, people use knowledge representations to argue for or against a statement or to value precise characteristics of this statement, and any user may design her own filter to see or not certain kinds of statements, e.g., statements that have been found obvious by certain kinds of persons. This approach would permit to merge (the information discussed or provided by the members of) many communities with similar interests, e.g., the numerous different communities working on the Semantic Web.
- When information repositories do not have a selection committee, they often allow any user to remove any statement. This is discouraging many potential authors and lead others to engage in "edit wars".

From an application viewpoint, the approach seems interesting for collaboratively-built states of the art, corporate memories, catalogs, e-learning, e-government, e-science, research, etc.

This approach – which exploits and encourages the representation of undecomposable objects – presents a viable and better alternative to the classic "module based approach", i.e., the storage of knowledge in separate files, KBs or large contexts, in formal or informal information repositories. It is viable because the information stay minimally organized and hence information overload can be avoided. It is a better alternative because it leads to more relations between objects. Indeed, in knowledge libraries, e.g., existing ones such as the Ontolingua library or imagined ones such as "The Lattice of Theories" [Sowa, 2000, 2005], the "module based approach" is intended to create "minimal and internally consistent theories" to maximize their re-use; however, this also leads to few relations between objects of different modules, as well as implicit redundancies or inconsistencies between them, and hence more difficulties for module creators or module users to compare, merge or relate (objects of) different modules. Furthermore, the isolation of knowledge into large modules is often arbitrary. On the other hand, if needed for some applications, such modules and their inclusion relationships could be automatically generated from the above described shared KB, based on the relations between objects and some criteria for dividing knowledge into modules. This was acknowledged by [Sowa, 2003]. Regarding module generation for inferencing purposes, see [Le Pham et al., 2008].

This shared KB approach removes or reduces many problems related to the existence of (module) versions. If a user adds or removes a definition to a category, she associates a new meaning to the same category identifier. If this leads to manually or automatically detected redundancies or inconsistencies, manual or automatic cloning can solve the problems. To permit this cloning to be replicated in other KBs (e.g., via the mechanism described in the previous subsection), the occurrence of this cloning should be represented. For example:

```
pm#category_cloning  input: wn#bird,  output: oc#bird,  time:  21/01/2005;
```

If the cloning has been done automatically, such a statement can be generated automatically. However, to prevent such cloning, whenever possible it is preferable to use precise category identifiers from the beginning, e.g., wn#carinate___carinate_bird___flying_bird. However, since contexts can be used to set constraints on a relation, it is not necessary to declare identifiers such as pm#Paris_as_the_capital_of_France_from_1990_to_2000. It is actually better to avoid such an identifier since, although it is possible to declare it as an extended specialization of wn#Paris___French_capital___capital_of_France (which is an instance of the first order type wn#national_capital), very few tools allow and handle specializations of individuals.

The hypothesis that this shared KB approach relies on are that:

- conflicts can always be solved by adding more precision until they boil down to inconsistent "observations", "interpretations" or "preferences",
- solving inconsistencies and redundancies tends to increase or maintain the precision and organization of the KB,
- different, internally consistent, ontologies do not have to be structurally modified to be integrated (strongly inter-related) into a unique consistent semantic network.

The author of this document has not encountered any ontology integration or mapping that invalidated these hypothesis (in each case, they seemed true). The least conceptually straightforward mapping that he made was between DOLCE and OCHRE, two general top-level ontologies that were somewhat similar but based on different ontological assumptions. Mapping them required the introduction of intermediary categories, as illustrated in the following example. Its approach was validated by the authors of DOLCE.

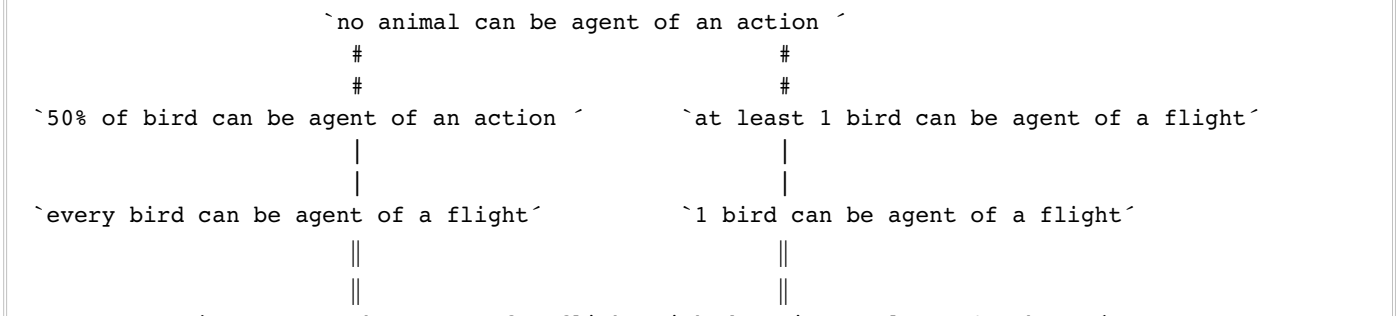
```
metaOchre#set_of_DOLCE/OCHRE_types___structure_for_translation_from_OCHRE_to_DOLCE
  <  pm#non-empty_set_of_types,
  member:  {(metaOchre#PT  metaOchre#P  metaOchre#F  metaOchre#C  metaOchre#A  metaOchre#SI
             metaOchre#CM)}
           {(metaDolce#ED  metaDolce#PD  metaDolce#Q  metaDolce#T  metaDolce#T
             metaDolce#P_2  metaDolce#P_3  metaDolce#K  metaDolce#PC  metaDolce#qt
             metaDolce#ql)};
metaOchre#PT
  definition: "non-empty set of OCHRE's particulars",
  member:  ochre#PT __[any->?, ?<-any];
metaOchre#TH
  definition: "non-empty set of DOLCE's endurants",
  = metaDolce#ED,
  member:  (pm#ochreTH_member_of_metaOchrePT  < ochre#TH,
             member of=>  metaOchre#PT __[any->?])  __[any->?, ?<-any],
  member:  dolce#ED __[any->?, ?<-any];
metaOchre#TK
  member:  (pm#ochreTK_member_of_metaOchrePT  < ochre#TK,
             member of=>  metaOchre#PT __[any->?])  __[any->?, ?<-any];
metaDolce#PD
  definition: "non-empty set of DOLCE's perdurants",
  member:  (pm#something_with_member_a_member_of_metaOchreTK
             member=>  (pm#something_member_of_metaOchreTK
                          member of=>  metaOchre#TK __[any->?]
                        ) __[any->?] ) __[any->?, ?<-any];
//and so on for the other members of metaOchre#set_of_DOLCE/OCHRE_types
```

2.2.5.4. Example of technique for object comparison and conflict detection

WebKB-2 detects *potential conflicts between two statements* by detecting exclusion and specialization relations between (parts of) them. If one of the statements specializes the other, there is a potential conflict. In WebKB-2, a statement Y is detected as being a *specialization* of a statement X (i.e., Y includes the information of X and hence *either contradicts it or makes it redundant*) if X structurally matches a part of Y and if each of the terms in this part of Y is identical or a specialization of its counterpart term in X. Six examples of relations of specialization or exclusion (between statements) that WebKB-2 can detect that way are given by Table 2.2.5.4.1. Two of the six examples illustrate an "instantiation" relation, i.e., a specialization relation where one of the types in the source statement is replaced by an instance in the destination statement. Such a relation does not reveal an inconsistency or a total/partial redundancy that needs to be made explicit, since adding an instantiation can be seen as giving an example for a more general statement. Hence, its detection does not lead the protocol to reject a statement. Apart from this case, if a new belief specializes, generalizes or is equivalent to an already stored belief, and if they are not connected by a correction relation, there is an implicit "inconsistency or partial/total redundancy" between them and the new belief is rejected.

Table 2.2.5.4.1. Examples of specialization/exclusion relations (between statements) detected by WebKB-2

: exclusion relation; -------- : specialization relation; ===== : instantiation relation



In KIF, `Tweety can be agent of a flight ´ may be written:

```
(modality  `(exists ((?f flight)) (agent ?f Tweety))  possible)
```

The above cited matching takes into account numerical quantifiers and measures, not just existential and universal quantifiers. Apart for this, it is similar to the classic graph matching (or "projection") in Conceptual Graphs, which is performed on "positive conjunctive existential formulas, with or without non-restrictive associated outer-contexts". When performed on such statements, this classic graph matching is sound and complete, and can be computed with a polynomial complexity if the query graph (i.e., X in the above description) has no cycle [Mugnier & Chein, 1992] [Chein & Mugnier, 1997]. When performed on other kinds of statements, graph matching for detecting a specialization is not always sound and complete. However, this operation works with languages of any expressiveness and the results of searches for specializations of a query graph are always "relevant for knowledge retrieval purposes". For example, if a query statement is "a bird that is agent of a flight", the answer "no bird can be agent of a flight" is a relevant one.

If two types are not related by exclusion relations, any specialization of one type may also be a specialization of the other. Since it is often not easy to explicitly relate a term to all its generalizations in a KB, a search for the specializations of an object may not retrieve all the possible specializations of that object. This is why, for knowledge retrieval purposes, it is often interesting to search objects which are not exclusive with the query object, rather than looking for its specializations and generalizations. For such querying and other knowledge inferencing purposes, ideally, all the exclusion relations between the direct subtypes of a type should be made explicit. The simplest way to do that is to create direct subtypes only via the use of open/complete subtype partitions. This is one of the "knowledge representation best practices" advocated to the users of WebKB-2. It has not been included as a rule to be enforced by the previously described protocol but it could be.

2.2.6. Supporting the Valuation and Filtering of Knowledge or Knowledge Sources

To establish and display a valuation for an object (product, text, statement, person, etc.) current information management systems

- permit people to relate this object to a free-text message (via argumentation relations in the case of argumentation tools and certain hypertext or knowledge-based systems) or grade some of its characteristics using a scale or a number,
- associate to each object the list of the free-text messages as well as some statistical measures (sum, average, etc.) based on the values for each of the characteristics, and
- order the objects according to these statistical measures.

Subsection 2.2.1 cited Knowledge Zone as an example for a system grading ontologies, and listed many reasons why grading multi-statement resources and making statistical measures about these gradings compounds several problems. Some of these problems apply to "multi-criteria decision making" since they order objects according to many criteria, except in the rare case where an object is better than all the other objects for all the considered criteria. Here are some relations between some knowledge valuation related processes.

```
supporting_the_valuation_of_knowledge_or_knowledge_authors
  >  {supporting_a_precise_valuation_of_knowledge_statements_or_authors
      supporting_free_text_annotations}
     supporting_the_association_of_grades_to_knowledge_or_knowledge_authors
     ordering_knowledge_or_knowledge_authors_according_to_their_valuations,
  subprocess:
     exploiting_knowledge_representations_for_valuating_knowledge_statements_or_authors
     valuating_how_consensual_the_input_statement_is
     valuating_the_interest_of_the_input_statement
     valuating_the_usefulness_of_the_input_statement
     valuating_the_usefulness_of_the_input_user;
```

It is not possible to avoid loosing information when creating values synthesizing the opinions of several persons. Hence, as opposed to the techniques of the two previous subsections, the approach presented in this subsection is not loss-less. However, it is meant to reduce unintended loss of information in the valuation process. First, it exploits valuations via knowledge representations and starts from "individual valuations" (i.e., valuations by each person as opposed to "global evaluations" which are already statistical measures on individual evaluations). Second, it encourages the creation of true and precise "individual valuations", via (preferably formal) statements rather than via votes. Third, it allows each user to specify its own method to calculate a global evaluation instead of using the default global evaluation method. Thus, this approach is more a template than a fixed method. It, and more generally all the approaches useful to create a cgosSW, are complementary to multi-criteria decision making techniques in the sense that they permit to gather more precise information than other methods and hence provide better inputs to these techniques when they remain useful helps for making decisions.

It is important to distinguish three complementary kinds of techniques that are called "argumentation techniques" as well as "decision support techniques". One kind is about multi-criteria decision making techniques. A second kind is about logics supporting (or permitting to explain) certain kinds of argumentations, e.g., defeasible reasoning and logics for legal argumentation. A third kind is about "argumentation systems" based on semantic networks of users' beliefs connected by argumentation relations, such as those used in hypertext argumentation tools (e.g., ArguMed, SIBYL, gIBIS, ArgNoter and the ontology-based ScholOnto) or in the following theories described in Wikipedia: argumentative dialogue, critical thinking, computer-supported collaborative argumentation, argumentation-based design rationale and Brandom's model of discursive practice. Some systems of the third kind are also of the second kind. By allowing the creation of argumentation structures and more generally a cgosSW, and by proposing a technique to value contributions and contributors, WebKB-2 belongs to the third kind.

WebKB-2 permits its users to create meta-statements for arguing for or against a statement or for representing various criteria on a statement, e.g., on its originality.

```
wn#valuation  //an indirect subtype of pm#process
  >  {pm#individual_valuation __[pm]  pm#global_valuation _[pm] }  wn#believing
     valuating_how_consensual_the_input_statement_is
     valuating_the_interest_of_the_input_statement
     valuating_the_usefulness_of_the_input_statement
     valuating_the_usefulness_of_the_input_user;
wn#believing .(agent: ?a, object: ?o)
  >  pm#finding_the_destination_statement_interesting,
  agent: (pm#causal_entity  pm#believer of: pm#description_content/medium/container __[?a->?o]
         ) __[any ?b -> 1..* ?a],
  object: pm#description_content/medium/container __[?b -> 1..* ?o];
valuating_the_usefulness_of_the_input_statement .(input: 1..* ?i, output: ?o)
  agent:  1..* pm#causal_entity ?a,   //i.e. "agent: pm#causal_entity __[any ?v->1..* ?a]"
  input:  1..* pm#description_content/medium/container ?i,   //i.e. "... __[?v->1..* ?i]"
  output: (wn#usefulness ?o  pm#measure of: ?i) __[pm#author: ?a];
```

However, the shared KB edition protocols do not exploit any measure of the "usefulness" of each statement, a value that would represent its "global interest", acceptation, popularity, originality, etc. Yet, this seems interesting for a knowledge repository, especially for argumentation structures or semi-formal discussions (see the examples given in Table 2.1.1.22.1 and in Section 4.1). Indeed, statements that are obvious, not argued, or for which each argument has been counter-argued, should be marked as such (e.g., via darker colors or smaller fonts) in order to make them less visible (or invisible, depending on the selected display options) and thus discourage the entering of such statements.

General idea. When the statements of a user have individual measures of usefulness associated to them, these measures can be used (among other things) to measure the global usefulness of that user. Conversely, the global usefulness of a user can be used (among other things) to weight the individual measures that she creates. This feedback loop should help highlight "good" contributions (well argued, interesting, ...) and this highlighting should encourage them.

Here are descriptions of the *default* valuation measures that will first be implemented in WebKB-2. Then, users will be allowed to use FS to modify these measures by making them take into account additional elements and use functions designed by the users.

- **Global measure of how consensual a statement is.** Since it does not make sense to believe in a definition, it cannot "directly" be consensual. However, i) the *relation of type pm#name (or pm#extended-specialization)* between a word and a (defined) category can be consensual, and ii) a measure of how consensual this relation is can also take into account how many times the category is *used in statements not authored by the creator of the category*. An observation, interpretation or preference can be shared (i.e., can have many *believers and/or argumentation relations* connected to it) and hence can be consensual. Any statistical measure to sum and weight the three above cited kinds of relations (the ones highlighted with italic characters) would be arbitrary. Regarding the taking into account of argumentation relation, here is one default function that is considered for implementation in WebKB-2.
    - Use 0 to value how consensual a belief that has no argument nor counter-argument connected to it (examples of counter-argument relation types: pm#counter-example, pm#counter-fact and pm#corrective_specialization).
    - Use 1 (to specify that the belief is "confirmed") if i) each of its arguments has a consensual value of 0 or 1, and ii) it has no confirmed counter-argument.
    - Use -1 if the belief has at least one confirmed counter-argument.
    - Use 0 in the remaining case: no confirmed counter-argument but each of the argument has a consensual value of -1.

- **Global measure of how interesting a statement is.** This measure should take into account the individual measures of interest for this statement by users of the KB. These individual measures can be represented using the above cited pm#finding_the_destination_statement_interesting process type (this can then be considered as a binary kind of votes by the users) or take into account similar kinds of votes (or more precise valuations) on more precise attributes such as "originality" and "acceptation".
  The types of the beliefs (typically, pm#observation, pm#deduction, pm#assumption and pm#preference), when they are made explicit, can also be used as a factor to calculate the global usefulness of a statement (e.g., by considering that a deduction is more interesting than an observation).
  In any case, the *usefulness of the valuating users* (see the last point below) may be taken into account to weight their votes or representations. Here is one candidate for a default function to calculate this usefulness.
    - Let any user give a value to the interest of a statement, say between -5 and 5 (the maximum value that the creator of a statement may give to it is, say, 2).
    - Multiply this value by the usefulness of the valuating user to obtain her "weighted individual interest" in that statement.
    - Average the weighted individual interests to obtain the "global interest" of the statement.

  Because of the weights, this function may be seen as a simple multi-valued voting system where more competent people in the domain of interest are given more weight (a more elaborate social network based voting system can be found in [Rodriguez et al., 2007]). With this function, a statement that does not deserve to be visible (e.g., because it is clearly a particular case of a more general statement) is likely to receive a negative global interest. It seems preferable to let each user explicitly give an interest value rather than taking into account the way the statement is generalized by – or connected to, or included in – other statements because deducing an interest value from the existence of such relations is difficult. For example, a belief that is used as a counter-example may be a particular case of another belief but is nevertheless very interesting as a counter-example.

- **Global measure of the usefulness of a statement.** This measure should take into account the above two measures for this statement. Here is one function that is considered for implementation in WebKB-2.
    - If the consensual value of the statement is equal to 1, its global (measure of) usefulness is equal to its global interest.
    - Otherwise, the global usefulness of a statement is equal to its consensual value (i.e., provided that only argumentation relations are taken into account, -1 or 0; indeed, a belief without argument may be considered as having no usefulness, whether it is itself an argument or not).

- **Global measure of the usefulness of a user.** This measure should take into account the global measures of usefulness of the statements of the user and may also take into account her participation to valuating statements of other authors. Here is one function that is considered for implementation in WebKB-2:

```
sum of the global usefulness of the statements from the user +
 square root (number of times the user valuated statements of other users)
```

  The second part of this equation acknowledges the participation of the user in valuations while decreasing the weight of these valuations as their number increases. Functions decreasing more rapidly than square root may perhaps better balance the value of the two processes: contributing information and valuating information.

It is clear that the above points constitute only a framework that should be refined. However, even if the above described simple functions are used for the default valuations, it may be hoped that this framework should incite the users to be more careful and precise in their contributions (affirmations, arguments, counter-arguments, etc.) and give arguments for them. Indeed, unlike in traditional discussions or anonymous reviews, careless statements here penalize their authors. This may lead users not to make statements outside their domain of expertise or without verifying their facts. (Using a different pseudo when providing low quality statements does not seem to be an helpful strategy to escape the above approach since this reduces the number of authored statements for the first pseudo). Since counter-arguments must be justified, it may also be hoped that this framework avoid the under-rating of "correct but outside-the-main-stream contributions". Finally, when a belief is counter-argued, the usefulness of its author decreases, and

hence this information provider is incited to deepen the discussion or remove the faulty belief. However, there is still a need for some specially privileged users to remove "completely irrelevant statements" (spams) that have been marked as such by some users (and hence that were not prevented by the constraints of the shared KB edition protocols).

In his description of a "Digital Aristotle", [Hillis, 2004] describes a "Knowledge Web" to which researchers could add ideas or explanations of ideas "at the right place", and suggests that this Knowledge Web could and should "include the mechanisms for credit assignment, usage tracking, and annotation that the Web lacks" (pp. 4-5), thus supporting a much better re-use and evaluation of the work of a researcher than what the current system of article publishing and reviewing permits [AFIA, 2002]. However, Hillis does not give any indication on such mechanisms. The approaches presented in this subsection and the two previous ones may be seen as frameworks for such mechanisms. Complementary approaches, e.g., the module-based approach of Co4 for finding *consensual* knowledge were also cited.

The above described framework may also be seen as the beginning of a technical vision for Brandom's very general "model of discursive practice" [Brandom, 2000]. To support Brandom's vision, it is necessary that the users are able to define their own valuation functions, and it is necessary to exploit an inference system for allowing each user to test the consequences of adding certain statements to the network and for allowing each user to change the "rules of the game for evaluating the interest and veracity of certain statements". The authors of [Keeler & Majumdar, 2008] work on Brandom's statement valuation "game" based on i) the synthesis of [Holland, 1998] about (natural or artificial) mechanisms valuating, selecting or composing facts, methods or genes, ii) an automatic extraction of simple IF-THEN rules from natural languages sentences, and iii) a measure of similarity between these rules.

Although independently developed, the above described framework appears to be an extension of the version designed for SYNVIEW [Lowe, 1985]. In this hypertext system, statements had to be connected by (predefined or user-invented) relations and each statement was valuated by users (this value, and another one calculated from the value of arguments and counter-arguments for the statement, were simply displayed near the statement in order to "summarize the strengths assigned to the various items of evidence within the given contexts"). In 1986, the authors of SYNVIEW removed the constraint of using explicit relations between statements (the statements still had be organized hierarchically but the relations linking them were unknown) and replaced the possibility of grading each statement by the possibility of ranking them within the list of (sibling) statements having a same direct super-statement. This change was made in the hope of easing information entering and thus hopefully permit the collaborative work of a small community towards the creation of an information repository large enough to interest other people and lead them to participate and store information too. A similar move away from structured representations was described in [Buckingham-Shum et al., 1999] for the same reason and the idea of making the approach more "scalable". Although such a move clearly makes information entering easier, Subsection 2.2.1 showed that it actually makes the system far less scalable. Such moves apparently failed to attract more interest than the original more structured approaches.

The next section lists and organizes "best practices to represent knowledge in more precise, organized and normalized ways".

2.3. Following Normalization Rules or Best Practices When Representing Knowledge

This section lists some rules to normalize knowledge and thus ease its exploitation with simple methods but without restricting the expressiveness of the languages. These rules can be seen as "best practices" and are complementary to those of [Rector & Rogers, 2006] and those of the "Semantic Web Best Practices and Deployment (SWBPD) Working Group" [Swick et al., 2006]. The lexical or structural normalization rules given below may also be seen as ontological normalization rules since, for example, they cite certain relation types. However, unlike the content of Chapter 3 which organizes *many* conceptual categories that are important for representing knowledge in a *normalized* way, these lexical or structural normalization rules do not give any ontology.

```
representing_knowledge
  <  knowledge_extraction/modeling/representation,
  >  (following_a_style/rule/method_when_representing_knowledge
       >  (following_a_precision-oriented_normalization_rule_when_representing_knowledge
            definition: "following a knowledge representation style/rule/method that
                         i) reduces the number of non-easily-automatically-comparable ways
                             information can be represented (with the criteria for an
                             easily automatically comparable way being a unique graph matching
                             process), or
                         ii) leads to representing a piece of information in a way that
                              permits to generate (and hence be compared with) the
                              representations that are created if that method is not followed.
                         The more precise or organized the representations, the
                         more normalized they are because the more they make explicit common
                         basic components (e.g., basic relations) that permit comparisons
                         with other representations. This also eases readability and
                         understanding",
          >  following_a_lexical_normalization_rule_when_representing_knowledge
             following_a_structural_normalization_rule_when_representing_knowledge
         )
         using_a_normalized_and_scalable_input_file_structure_when_representing_knowledge
     );
```

2.3.1. Lexical Normalization

```
following_a_lexical_normalization_rule_when_representing_knowledge
  >  (following_a_lexical_normalization_rule_for_category_identifiers
       > following_a_loss-less_category_naming_style
         using_singular_nouns_or_nominal_expressions_for_category_identifiers
         ending_second_order_concept_types_by_class_or_type
         ending_second_order_relation_types_by_relation/function_type_or_property )
     following_a_lexical_rule_for_an_informal_category_annotation/definition;
```

2.3.1.1. Following a loss-less category naming style. Paragraph 2.1.1.24 states that any new identifier following the "W3C category naming style" is converted into the "loss-less category naming style" whenever i) there is a way to quickly convert it back to the W3C category naming style for knowledge export purposes, and ii) this does not lead to a lexical conflict. Indeed, the W3C category naming style is not loss-less and is less readable. Furthermore, using the "loss-less category naming style" with European languages often permits to lexically distinguish between types and individuals since these last ones are often denoted by words with a capitalized first letter. This is as important as lexically distinguishing concept types from relation types (as done with the W3C category naming style).
For many reasons, it would be simpler for people if XML – and hence RDF and then most KRLs since most KRLs are now meant to be translated in RDF/XML – were case insensitive and if the '_' or '-' characters were ignored. Similarly, OWL has different lexical scopes for concept types and relation types; this can be seen as a design flaw by some persons since this makes things harder for the programmers (especially since relation types, like any other kinds of types, should be allowed to be described and hence should be allowed to be used in concept nodes) and it does not

make life simpler for the users. Protégé-2000, a popular editor for creating ontologies in RDF+OWL, has a single name space (WebKB-2 too). Domain names, HTML and the Meta Content Framework Using XML [www-MCF/XML, 1997] are case insensitive. The main reason for case sensitivity seems to be that the performance cost of mono-casing in Unicode is important [www-XML-case-sensitive, 2004]. By default, for search purposes, WebKB-2 first converts the query words or category identifiers in lowercase and ignore the '_' or '-' characters as well as the final 's'; if this leads to lexical ambiguities (i.e., several possible identifiers), the given spelling and capitalization are used. In an expression or statement in FE or FCG it is possible to add a final 's' to a name or an unprefixed category identifier when it is used with a numerical quantifier (as in `75% of cat*s*'), as long as there is no lexical ambiguity.

2.3.1.2. Using singular nouns or singular nominal expressions for category identifiers. This is a convention in the Meta Content Framework Using XML and there are many reasons why a singular nominal expression – or a gerund (the '-ing' form of an English verb) for referring to an action – should be used for naming a type of concept or relation. For example, the names "defining" (for the action) and "definition" (for the output of the action or for the relation) should be used instead of names such as "hasDefinition", "definitionOf", "define", "defined" and "definitions".

- This eases the reading of relations according to the graph-oriented reading convention where "X R: Y" reads "X has for R Y" or "Y is the R of X" (this convention is given in the documentations of many frame-based or graph-based languages, e.g., in some of the documentations of RDF and Conceptual Graphs). This also makes the use of the identifiers far less odd in natural languages or controlled natural languages.
- Respecting the graph-oriented reading convention permits the relation types to have similarly ordered signatures: first the type of the source (a type of collection if there are many sources, inputs or parameters), then the type of the main destination, and then possibly other types for specifying information on the context, e.g., modalities. This eases the ordering and reading of the relation type hierarchy and hence its understanding as well as the correct use of the relation types. When a relation type not respecting this convention is integrated to the MSO, e.g., a relation type from Ontolingua (an ontology that does not respect the graph-oriented reading convention), it is connected by a relation of type pm#equivalent_object or pm#inverse to a relation type that respects the convention. Not making such an harmonization makes the reading or understanding of the ontology difficult. The SUMO is an example of ontology where no harmonization in made. The SUMO follows the Ontolingua convention except in at least two cases: i) for functional binary relation types (alias, "unary functions"), and ii) for types of basic relations classically used for representing natural language sentences, e.g., case relations.
- This reduces the use of *verbs* for naming relation types and hence the use of relation nodes instead of concept nodes for representing processes. It is better to use concept nodes (with a nominal expression for their types) for representing *processes* because, as opposed to relation nodes, concept nodes can be quantified in various ways (as in `at least 2 hit' or `75% of hitting') and connected via many common relations to other concepts, e.g., relations to represent the agent, object, instrument, goal, time and place of the process. This is easier than declaring relation types with names such as "hits", "hits_with_instrument", "hits_with_instrument_for_a_reason" and defining them with respect to a concept type with name "hitting" to permit the statements using these relations to be automatically compared. This also avoids 'duplications' and permits other users to "correct" a particular basic relation (e.g., the one used for representing the instrument) rather than the whole process. Finally, many users would not actually take the time to define relation types such as "hits" or "hits_with_instrument" with respect to a concept type "hitting". For similar reasons, using names such as "define", "defined" or "definedIn1990" for types of processes should be avoided in favor of the nouns "definition" or "defining" ("two definings" is correct in English, even though it sounds odd). Using this last name is better because it clearly refers to a process and this leaves the possibility to use "definition" for the identifier referring to the output of the "defining" process (in the MSO, wn#definition is the output of wn#defining) or for the identifier of a "definition" relation (e.g., pm#definition in the MSO).

```
wn#defining .(pm#input: *x, pm#output: *y)  < wn#indexing,
  input:  (pm#thing *x  pm#definition: *y),
  output:  wn#description *y;
```

- This reduces the use of adjectives as category identifiers. Indeed, the use of an adjective is not warranted when the category does not refer to an attribute or measure, as with sumo#abstract which, given its meaning, should actually have been named sumo#abstract_thing (or sumo#abstract_entity since `sumo#entity = pm#thing'). However, (proto-)ontologies such as WordNet include categories with adjectives, verbs or adverbs as names, and it is interesting to integrate such categories, for example to permit a more direct representation of natural language sentences. This is why in WebKB-2 the following statements are equivalent.

```
[an abstract process];
[a process, kind: abstract];
[a process, kind: abstract_thing];  //there is a process that is an abstract_thing
[an abstract_thing, kind: process]; //there is an abstract_thing that is a process
```

Using an adjective for naming a subtype of pm#attribute_or_quality_or_measure (e.g., wn#red) is not a problem: an adjective is a way to express a measure (Subsection 3.1.2 gives the rationale for the classification of measures, attributes and qualities; they include minimizing the number of ontology specific relation types to use and hence creating simpler and more normalized knowledge representations). Thus, in a conjunction of types such as the one in the previous example, WebKB-2 interprets differently the use of a subtype of pm#attribute_or_quality_or_measure. Since pm#attribute_or_quality_or_measure is a subtype of pm#thing_that_can_be_seen_as_a_relation (see Paragraph 2.1.1.14), FCG statements such as the following ones are considered equivalent.

```
[a wn#red wn#mat];
[a wn#mat, attribute_or_quality_or_measure: some wn#red];
[a wn#mat, wn#color: some wn#red];   //since wn#color is also a supertype of wn#red
[a wn#mat, wn#red___redness: some wn#red];
```

The end of Subsection 3.2.4 lists the ways the WordNet categories having verbs, adverbs and adjectives as names are planned to be integrated in the ontology of WebKB-2.

A category with an adjective as name should be defined with respect to a category with a nominal expression as name. To allow this, FL, FCG and FE have some predefined keywords: "important", "small", "big", "great", "good" and "bad". For example:

```
wn#long :<=> important wn#length;  //rationale in the subsections 3.1.2 and 3.2.4:
//  wn#long < (wn#length < pm#attribute_or_quality_or_measure)
//  wn#length is neither a characteristic (a quality) nor a value (a quale) but
//    can be specialized by categories of each kind
```

Since WordNet relates each of its categories with adjectives as names to its nominal counterpart, definitions such as the above one can be generated. Then, the following statements are equivalent.

```
[Tom, owner of: a wn#long wn#car];
[Tom, owner of: (a wn#car, wn#length: an important wn#length)];
[Tom, owner of: (a wn#car, attribute_or_quality_or_measure: an important wn#length)];
```

However, for precision purposes, adjectives should still be avoided. For example:

```
[Tom, owner of: (a wn#car, length: 7 wn#meter)];
```

- This reduces the use of plural nouns and hence reduces the introductions of relations from/to collections as well as the use of collections with a collective or cumulative interpretation. This is a good thing because such collections are not easy to handle and most common knowledge inferencing tools – such as those based on OWL – do not much exploit them. For example, the next two statements are not comparable by simple graph-matching (projection): none is a specialization of the other.

```
[Tom, parents: the set {John,Mary} ];  //or:  [Tom, parents: parents_of_Tom];
[Tom, parent: a wn#man];
```

On the other hand, using collections in their distributive interpretation is not a problem: they are just shortcuts to avoid repeating relations. For example, the FCG statement, [Tom, parent: {John,Mary}] specializes [Tom, parent: a wn#man] if `John' or `Mary' is instance of wn#man. The possibility of using numerical quantifiers is important too since numerical quantifiers can be easily compared and they reduce the need of declaring concept types representing collections.

2.3.1.3. Naming of second order types. To help people distinguish second order types – or types of greater order – from first order types, and hence eases their understanding of the ontology organization, it is interesting to end the identifiers of second order types by

- "_class" or "_type" if they have concept types as instances,
- "_relation_type" (or "Property" as is the case in RDFS/OWL) if they have relation types as instance and, more precisely,
- "_function_type" (or "FunctionalProperty") if they have function types as instances.

This would for example eases the understanding of SUMO and CYC.

2.3.1.4. Informal definitions and annotations of categories. Like all other definition bodies or statements, informal definitions and category annotations should preferably be undecomposable (to that end, several different definitions or annotations can be written) and should be either be expressions (e.g., nominal expressions or sentences beginning by a relational expression such as "example: " or "e.g., ") or stand-alone statements. These last ones should be understandable without having to know that they are related to a particular category, and hence, they should not use expressions such as "this category". This permits anyone to relate these expressions or statements to other ones (e.g., a formal version or a specialization) or to argue for or against their relations to the category they are associated to. Informal definitions and annotations of types may refer to the category itself or to its instances: it is not possible or handy to either always speak of the instances or always speak of the type. Finally, the more formal the definitions and annotations, the better.

It is better to use a long explicit category identifier than to use an informal definition to compensate for an imprecise short identifier. Indeed, statements should be understandable without having to access the definition of the terms they include (this is a loss of time and may require losing sight of other important information thereby making mental synthesis a more difficult exercise). Hence, for example, declaring the identifier pm/km#sharing_of_a_knowledge_base_that_is_physically_distributed_or_not is at least as good as declaring:

```
KB_sharing  definition: "sharing of a knowledge base, physically distributed or not";
```

Finally, the "generation of categories for organization purposes" described in Paragraph 2.1.1.25 participates to lexical and structural normalization.

2.3.2. Structural or Semantic Normalization

```
following_a_structural_normalization_rule_when_representing_knowledge
  >  (following_a_normalization_rule_for_the_signatures_of_relation_types
       >  (following_the_graph-oriented_reading_convention_for_the_signatures_of_binary\
_relation_types
            subprocess:  using_singular_nouns_or_nominal_expressions_for_category_identifiers
                         avoiding_the_declaration_or_use_of_non-binary_relation_types)
     (maximizing_precision_and_organization_while_minimizing_the_use_of_expressive_constructs
       subprocess:  (maximizing_precision_and_organization
                     subprocess: relating_formal_and_informal_categories
                                 connecting_or_adding_to_large_ontologies
                                 relating_to_and_organizing_informal_terms_and_statements)
                    minimizing_the_use_of_expressive_constructs_but_not_high-level_constructs
                    avoiding_the_declaration_or_use_of_non-binary_relation_types
                    defining_non-binary_relation_types_with_respect_to_binary_relation_types
                    (using_subtype_relations_instead_of_instance_relations
                      >  using_first-order_types_instead_of_second-order_types
                         using_first-order_types_instead_of_individuals)
                    (keeping_the_relation_type_hierarchy_small_and_organized
                      subprocess:  avoiding_the_representation_of_processes_via_relation_types
                    ));
```

2.3.2.1. Following the graph-oriented reading convention for the signatures of binary relation types. This was argued for in the first two points for Subsection 2.3.1.2 ("using singular nouns or singular nominal expressions for category identifiers"; this last process can be seen as a subprocess of this one).

2.3.2.2. Avoiding the declaration or use of non binary relation types. Many KRLs and KR systems do not support non-binary relations and there is no common reading convention for non-binary relations. However, many of these relations can be replaced with binary relations. For example, instead of being declared as ternary relations, relation types with names such as "location_between" and "sum" can be declared as binary relations using a set.

```
pm#location_between .(pm#spatial_object, {pm#spatial_object});
pm#sum_of_real_numbers .({sumo#real_number} -> sumo#real_number);
```

Relations with such types can be read with the graph-oriented reading convention. Here are some equivalent statements.

```
[the set {1,2,3,4}, sum: 10];  [sum({1,2,3,4}) = 10];        //in FCG
{1,2,3,4}_[cuml]    sum: 10;    sum({1,2,3,4})_[cuml] = 10;  //in FL
```

This use of sets may also be a good alternative to variable-arity relations since the use of sets may be more intuitive and there are probably more KRLs able to support sets than variable-arity relations.

2.3.2.3. Defining non binary relation types with respect to binary relation types. Most KRLs with a predicate-logic oriented notation allow n-ary relations but not contexts, e.g., CLIF. KRLs with a frame/graph oriented notation more commonly allow contexts and only binary relations. Whenever possible, if n-ary relation types are declared (e.g., to use temporal and modal arguments), they should be defined using binary relations. Examples in FCG:

```
[definition_3 .(thing *i, description *o, time_measure *t), supertype:  pm#definition,
   :<=> [a wn#definition, input: *i,  output: *o, time_measure: *t] ];
[definition_3 .(input: *i, output: *o, time_measure *t), supertype:  pm#definition,
   :<=> [ [a wn#definition, input: *i,  output: *o], time_measure: *t] ];
```

In WebKB-2, given the facility described in Paragraph 2.1.1.21 ("Contextual relations from/to (descriptions of) processes"), these two definitions are seen as equivalent. This facility can be seen as a kind of structural normalization.

These two definitions also state that pm#definition_3 is a subtype of the binary relation type pm#definition. It seems reasonable to allow relation types to be declared as subtypes of relation types having less arguments. Doing so or using variable arity signatures permits to organize relation types into a single specialization hierarchy. In the MSO, only the variable arity signature based approach is currently used. Here are examples in FL (reminder: "*" refers to an unknown number of arguments of unknown types, while "?" refers to a unique argument of unknown type).

```
relation .(*)
  >  (relation_from_collection .(collection, *)
        >  (member .(collection, *)
              >  kif#first .(kif#list -> ?)
                 kif#nthrest .(kif#list, kif#natural -> kif#list)
           ) );
```

2.3.2.4. Maximizing precision and organization. The more precise and organized the knowledge, the more understandable, retrievable, (correctly) re-usable, scalable and easy to validate it is. Hence, at least until a certain level of formalization is not reached, the more it prevents people to enter redundancies and inconsistencies. (When a certain level of formalization or ontological detail is reached, this opens or reveals (onto-)logical cans of worms and increases the number of alternative (onto-)logical ways to represent objects). Here are some of the things that information providers should do when entering a new category or statement:

- use the most precise quantifiers (e.g., "1" instead of "at least 1" when this is relevant);
- use the most precise terms,
- precise the direct and reverse cardinalities;
- use numbers rather than qualifiers; and

- connect each object (category or statement) they represent to its related objects (those already existing in the KB and the important ones not yet represented), at least via the most important or common relations:
    - transitive relations, especially (extended) specialization/generalization relations and mereological relations (to specify parts, containers, ...); as noted in Paragraph 2.1.1.25 ("Generation of categories for organization purposes"), for more extended specializations to exist, categories such as pm#enforcement_with_agent_a_government should be manually created if they are not automatically generated and should then be used in statements;
    - exclusion/correction relations (especially via subtype partitions);
    - instance/type relations;
    - basic relations from/to processes;
    - contextualizing relations (spatial, temporal, modal, ...);
    - argumentation relations (objection, argument, example, ...) but only after checking that what is to be represented cannot be (at least partially) represented in a more precise and organized way, typically via specialization or subprocess relations between processes (this is often possible; then, if still needed, argumentation relations only have to be set in the context node of the specialization or subprocess relations); similarly, to normalize the argumentation structures, whenever possible, instead of relations of type pm#objection or pm#correction, relations of type pm#restrictive_correction or pm#corrective_generalization should be used with relations of type pm#argument in their context; this improves the precision and organization of the argumentation structures and these corrections are more likely to be popular.

Before representing objects, the information providers should i) check which categories share a name with these objects, and ii) check which relation types are associated to these kinds of objects (e.g., via definitions or signatures). Whenever possible, objects should not be used before being declared, i.e., be given at least one type or supertype. Except for the use of certain relation types, this is always possible. This permits early semantic checks and participates to normalize the presentation of the code (and thus, to a certain extent, its content).
In WebKB-2, the shared KB edition protocols enforce a (very) minimal organization. They should probably be completed by "certificates of connectivity" delivered by the system for each newly entered object based on the existence or not of the above listed kinds of relations from/to the object. Then, if needed, certificates of "semantic well-formedness" or "semantic normalization" (e.g., in the spirit of Codd's database normalization levels) could also be envisaged for each statement and hence, via statistical measures, for the whole KB.
This paragraph and the next ones in this subsection describe advisable subprocesses of `maximizing_precision_and_organization_while_minimizing_the_use_of_expressive_constructs'.
Following, enabling or enforcing the processes described in the previous sections (supporting_knowledge_sharing_between_KBs, supporting_the_collaborative_building_of_the_KB_specified_as_output and supporting_the_valuation_of_knowledge_or_knowledge_authors) are complementary advisable subprocesses. Hopefully, in the future, the lexical, syntactic, structural and semantic rules associated to all these processes will be refined to reduce the entering of formal/informal statements that are over-general, false, redundant or poorly structured and inter-related. The next paragraphs focus on structural and semantic rules for (semi-)formal knowledge.

2.3.2.5. Minimizing the use of expressive constructs but maximizing the use of high-level constructs. The more precise and organized the knowledge, the more understandable, retrievable and re-usable it is. However, the more it uses expressive constructs (e.g., second order statements), the less inference engines can use it and the less the inferences can be efficient, complete or consistent. Hence, for example, using exclusive relations or cardinalities instead of using a general negation on a whole statement is a good thing. ***However, for general knowledge representation and sharing purposes, there is no point to bias or restrict the precision, organization and readability of knowledge by using low expressiveness constructs*** (e.g., those of OWL-Lite) when entering it since i) such knowledge is not dedicated to particular kinds of application, and ii) knowledge using expressive constructs can be automatically translated to less precise/correct knowledge using less expressive constructs, or these expressive constructs can be directly only partially interpreted by inference engines (as is for example the case with the ext-gen or ext-spec operators of WebKB-2 which are efficient and give "relevant" results – but not always complete results –

whichever the expressiveness of the knowledge is). The next four points lists some characteristics of RDF+OWL related to this issue since RDF+OWL is the de-facto standard *general* model.

- As previously noted, using (or imposing the use of) a relation type pm#supertype_or_equal (owl#sub_class_of) whereas the relation type pm#supertype could be used, restricts possible inferencing and sooner or later leads to undetected redundancies and inconsistencies.
- OWL has various "profiles", i.e., versions of various expressive power: OWL(1)-Lite, OWL(1)-DL OWL(1)-Full, OWL(2)-EL, OWL(2)-QL and OWL(2)-RL. Even RDF+OWL-Full is not particularly expressive (no genuine meta-statements, limited interpretations of collections, limited kinds of quantification, no distinction between definition and universal quantification, ...). However, RDF+OWL will continue to be progressively extended. Furthermore, more expressive approaches than RDF+OWL but still closely related to it could also be adopted by the W3C in the future to answer the demand for more expressiveness [Patel-Schneider, 2005].
- RDF+OWL is also "low-level" in the sense that it often does not offer constructs to express things in concise and general ways. This is particularly clear with its representation of cardinalities which leads to long and unintuitive formulations, and hence to difficult to understand, check and compare knowledge. Here is an example of equivalent sentences in English (En), FL, KIF and RODX (RDF+OWL-DL/XML). Many other examples are given in Chapter 4.

```
En:   Any human body has at most 2 arms. Any arm belongs to at most 1 body.
FL:   human_body  part:  wn#arm __[?->0..2, 0..1<-?];
KIF:  (forall ((?b pm#human_body)) (atMostN 2 '?a wn#arm (pm#part ?b '?a)))
      (forall ((?a wn#arm)) (atMostN 1 '?b pm#human_body (pm#part '?b ?a)))
RODX: <rdf:Property rdf:ID="ArmPart"><rdfs:subPropertyOf rdf:resource="&pm;part"/>
        <owl:inverseOf rdf:ID="ArmPartOf"/>
        <rdfs:range rdf:resource="&wn;Arm"/> </rdf:Property>
      <owl:Class rdf:about="&pm;HumanBody"><rdfs:subClassOf>
        <owl:Restriction><owl:onProperty rdf:resource="#ArmPart"/>
          <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">2
          </owl:maxCardinality></owl:Restriction> </rdfs:subClassOf></owl:Class>
      <owl:Class rdf:about="&wn;Arm"><rdfs:subClassOf>
        <owl:Restriction><owl:onProperty rdf:resource="#ArmPartOf"/>
          <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1
          </owl:maxCardinality></owl:Restriction> </rdfs:subClassOf></owl:Class>
With (in KIF):
(defrelation atMostN (?num ?var ?type ?predicate) :=
  (exists ((?s set)(?n)) (and (size ?s ?n) (=< ?n ?num)
    (truth ^(forall (,?var) (=> (member ,?var ,?s)
                                (and (,?type ,?var) ,?predicate)))))))
```

- Other representations not using RODX but still directly using RDF+OWL-DL, for example via Notation 3 (N3), would also be rather long and unintuitive.

Using high level constructs (short expressive constructs such as numerical quantifiers, expressed via quantifiers/relations such as `atMostN' or via syntactic sugar such as in FL) ***is needed.*** Since writing constructs such as `atMostN' is not easy and since their definitions are unlikely to be exploited by most current engines, ***the ontology of a would-be general language such as RDF+OWL-Full should include such constructs***. Then, inference engines may be adapted to (partially) interpret them or not. Like FE and FCG, the Lisp-based KRL of the Knowledge Machine [www-KM, 2006] keywords such as "a", "the" and "every" for its existential and universal quantifiers. FL, FE and FCG will continue to be completed to include more high-level constructs for representing expressions common in natural languages such as "1 by 1" or "3 by 3".

2.3.2.6. Using first-order types instead of second-order types. Using second-order types permits to associate properties to a type without associating them to its subtypes too. For example, the OntoClean methodology [Guarino & Welty, 2002] encourages the implicit or explicit use of class properties (hence second-order types) such as rigidity and unity in order to reduce mis-uses of the subtype relation such as for example i) subtyping wn#water by wn#ocean, ii) subtyping a type representing a role by a type not representing a role, and iii) using subtype relations instead of partOf or instance relations. These second-order type of DOLCE are shown by Table 3.1.10.2.
However, using second-order types when first-order types would be sufficient is not a good idea. For example, declaring a second-order type named "product_type" and relating types of products to it via pm#kind relations, instead of declaring a first-order type named "product" and relating types of products to it using pm#supertype relations, is a kind of problem that occurs in general ontologies such as SUMO (e.g., see Table 3.1.10.2), CYC and TAP. Indeed, using supertype relations between first-order types whenever it is possible

- reduces the 'duplications' of these types as second-order types (if only because these first-order types are useful and will sooner or later be created),
- reduces the 'duplication' of supertype relations as pm#kind relation, and
- permits a more direct (and easier to build, navigate, understand and automatically exploit for knowledge checking or retrieval) connectivity between the first-order types (since pm#supertype is transitive).

OWL proposes a second-order type owl#transitiveProperty since transitivity is a property of a relation type which is *not* necessarily shared by its subtypes. However, since it is often shared, to ease the organization of transitive relation types and avoid the tedious setting of a pm#kind relation to owl#transitiveProperty from each of them, the MSO proposes the following two types.

- The type pm#relation_instance_of_transitiveProperty_unless_directly_overrided is instance of owl#transitiveProperty and its subtypes are considered transitive except for those negating this property, as in the following example:

```
rX supertype: relation_instance_of_transitiveProperty_unless_directly_overrided __[.->0];
```

- The type pm#type_instance_of_a_certain_second_order_type_unless_directly_overrided which was introduced in Paragraph 2.1.1.19. For export purposes, if needed, the instances of this type can be filtered out and pm#kind relations to the relevant second-order type can be automatically added from the subtypes of these instances.

A more debatable case is the representation of "species" as second order types which have various types of plants or animals as instances. It seems better to represent them as subtypes (or instances) of pm#collection which have various types of plants or animals as members. This is the case in WordNet and hence in the MSO.

```
wn#class_Mammalia___Mammalia
  member:  wn#young_mammal  wn#mammal
           wn#subclass_Prototheria  wn#subclass_Pantotheria  wn#subclass_Metatheria
           wn#subclass_Eutheria   wn#Ungulata  wn#Unguiculata,
  member of:  wn#subphylum_Vertebrata,
  supertype:  (wn#class <  (wn#taxonomic_group  <  wn#biological_group));
```

2.3.2.7. Using first-order types instead of individuals. Individuals (i.e., instances of first-order types) may also be over-used. For example, it may be tempting to represent a certain doctrine, language, program or day of the week as an individual, but then what about their variants and their occurrences? For example, "Monday" has a potentially infinite number of occurrences, and so has "Whitmonday" (the day after Whitsunday). Considering the currently existing tools, the simplest solution (for people and for automatic exploitation) is to represent "Whitmonday" as a subtype of "Monday" and its occurrences as individuals (anonymous or not). Similarly, an alphabetic character (seen as a symbol) and the content of a book may also have (existing or potential) variants; for example, the Bible has many versions in many languages and the character 'A' has "versions" too (uppercase, lowercase, ...). There are many ways to view, categorize and relate such "versions" but, as for "Whitmonday", using subtype relations seems the simplest way, including for inferencing purposes. This is the option that was chosen when integrating WordNet into the MSO. Things that intuitively cannot have versions, e.g., persons or cities, were declared as individuals in the MSO even though categories such as pm#Paris_as_the_capital_of_France_from_1990_to_2000 can be declared. To relate such an identifier to wn#Paris, an extended specialization relation (e.g., with type pm#term_specialization) can be used. However, it seems preferable to use contexts on statements than to declare such identifiers.

2.3.2.8. Keeping the relation type hierarchy small and organized. The more organized the relation types, the more understandable they are and the more comparable the statements using them are. The smaller the hierarchy of relations, the less 'duplications' with respect to the concept type hierarchy it has and the more it contains only primitive or common relation types, and hence i) the easiest the relation type hierarchy is to understand and use, ii) the easiest it is to find and use these relation types, and iii) the more comparable the statements are. To keep the relation hierarchy organized without drawbacks for the precision of knowledge representations, it is necessary to i) allow subtype relations between types of relations of different arities, as above specified, and ii) organize relation types according to their arguments (other methods can also be used in addition). To keep the relation hierarchy small, certain kinds of concept types should be allowed to be used in relations nodes and hence should be allowed to have associated signatures. Subsection 4.2.13 gives definitions in KIF for relations permitting to generate and organize relation types based on the signatures associated to concept types and the organization of these types. These KIF definitions may be used for exporting purposes or for defining this relation type generation process.

2.3.2.9. Avoiding the representation of processes via relation types. This is a subprocess of the one described in the previous paragraph. Some rationales and techniques have been cited in the previous subsection and the previous paragraph. Relatively few basic relation types are required for most general knowledge representation, e.g., for the representation of natural language. When the content of courses taught at Griffith University were represented in the MSO, most of the used relation types were (without the "pm#" prefix): subtype, instance, specialization, part (physical_part or subprocess), technique, tool, definition, annotation, use, purpose, rationale, role, origin, example, advantage, disadvantage, argument, objection, requirement, agent, object, input, output, parameter, attribute, characteristic, support and url. (This list is ordered topically, not by frequency of occurrence. The representations in this document also use a small number of relation types. This eases the automatic comparison of these representations and hence their retrieval or exploitation.

2.3.2.10. Connecting/adding to large ontologies. The biggest the ontology, the more complete, precise and organized it has to be for being manageable, and hence the more it guides or spares knowledge entering, and the more users (e.g., information providers) it is likely to have. For general knowledge sharing and representation purposes, specializing (or connecting to) a large general ontology not only saves efforts and improve re-usability and scalability but also eases the comparison and retrieval of objects specializing (or connecting to) this ontology. Here are some quick description of some large general ontologies.

- DBpedia [www-DBpedia 2009] is currently the largest general ontology but, apart from its core lexical ontology (YAGO, an extension of WordNet with certain individuals from Wikipedia), it is essentially a collection of loosely aligned databases of individuals (instances of first-order types). Unlike in WebKB-2, the reused WordNet has not been corrected, completed and given intuitive identifiers to transform it into a genuine lexical ontology. Finally, DBpedia cannot be collaboratively corrected and extended by Web users, it is mainly generated from Wikipedia. Thus, the categories of DBpedia are mainly only meant to be referred to in the spirit of the Tim Berners-Lee's Linked Data philosophy [www-Linked-Data, 2009], for example in the context of the Linking Open Data project [www-LDOP, 2009].
- CYC (or its public subset, OpenCYC) is a large general ontology that is well organized although not in an intuitive manner nor a convenient one for general knowledge representation and sharing, for example because of i) its over-categorization of types for description content/medium/containers, ii) its over-use of 2nd-order types, and iii) its lack of use of basic relations such as case relations. CYC also cannot be collaboratively corrected and extended by Web users.
- The problems of CYC are also problems of the SUMO, an ontology which merges or maps several general ontologies. The merging is not loss-less. The merging and the mapping are done in less precise, organized and principled ways than in the MSO. For example, the mapping to WordNet categories uses "related-to" relations from categories of the core of SUMO, not identity or specialization relations.
- The main site of WebKB-2 permits Web users to refer or correct and extend a general ontology which is a bit smaller than SUMO and OpenCYC but is adapted to *general* knowledge representation and sharing. This main site does not fulfill the integrating_all_published_information_specified_as_parameter specification but fulfills

the supporting_the_collaborative_building_of_the_KB_specified_as_output specification. Finally, by connecting categories from other ontologies, the MSO complements them and can be seen as a useful resource for the Linking Open Data project.

2.3.2.11. Relating to – and organizing – informal terms and statements. Relating formal objects to informal ones improves the retrieval of these formal and informal objects and eases their understanding. An informal object may have less meanings or interpretations than another one, and its meanings may be less general. Hence as with formal objects, it is interesting to organize informal objects via specialization relations (manually for informal terms, and manually or automatically for informal or semi-formal statements). To that end, the MSO proposes the pm#extended_specialization relation type and WebKB-2 exploits it.

2.3.3. Application for Correcting some Examples or Advices from W3C People

Even if the two previous subsections include recommendations that may appear obvious to many knowledge engineers, they were not meant to include simple advices that would be obvious to all knowledge engineers. To find simple advices, a well-referenced document titled "Ontology Development 101: A Guide to Creating Your First Ontology" is [Noy & McGuinness, 2000]. However, this guide seems more oriented towards creating an ontology for a particular application than towards knowledge sharing purposes. Indeed, it contains statements that have been argued against in the two previous subsections. Here are some examples.

- "Deciding whether a particular concept is a class in an ontology or an individual instance depends on what the potential applications of the ontology are.". It is true that using individuals may shorten the code for certain applications but, for knowledge sharing purposes, using types is much better if the referred objects can have specializations (e.g., if it has occurrences or versions).
- "You do not need to specialize (or generalize) more than you need for your application (at most one extra level each way).". For knowledge retrieval, understanding and sharing purposes, each object should be semantically connected (at least via the main kinds of relations cited in Paragraph 2.3.2.4) to all related objects in the current ontology and, if this ontology is not a large one, to the related objects in a large ontology.
- "A class Wine actually represents all wines. Therefore, it could be more natural for some designers to call the class Wines rather than Wine. No alternative is better or worse than the other (although singular for class names is used more often in practice)". Paragraph 2.3.1.2 lists arguments against this.
- The ontology used for illustration purposes in this guide includes relation types with names such as "bestWineries", "produces" and the not-very-explicit "io".

Here are other examples showing that the recommendations of the two previous subsections are unfortunately not obvious to every knowledge engineer.

- The first few versions of the RDF documentation stated that, for some uses, representing "the price of that pencil is 75" was better than representing "the price of that pencil is 75 US cents".
- When integrating WordNet into the MSO, the string "_USA" had to be added to category names such as "North" (hence now "North_USA"), and the string "in USA" had to be added within the annotations of some categories with names such as "Department_of_Education".
- Representations such as "any bird flies" are common.

The "OWL Web Ontology Language Guide" [Smith, Welty & McGuinness, 2004] uses a small ontology as an example translation in FL (see Table 2.3.3.1 below). This ontology is extremely restricting (it only permits to state sub_area relations), semantically shallow and hard to extend or re-use. Table 2.3.3.2 shows a more scalable approach: i) it uses a process based modeling (the given example illustrates what this permits), ii) it uses types from the MSO and thus only wine#Chateau_Margaux has to be declared, and iii) even when the types already declared in the MSO

are counted, this new version is not much longer than the original one. In this original version, dividing the sub_area relation into three subtypes serves no purpose: using one "spatial_part" relation is more general and no more ambiguous. Finally, in the original version, several inverse relations have to be declared because the RDF+OWL/XML syntax does not provide any way to change the direction of a relation node!

Table 2.3.3.1. The W3C example of "Wine ontology" directly translated in FL

```
wine#ProductionArea  >  wine#Country  wine#Region  wine#Vineyard;

wine#hasSubArea (?,?)   kind: owl#TransitiveProperty,  inverse:  wine#subAreaOf,
  >  wine#hasRegion  wine#hasSubRegion  wine#hasVineyard;

    wine#hasRegion (wine#Country <- wine#Region)   inverse:  wine#regionOf;
    wine#hasSubRegion (wine#Region<- wine#Region)  inverse:  wine#subRegionOf;
    wine#hasVineyard (wine#Region<-wine#Vineyard)  inverse:  wine#vineyardRegion;

//Examples of uses:
wine#Country  instance:  wine#France  wine#Italy;
wine#Region   instance:  wine#Aquitaine  wine#Roussillon;
wine#Vineyard instance:  (wine#Chateau_Margaux  wine#vineyardRegion: wine#Aquitaine);
```

Table 2.3.3.2. Correction of the previous example for knowledge sharing and retrieval purposes

```
wn#vineyard  instance: wine#Chateau_Margaux;
/* Concept types and relations already declared in the MSO:
   wn#process  > wn#production;
   pm#spatial_object  >  (wn#location  >  wn#country  wn#region  wn#vineyard);
   pm#physical_entity  > wn#wine;
   pm#attribute_or_quality_or_measure  >  wn#volume;
   pm#object .(pm#situation, ?);   //or:  wine#object .(wn#process, wn#wine);
   pm#place (pm#situation -> pm#spatial_object);
   pm#time (pm#situation, pm#time_measure)
   pm#spatial_part .(pm#spatial_object, pm#spatial_object)  kind: owl#transitive_property;
*/
//Example of use in FCG:
[wine#Chateau_Margaux,  pm#spatial_part of: wn#Aquitaine,
   pm#place of:  (a wn#production,  pm#time: 2003,  pm#object: some wn#wine,
                                    wn#volume: 20000 wn#liter)];
```

2.3.4. Normalization of Input Files

Uses of input files. Creating an ontology, especially a large one, has analogies with programming: this leads to i) modularize the code in various input files, ii) make comments, iii) submit each file to the parser and correct mistakes, and iv) correct structural and ontological decisions which affect many objects. This last point requires modifications of all the affected objects in one transaction, i.e., by re-submitting one or several input files. Thus, like (good) procedural/declarative programs, (good) ontologies cannot (efficiently) be created via a line command interface or a graphical interface that is not multi-views, does not include a textual editor or does not permit to build several input files. In other words, directly updating a KB via a line command editor or a "classic" ontology editor does not permit to design a good KB or to design it solely using the line command editor. Here, the expression "classic ontology editors" refers to

- editors using a window mainly composed of a side pane showing a small portion of the hierarchy of specializations between categories and a larger central pane showing *only* the direct relations from the category selected in the first pane, and more generally,
- editors that do not permit to view a (specialization/part/...) hierarchy of *many* objects according to various filtering and display criteria and, at the same time, some selected direct relations from these objects.

Such a view is indeed necessary to permit people to compare and hence understand objects and their connections, and then make good design/update choices. Most current ontology editors, including Protégé, are "classic ontology editors". One reason is that their simple design pleases (at least the) beginners in knowledge representation. The focus for the interfaces of WebKB (WebKB-1 and WebKB-2) was on

- the generation of Web-accessible search and comparison interfaces,
- the generation of cascading knowledge entering forms, and
- the use of input files (this is the knowledge entering method that the users are advised to use; if they wish to, they may use "classic ontology editors" to generate these input files).

Like most ontology editors and many ontology servers, WebKB-1 does not have a persistent KB: the user must specify which input files should be loaded into the KB before making queries. Thus, WebKB-1 can only be a "personal" KB server. WebKB-2 has a persistent shared KB. Hence, once an input file loads without problem (i.e., without errors being detected) and is considered as "complete" by its author, this one *commits* its content to the shared KB (i.e., unlike with the previous parsings of the file, the updates to the KB are not rolled back at the end of the parsing of the file) and then avoids to make modifications to this file. Indeed, to be able to re-submit an input file into the shared KB, its author must first (at least implicitly) ask WebKB-2 to delete the content of the committed previous version of the input file. This entails that the parts of this content that were reused by other users must be cloned: their creators become one of these other users. Then, ideally, when the new version of the input file is loaded, the objects that were cloned because of the deletion but that have not been modified in the new version should be un-cloned, i.e., their creator should be changed back. This is easy to describe but this is not yet implemented in WebKB-2. Hence, currently, to incorporate changes made to already committed input files the whole KB should be re-generated. In any case, once committed, an input file should essentially be considered by its author as a backup file for the content it contributes to the shared KB. This is useful since, like all tools, shared KB server have bugs, which means that the shared KB sooner or later gets corrupted and cannot be fully recovered without re-generating it from backup files.

Systematic knowledge modeling and organization of input files. WebKB permits to mix formal and informal statements in a same input file and hence permits not to separate (semi-)formal statements from their documentation or the document elements they index.
A recurrent problem when representing knowledge on a domain is how to organize the input file for grouping related pieces of information together in order to retrieve them efficiently, compare them and thus progressively represent them and make good modeling decisions. Unless one already knows the domain very well, ***choosing the right domain-dependent conceptual distinctions and corresponding (sub-)sections for grouping information or knowledge representations is difficult.*** Furthermore, as with any other structuring of a document via sections and subsections, such a choice often depends on the volume of information that one expect to finally have to represent in each (sub-)section. This is very difficult to know in advance and is not a scalable approach.

Fortunately, ***there exists a systematic, scalable and domain-independent approach*** since there exist domain-independent conceptual distinctions that are efficient guides for partitioning objects. This approach is simply to ***use certain partitions of the subtype hierarchy of the MSO*** (hence, certain levels of it; see Table 3.1.1.1 for the best high-level candidates) ***for structuring a document in sections and subsections***. Which levels are used depends on the volume of information in each section and their different kinds. When a section becomes too big for guiding search and comparison, it can be divided according to subtypes. Adding intermediate sections changes the numbering of subsequent or lower sections (if they are numbered) but, even if this is done manually, this is not an important extra work.

All the input files of the MSO are organized that way: no better way was found for managing the large amount of information that had to be represented. The highest level distinction that is always first used is "situations / entities / roles shared by situations and entities". Then, which distinctions are used depends on the domain. As an example, Table 2.3.4.1 shows the Table of Content of an input file for high-level information security related concepts. Like many other input files of the MSO, this one was incrementally created by systematically dispatching information from many (partially redundant) Wikipedia pages on the subject into sections representing the most important distinctions of the MSO.

Within older input files on domains such as Conceptual Graphs and Formal Conceptual Analysis, the top-level decomposition was less systematic since the titles of the top-level sections were similar to the following ones (in this list, the parenthesis are used for giving examples of subsections): "Events (Conferences, ...)", "Problem-solving processes (tasks, methodologies, ...)", "Description supports (structures, languages, ontologies, ...)", "Description containers (publications, mailing lists, ...)", "Instruments (inference engines, editors, ...)" and "Agents (organizations, teams, ...)".

For files representing the content of certain courses at Griffith University, only a restricted decomposition was necessary. For example, the titles of the top-level sections for the content of a Workflow Management course were: "Process ('Modeled or performed process', 'Process/method to model business processes')", "Variable ('Methodology-related variable')", "Data structure ('Case description', 'Other structure')", "Instrument (tool)" and "Agent (organizations)".

Table 2.3.4.1. The Table of Content of a classification for information security related concepts in the MSO

```
1.  General roles  -- (in-)security related roles playable by entities as well as
                      situations;  threats (errors, failures, faults) are specialized
                      in Section 3 of the Resist ontology

2.  Entities  -- (in-)security related things that are neither states nor processes
   2.1.  Attributes  (criteria)  -- availability, integrity, confidentiality, ...
   2.2.  Descriptions and description supports  -- statements, documents, languages, ...
      2.2.1.  Regulations
      2.2.2.  Policies  -- computer security policies/models, insurance policies, ...
      2.2.3.  Models and principles or strategies
      2.2.4.  Descriptions of controls or measures
      2.2.5.  Certifications (digital signatures, ...) and results of evaluations
      2.2.6.  Skill qualifications  (diplomas, ...)
      2.2.7.  Policy languages
   2.3.  Agents
      2.3.1.  Organizations  -- professional organizations, standard sources, ...
      2.3.2.  Individuals  -- security professionals, hackers, ...
   2.4.  Instruments (devices, systems, ...)
      2.4.1.  Hardwares  -- security evaluated/focused/threatening hardwares
      2.4.2.  Softwares  -- security evaluated/focused/threatening softwares

3.  Situations  -- (in-)security related states or processes
   3.1.  States  -- situations that are not processes, e.g., safety and insecurity
   3.2.  Security threatening processes  -- attacking, eavesdropping, ...
           -- The remaining sections are about "security supporting" processes
   3.3.  Managing  -- economic/sanitary/social/information risk management/engineering
   3.4.  Evaluating  -- security classification/evaluation
   3.5.  Designing
      3.5.1.  Applying design principles
      3.5.2.  Using cryptographic techniques
      3.5.3.  Supporting authentication and access controls
   3.6.  Satisfying criteria  -- supporting security needs
      3.6.1.  Supporting confidentiality
      3.6.2.  Supporting integrity
      3.6.3.  Supporting accessibility and interoperability
      3.6.4.  Supporting imputability
   3.7.  Fault tolerance/forecasting/prevention/removal  -- Section 4 of the Resist ontology
```

Normalization of input files. Since the above described approach can be seen as a way to normalize input files, in the same way that a certificate of "semantic well-formedness" could be delivered about any statement and thus about a whole KB, a certificate of "semantic-based structure well-formedness" could be delivered about an input file. This would require using a category identifier as the title of a section or using another way to make explicit the association between a section and the category it refers to. In the input files of the MSO, this association is only implicit. Some criteria for "semantic-based structure well-formedness" could for example be:

- "all groups of statements are made explicit, via informal means (sections) or formal means",
- "all relations between groups of statements (e.g., relations with types such as `specialization' and `objection') are made explicit",
- "all groups of statements are inter-connected by transitive relations (typically specialization or mereological relations",

- "all formal terms are declared (i.e., typed or supertyped) before being used", and
- "all statements are (semi-)formal" (this can for example be the case with a document fully written in a (semi-)formal controlled language).

It is worth noting that the document formating rules that publishers generally impose, or the "document quality standards" that many organisms impose, are inconsistent with criteria for "semantic-based structure well-formedness". For example, Wikipedia's quality standards recommend to avoid using a list format for a text that can be presented using prose.

Use of the approach for organizing informal lists or hierarchies (topic hierarchies, FAQs, menus, etc.). Table 2.3.4.2 shows how the entries of the Google/ODP directory [www-ODP, 2009] can be structured using categories from the MSO (see the terms with a lowercase initial) in order to ease information retrieval.

Table 2.3.4.2. Adding some structure to the Google/ODP directory

```
thing
  situation
    activity_with_agent_a_person
      Business: Industries, Finance, Jobs,...
      Games: Board, Roleplaying, Video,...
      Shopping: Autos, Clothing, Gifts,...
      News: Media, Newspapers, Current Events,...
      Recreation: Food, Outdoors, Travel,...
      Society: Issues (see below), People, Religion,...
      Sports: Basketball, Football, Soccer,...
    activity_with_object_a_person
      Health: Alternative, Fitness, Medicine,...
  entity
    spatial_entity
      Reference: Education, Libraries, Maps,...
      Regional: Asia, Europe, North America,...
      physical_entity
        non-living-entity
          Arts: Movies, Music, Television,...
          Science & Technology: Biology, Psychology, Physics,...
          Computers: Hardware, Internet, Software,...
        living-entity
          person
            Kids and Teens: Computers, Entertainment, School,...
            Home: Consumers, Homeowners, Family,...
```

2.4. Knowledge Comparison and Knowledge-based Indexation and Retrieval

Here is a categorization of some processes (with some of their related objects) discussed in this section.

Table 2.4.1. Some concept types related to knowledge-based retrieval

```
knowledge-based_search/retrieval
  <  (is#information_search/retrieval
        >  (is#lexical_search/retrieval  >  is#regular_expression_based_search) )
  exclusion:  is#lexical_search/retrieval,
  >  {retrieval_of_data_indexed_by_knowledge  knowledge_search/retrieval},
  subprocess:  document_generation,
  instrument:  (knowledge-based_search_interface  <  (search_interface <  wn#user_interface)
                 >  {(dynamically_generated_knowledge-based_search_interface
                      static_knowledge-based_search_interface)} )__[any->1..*];

     retrieval_of_data_indexed_by_knowledge
       object: a (. pm#description_content/medium/container_indexed_by_knowledge
                    <  pm#description_content/medium/container,
                    knowledge_based_indexation/annotation: a pm#knowledge_representation),
       subprocess:  knowledge_retrieval;

     knowledge_search/retrieval
       >  {whole_graph_retrieval  graph_portion_retrieval___path_retrieval}
          {specialization_retrieval  generalization_retrieval  comparable_graph_retrieval}
          analogy_retrieval  {structure_only_based_retrieval  rule_based_retrieval}
          {complete_knowledge_retrieval  incomplete_knowledge_retrieval}
          {consistent_knowledge_retrieval  inconsistent_knowledge_retrieval}
          knowledge_search/retrieval_in_WebKB,
       subprocess: a (. knowledge_inference/reasoning/generation
                        >  {monotonic_reasoning  non-monotonic_reasoning}
                           {consistent_inferencing  inconsistent_inferencing}
                           {complete_inferencing  incomplete_inferencing}
                           {structure-only_based_inferencing  rule_based_inferencing} ),
       instrument:  (search_operator  <  wn#subroutine) __[*<->*],
       object: (a pm#knowledge_representation
                   object of:  a storing_knowledge_assertions_or_queries_in_a_document
                               a knowledge_comparison );
       /* //pm#knowledge_representation was declared at the end of Table 2.1.3.1:
          pm#knowledge_representation  <  wn#symbolic_representation,
            >  {pm#formal_term pm#formal_or_semi-formal_well-formed_statement};
       */

          structure_only_based_retrieval
            > { (structure_only_based_specialization_retrieval  <  specialization_retrieval)
                (structure_only_based_generalization_retrieval  <  generalization_retrieval)
                (structure_only_based_comparable_graph_retrieval
                  <  comparable_graph_retrieval)  };

          knowledge_search/retrieval_in_WebKB-2
            instrument:
              (search_operator_of_WebKB-2  <  search_operator,
                >  spec  gen  ext-spec  ext-gen  specGtypes  ext-specGtypes  comp  ext-comp
                   spec/noExcl  gen/noExcl  ext-spec/noExcl  ext-gen/noExcl  specGtypes/noExcl
                   ext-specGtypes/noExcl  comp/noExcl  ext-comp/noExcl
              ) __[any->1..*];
```

2.4.1. Knowledge-based Indexing of Any Document Element And Document Generation

2.4.1.1. Storing knowledge commands (assertions or queries) within an informal document.

- ***One way to do so is, within a text file, to use special marks for isolating the formal parts and/or the informal ones within special marks (e.g., HTML or XML marks).*** The formal parts can
  - use knowledge assertions or queries in a KRL, or
  - use procedures – or an API permitting to call procedures – written in regular programming language (e.g. MQL [www-MQL, 2009], the query language of Freebase; many of such APIs and procedures use languages such as JAVA or PHP).

  Many Semantic Web tools propose special "semantic tags" to include the result of certain semantic queries within a document (at the place the queries are), and such tags are often aimed to be used by Java programmers, not end-users.
  Microformats, RDFa and the language of Semantic Wikipedia are now popular ways to hide knowledge within HTML tags. Their ancestor is the approach of Ontobroker [Decker et al., 1998] which parsed knowledge stored within the (invented) "onto" attribute of the HTML tag "A" as in the following example.

```
<a onto="page:Researcher"></a> <!-- this HTML page is about a researcher -->
<a href="http://www.iiia.csic.es/" onto="page[affiliation=href]">IIIA</a>
                            <!-- this researcher is affiliated to IIIA -->
```

  Example with a microformat for geographic information:

```
The birds roosted at <span class="geo"><span class="latitude">52.48</span>,
                                       <span class="longitude">-1.89</span> </span>.
```

  Example with RDFa:

```
<p xmlns:dc="http://purl.org/dc/elements/1.1/"
   about="http://www.example.com/book/wikinomics">
      In his latest book <cite property="dc:title">Wikinomics</cite>,
      <span property="dc:creator">Don Tapscott</span> explains ...
      The book is due to be published in
      <span property="dc:date" content="2006-10-01">October 2006</span>.
```

  WebKB-1 permits to store knowledge representations within the "alt" attribute of HTML tags for non-textual elements (IMG, AREA, APPLET, and INPUT). Indeed, "alt" is meant to specify alternate text describing the content of these elements. This was used in WebKB-1 to index images and then allow their retrieval using conceptual queries that can be composed via a menu (http://www.webkb.org/kb/images/clubMed/). Here is an indexation used for one of the images (the representation is in CGLF since it was part of one of the earliest demonstration examples of WebKB-1, before CGLF was replaced with FCG and FE).

```
<img src="http://www.webkb.org/kb/images/clubMed/cocoTreeOnBeachNearStraightJetty.jpg"
     alt="[Coco_tree]- { ->(On)->[Beach]->(Near)->[Jetty:*j]->(Attr)->[Straight];
                         ->(Near)->[*j];
                       }">
```

  The authors of Ontobroker claimed that their approach permitted to avoid repetitions between formal and informal text. This is true at least for simple statements. However, this approach is mainly only useful for representing and hiding *simple* kinds of knowledge representations since i) using expressive KRLs with this approach is not easy, and ii) most of the representations (e.g., the relations) are always hidden and hence cannot be shown for displaying a precise and organized version of the information or cannot be used for navigating between objects. This is why WebKB does not use this approach but instead permits to isolate FS commands and control structures within special marks, e.g., HTML marks such as "<script language='FS'>" and "<script>". If needed, the document author may hide these commands using HTML comments.
  In certain contexts, WebKB-1 also permits to reuse certain HTML marks for simple knowledge representations, as shown by the next equivalent statements.

```
FE:   The car that has for owner John and has for weight 1750 kg.
FCG:  [the car, owner: John, weight: 1750 kg]
HTML: <dl><dt>The car <dd>owner: John
                      <dd><dl><dt>weight<dd>1750 kg</dl></dl>
```

- ***Another way is to use a structured document editor.*** One of the goals of my PhD thesis [Martin, 1996] was to support the management and interconnection of formal and informal information via formal and informal means, in a knowledge engineering context. To that end, I designed CGKAT (a Conceptual Graph based Knowledge Acquisition Tool) which integrated the structured document editor Thot [Quint & Vatton, 1992] with my extension of the Conceptual Graph (CG) workbench CoGITo [Haemmerlé & Guinaldo, 1999]. To permit this, using the data description and presentation languages of Thot – which were respectively equivalent to XML and much more powerful than CSS – I *first* designed a data/structure model and a presentation model for a CG to be edited in a graphical way and handled via Thot like any other document element (DE). Thus, any CG or any structural part of a CG could be hyperlinked to any other DE and had associated specific menus (generated from the data+presentation model for a CG) for editing or changing the presentation of this DE. *Then*, using another language of Thot, its API and my extension of CoGITo, I created an event model for a CG (assertion or query) which permitted any creation, modification or activation of a CG via Thot to be similarly created, modified or activated in a KB of CoGITo. Conversely, this also permitted the results of inferences made via CoGITo (query results or error messages) to be displayed in a document or menu of Thot.
  To my knowledge, such an integration of a structured document editor with an inference engine has so far not been replicated. However, when Amaya (the W3C/INRIA browser based on Thot but using XML and CSS) finally has data/presentation/event model specification languages and graph editing capabilities as powerful as those of Thot, my work with Thot could be re-used to create an API for Amaya allowing i) the graphical display, editing and embedding of knowledge representations in XML/HTML documents, and ii) the exploitation of external inference engines to check these representations and answer queries. Such an API would likely be very popular since many KR notations currently exist and are not easy to display, edit and embed within XML/HTML documents.

Here are some types and relations to begin categorizing some of the above approaches and thus for example guide an author of a microformat-based approach in its categorization. This is also an example of an organized summary of information (the one given in the first above bullet point).

```
storing_knowledge_assertions_or_queries_in_a_document
  >  (creating_a_document_intertwining_formal_representations_and_informal_text
       > {( (inserting_formal_representations_in_a_structured_document_language_or_editor
              >  inserting_XML-based-formal_representations_in_an_XML_document)
            inserting_formal_representations_without_structured_document_language/editor )}
         (using_a_command_based_KRL_within_informal_text
           >  using_FS_within_an_informal_text)
         (using_a_programming_API_within_an_informal_text
           >  using_MQL_within_an_informal_text),
       subprocess:  (embedding_formal_or_informal_representations_between_special_marks
                      >  (hiding_formal_representations_within_XML/HTML_tags
                           >  using_micro_formats  using_RDFa)
                         (embedding_formal_representations_between_marks
                           >  embedding_formal_representations_between_HTML_marks
                              (embedding_formal_representations_between_FS_marks
                                 object:  pm#AND-set_of_statements_in_FS,
                                 description_place:  pm#document_element,
                                 instrument: (FS_mark  <  pm#string,
                                              > (FS_end_mark  instance:  "<script>"  ")$")
                                            ) ) ) )
     );
//subprocess:  knowledge_extraction/modeling/representation;  //already stated
```

2.4.1.2. Indexing any document element (DE) using knowledge. WebKB-1 permitted to refer (and index) a DE before languages such as XLink, XPointer, XQuery and XPath appeared and, unlike them, permits to refer to *any* DE. Indeed, even if the DE has not been isolated via XML or HTML tags, WebKB-1 permits to refer to it by indicating its content and its number of occurrence in the document. Here is an example of two DE indexations in WebKB-1.

```
$(Indexation
   (Context:  Language: CG; Ontology: http://www.bar.com/topLevelOntology.html;
              Repr_author: phmartin; Creation_date: Mon Sep 14 02:32:21 PDT 1998;
              Indexed_doc: http://www.bar.com/example.html;
   )
   (DE: {2nd occurrence}  the red damaged vehicle )
   (Repr: [Color: red]<-(Color)<-[Vehicle]->(Attr)->[Damaged] )
)$
$(DEconnection
   (Context:  Language: CG; Ontology: http://www.bar.com/topLevelOntology.html;
              Repr_author: phmartin; Creation_date: Mon Sep 14 02:53:36 PDT 1998;
   )
   (DE: {Document: http://www.bar.com/example.html} )
   (Relation: Summary)
   (DE: {Document: http//www.bar.com/example.html} {section title: Abstract} )
)$
```

2.4.1.3. Sending commands with GET parameters, generating virtual documents and associating (menus of) commands to objects. The WebKB-1 and WebKB-2 servers can be sent one or several FS commands using the GET and POST parameter encoding methods of the HTTP protocol. The graphical interfaces for these servers may use HTML forms which send GET/POST commands to these servers. In answer to such such requests, these servers always generate and send back an HTML document containing the results of the commands (if they have results or if their execution must be acknowledged), in the given order of these commands, and preceded by text of the commands when this is specified via a presentation option. To send a command, an alternative to using a form is to use an HTML hyperlink whose destination URL begins with the URL of a server and whose GET parameters includes commands: clicking on the hyperlink calls the server and permits to see the results of the command. This execution of commands by activating an hyperlink is called "dynamic/virtual document generation" by researchers on hypertext/structured documents.

By default, in the results provided by WebKB-2, each displayed category name (identifier or simple name) is within an hyperlink that can call WebKB-2 with a search command permitting to see the direct relations from this category and the hierarchy of all its important supertypes. Each name of a *transitive* relation type is within an hyperlink that permits to call WebKB-2 to see the hierarchy of the transitive closure of this relation from this category. Menus at the bottom of each generated page (see Figure 2.4.3.2) permit to change the default search and presentation options (e.g., the exploration depth, the used KRL, the kinds of filtered out relation types, and the kinds of authors the knowledge of whom should be filtered out). Figure 2.4.3.3 shows the current interface for selecting some of these options in WebKB-2. This approach permits

- an easy navigation of the KB,
- the use of a common interface (a Web browser) – and hence, for example, the use of the "back" button and scrollbars or the opening of multiple windows – to see and exploit the shared KB, and
- the display of a large quantity of information when a textual format is used (e.g., *all* the categories sharing a same name and, for *each* category, *all* its direct relations, *all* its supertypes and, *all* its instances).

Subsection 2.4.3 illustrates all this. This approach is also very flexible and relatively easy to implement. However, instead of only associating a search command to objects, it would be preferable to associate a cascading pop-up menu showing i) the direct and indirect relations from/to the object, and ii) commands for adding, removing, searching or comparing relations from/to this object or comparing it with a related object, with for each command a sub-menu for changing the default options. This would avoid the necessity to use *separate* menus for finding or accessing commands that may apply to the selected object. Separate menus require scrolling or opening another window and

hence require to hide some of the displayed information. This object-centered interface is currently only partially implemented in WebKB-2 and hence is not yet usable on its main site (www.webkb.org).

2.4.1.4. A language of commands for a RESTful Web service. Despite the advantages of the approach described in the previous paragraph, WebKB-1 and WebKB-2 still seem to be the only knowledge servers that propose a language of commands that can be used with GET parameters and hence

- can be considered as a REpresentational State Transfer (RESTful) Web service [Fielding, 2000] (any application may send FS commands to WebKB-1 or WebKB-2 and then interpret the results, e.g., an RDF/XML file if this KRL has been selected for the output KRL), or
- permit to exploit other RESTful Web services.

When they can be sent commands, other knowledge servers use verbose messages such as KQML/SOAP/XML messages or use an API and sends calls to the server via protocols such as AJAX or the Java Message Passing Interface. The two approaches (RESTful and message-based) are complementary.

2.4.1.5. Script/shell-like commands, document exploration/exploitation/generation commands and knowledge search/generation commands. WebKB-1 and WebKB-2 provide these different kinds of commands. Here are examples of commands that can be selected (and then sent to WebKB-1 for execution) at http://www.webkb.org/interface/syntaxBasedIR.html

```
    //the next command loads in parallel all the files accessible via hyperlinks (1 level of
    //exploration) from http://www.webkb.org/kb/index.html and, via the shell "pipe", applies
    //the shell command "grep" on each of them to see the lines including the word "knowledge"
accessibleDocFrom -maxlevel 1 -HTMLonly   ../kb/index.html | grep -i knowledge
    //example of a simple loop; a loop is a control command:
set a "1 2 3";  for i in $a { echo $i; };
```

Here are examples that can be selected (and then sent to WebKB-1 for execution) at http://www.webkb.org/interface/knowledgeBasedIR.html

```
    //maximal join on statements that include (a specialization of) KADS1_Model_of_Expertise
load http://www.webkb.org/kb/webkb1/KADS1.html;  spec [KADS1_Model_of_Expertise]  | maxjoin;
    //display of the subtype hierarchy of thing_needed_for_KADS_knowledge_engineering
spec thing_needed_for_KADS_knowledge_engineering;
```

The Sisyphus-I problem – an office allocation problem originally used for comparing knowledge acquisition tools and later also used for comparing CG inference tools (Sisyphus-I track of ICCS 1999) – was solved in WebKB-1 using a procedural script written in FS (searches for specializations of types and statements were combined via control structures grouped in procedures) [Martin & Eklund, 1999a]. The three files (ontology, dataset and procedures) for this experiment, the menu for loading and executing them, and the article explaining the solution are accessible at http://webkb.org/kb/webkb1/sisyphus1.html

2.4.1.6. Options to display the indexed document elements. When WebKB-1 retrieves knowledge representations in answer to a query, depending on the selected presentation options, it displays each representation *and/or* the DE itself. Each displayed DE and representation is associated to an hyperlink (directly, or via an hyperlink that follows the DE/representation and that has "Source" for textual content) that permits to retrieve the Web file – and the place within this Web file – where this element (representation or indexed DE) is stored. This hyperlink does not directly refer to the file (e.g., an input file) where this element is stored since this element may not be referable via a URL. Instead, the hyperlink contains a call to WebKB-1 which makes it generate a copy of the file (opened at the place of the element, when this is an HTML file) with the element preceded by "---------- Source ----------" and followed by "----------" or simply highlighted in pink if it is a block of text (e.g., if it is an indexed sentence or a description in FCG).

2.4.2. Operators For Searching and Comparing Categories Or Statements

2.4.2.1. Background. In this document, "(conceptual) graph" is a synonym of "(semi-)formal statement". This synonymy is used a lot in this subsection, not much in the rest of the document. Graph matching (or "projection") to test if a query statement generalizes another statement is a classic operation at the core of many inferencing mechanisms in CG related works since 1984 and other works such as for example Algernon, an inference system based on a tractable reasoning system called Access-Limited Logic [Crawford & Kuipers, 1991]. Projection has been shown to be a sound and complete way to test if the query graph is a logical deduction of another graph when both are "simple graphs" (non-redundant existentially quantified conjunctive formulas) and even when this other graph has a context considered as non-restrictive ("positive context"); in both cases, if the query graph is a tree (i.e., without cycle), the projection can be computed in a polynomial way [Chein & Mugnier, 1997]. Rules with simple graphs as premise and conclusion, interrelated by variables, have been shown to be sound and complete with respect to first order logics (FOL) [Salvat & Mugnier, 1996]. Full CGs (simple CGs plus classical negation) have also been proved sound and complete with respect to FOL [Kerdiles, 2001]. An algorithmic study of deduction in simple CGs with classical negation can be found in [Leclère & Mugnier, 2008]. These last cited works are implemented in CoGITaNT [Genest & Salvat, 1998], the successor of CoGITo – the CG workbench that was re-used and extended in CGKAT and WebKB-1. Unlike WebKB, various systems – e.g., MIEL [Haemmerlé et al., 2007] – handle simple CGs extended to take into account fuzzy values.

2.4.2.2. Efficiency of searching graphs in WebKB-2. In WebKB, the 'spec'/'gen' and 'ext-spec'/'ext-gen' operators use a graph-matching algorithm on all statements in the KB and these algorithms use a simple depth-first exploration of both graphs in parallel (the query graph and a candidate asserted graph). This exploration starts from their first nodes and continue as long as the types and quantifiers of the nodes match (otherwise the exploration is started from another node in the query graph or then in the other graph). Checks are made to avoid loops.
Hence, these algorithms do not have a polynomial complexity and bring nothing new from a theoretical viewpoint. Indeed, given these algorithms are meant to compare any pair of graph of any expressiveness (level by level when contexts exist and, for example, with a quantifier such as 'at least 85%' specializing the quantifier 'at least 60%' or 'most'), the complexity of the structures to compare meant that writing a polynomial algorithm would have been complex even though possible since the graph matching approach is no different from the one in a classic projection.
As previously noted, when graphs that are not simple graphs are compared, the graph-matching in WebKB does not amount to checking that the query graph is a logical deduction of the other one, it amounts to checking that this other graph is "relevant" to the query.
Given that for their queries people most often use statements without cycle nor context, the fact that these algorithms do not have a polynomial complexity never proved a problem even in the case of a search of extended specializations with hundreds of potentially relevant (simple) graphs to compare with the query – more precisely, hundreds of potentially relevant graphs amongst about 3000 graphs and with about 200,000 relations between categories. These last two numbers are not very important given the efficient category indexation provided by the underlying OODBMS: ***the answer was always computed within a second or two***. This can be checked by issuing queries at http://www.webkb.org (the computing time calculated using the Unix function 'gettimeofday()' is given at the end of the results). The ***time to load the database in memory*** (only for the first call to WebKB-2 if the calls are temporally close to each other) and the ***network delay*** (the machine where the main site of WebKB-2 is hosted has so far been in Australia) are ***more important factors since they each are at least two seconds long.*** The hosting machine is a classic Unix individual workstation and, so far, the graphs are not even ordered in a specialization hierarchy.

2.4.2.3. Extensions to traditional search operators and implications on the structure of the KB of WebKB-2. Such searches for specializations (or generalizations) of a query graph permit searches "by the content" and are relatively classic, even though an originality of WebKB-2 is to exploit an extended specialization relation on arbitrary graphs and to exploit it in a large KB. Below are some extensions to the methods for these classic searches. These extensions are currently not used in CG related works, except for one: a rule-based generalized form of "the third kind of extension presented below" is used in GALEN and presented in [Rogers & Rector, 2000].
Except for the first extension, the full implementation of these extensions in WebKB-2 has been put on hold because of a near-future ***structural change of the KB*** that will seriously ease i) their implementation, ii) knowledge navigation and presentation (especially in FL), and iii) a full implementation of certain fine points of the collaboration supporting mechanisms. This structural change consists in not keeping separate the statements provided by the users but *integrating each of their relations into a unique network the way it is presented in FL* (thus, with contexts associated to relations rather than to whole statements). Hence, at least internally, this new network will be more similar to those of description logics or other traditional logic-based semantic networks than to CG-based KBs (where users' statements are kept separate, as entered). In addition to easing the full implementation of the extensions, this structural change will speed up graph comparison because the structures will be simpler (the tables/classes ConceptNode and RelationNode will not be needed anymore) and hence faster for an OODBMS to traverse or manage.

1. **Using unions of graphs.** Let us assume the KB includes the separately entered FCGs [John, owner of: a car] and [John, owner of: an apartment]. A classic search for graphs specializing the query FCG ***[a man, owner of: a car, owner of: a lodging]*** would not retrieve the previous graphs since only their union specializes the query graph. When WebKB-2 tests if a graph G is a specialization of the query graph, it also looks for more information in graphs related to G i) by a same individual (i.e., with a same identifier or coreference variable), ii) that use a type in G with a universal quantifier (with an existential quantifier, there may not be any connection), or iii) that define necessary conditions for a type in G. If the union of G and those related graphs permits to answer the query graph, they are displayed but separately since joining them would often not produce a meaningful graph (e.g., their embedding graphs could not be joined). As another example, the following two FCGs could also be presented in answer to the previous query:
   ```
   [ [[the IBM_employee Tom, owner of: an apartment], time: 2000], author: Tom]
   [ [any IBM_employee, owner of: a car], author: IBM]
   ```
   However, WebKB-2 is not able to perform type expansion (i.e., replace a type by its definition within a graph) nor exploit rules – unlike in the CG tools CoGITaNT [Genest & Salvat, 1998] and Corese [Corby et al., 2004] for example; see also [Le Duc & Le Thanh, 2003] for a study about the combination of revision production rules and description logics.
   With the future network tightly integrating all the statements, searches for query graph specializations will not retrieve separate users' statements (as entered) but portions of the network. If two of these portions overlap, presenting them separately or not will depend on a presentation option. By default, they will not be presented separately.

2. **Exploiting partOf relations along with generalization relations.** For instance, let us assume that the categories representing the geographical areas 'Gold Coast' and 'Southport' are connected via a pm#part relation and that the KB includes the following FCG.
   ```
   [spamOnly@phmartin.info,  agent of:
      (a renting,  object:  (an apartment,  part: 1 bedroom,  place: Southport),
                   instrument: 140 Australian_dollars,  period: a week,
                   beneficiary: Spirit_Of_Finance)]
   ```
   The pm#part relation between the categories for 'Gold Coast' and 'Southport' should be exploited to find that this graph specializes the following one.
   ```
   [an apartment, place: (a district, part of: Gold_Coast)]
   ```
   In the general case, various rules such as those given in [Rogers & Rector, 2000] are required to state what can be deduced from certain combinations of certain partOf relations and generalization relations.

3. **The operators spec, gen, ext-spec, ext-gen, specGtypes, ext-specGtypes, comp and ext-comp.** Let us assume that the graph [John, owner of: a housing] is in the KB and that a query graph is [a man, owner of: an apartment]. The first graph is not a specialization of the query graph; it is only a "comparable" graph since wn#housing is a supertype of wn#apartment (hence a comparable type), not a subtype. However, a user may want such a graph to be provided as an answer for the above cited query graph. This is why WebKB-2 provides the search operators 'specGtypes' and 'ext-specGtypes' (alias '?') that respectively extend 'spec' and 'ext-spec' by also taking into account supertypes of the types of nodes in the query graph, not just subtypes or identical types. Except for type comparisons, 'specGtypes' and 'ext-specGtypes' are identical to 'spec' and 'ext-spec': the quantifiers of a matching graph must specialize or be identical to their counterparts in the query graph, and the matching graph may have more relations and nodes than the query graph.
   The operators 'comp' and 'ext-comp' lift these last constraints: the first can be seen as a union of 'spec' and its inverse 'gen', while the second can be seen as a union of 'ext-spec' and 'ext-gen'. They proved necessary in an application for the retrieval of accommodations and other tourism related services on the Sunshine Coast (Australia) since many hotels had not published certain services or their prices but still offered such services. In the interface of this application, a checkbox permitted to allow the "listing hotels not providing information to answer the query", and hence the use of the last cited operators.

4. **The operators spec/noExcl, gen/noExcl, ext-spec/noExcl, ext-gen/noExcl, specGtypes/noExcl, ext-specGtypes/noExcl, comp/noExcl and ext-comp/noExcl (alias '??').** A category X 'gen/noExcl' a category Y if X generalizes ('gen') Y (i.e., is a supertype of Y, a type of Y or a supertype of a type of Y) or is a specialization of a generalization of Y which is not exclusive with Y and which does not specialize Y. The 'gen/noExcl' operator is like the 'gen' operator except that it uses the above described gen/noExcl relation type instead of the pm#type_generalization relation type for comparing categories. Its inverse operator is 'spec/noExcl'. The other operators cited in the title of this paragraph are similarly constructed by extending those cited in the title of the previous paragraph. The rationales for these operators are similar to the rationales for extending 'spec' with 'specGtypes' and 'comp': even if this is not made explicit in the KB, the above category X could be a generalization of Y or they could have shared specializations, and hence Y is a relevant object for a spec/noExcl query using X. For example, a statement using (specializations of) the role types wn#adventurer or wn#achiever (which are non-exclusive subtypes of wn#person) is a correct answer to 'spec/noExcl [a wn#achiever]'.
   In many research works, e.g., [Corby et al., 2007], search operators are based on similarities between categories and these similarities are often computed via statistical measures (sum, ...) based on the number of specialization or generalization relations that permit to connect two categories. The above cited operators could be further specialized by other operators that also exploit such statistical measures. The important point is that categories directly or indirectly related by an exclusion relation are considered as *not similar*. Within the MSO, many direct or indirected exclusion relations have been set between the top-level categories of WordNet (about a hundred categories) and hence between a good part of randomly selected pairs of WordNet categories. Unfortunately, below these top levels, few exclusion relations exist between WordNet categories.

5. **Path expressions.** They should be permitted in query graphs. This is partially the case in SPARQL, the query language proposed by the W3C for querying RDF resources. SPARQL has been extended in various directions, e.g., for update purposes [Seaborne et al., 2008], for making it query XML embedded in RDF [Corby et al., 2009] and for extending its path description features with regular expression patterns [Alkhateeb et al., 2009]. In any case, the *SQL inspired syntax* of SPARQL is hardly a model of flexibility and concision, and is more adequate for retrieving individuals than (portions of) graphs. Using common regular expression operators (with '*' meaning '0, 1 or many times', '+' meaning 'at least 1 time' and '?' meaning '0 or 1 time') in query graphs expressed in a language such as FCG brings flexibility and concision. Let us assume the following FCG is in the KB.

```
[spamOnly@phmartin.info, agent of: (a research, within_group: KVO_group)]
```

   Users looking for a person conducting research on the "Gold Coast campus of Griffith Uni." (QLD#GCcGU) are unlikely to find this graph via classic searches for specialization only. However, since the category

pm#School_of_IT_at_GCcGU is connected via a pm#part relation to pm#KVO_group and via a location relation to QLD#GCcGU and since pm#relation is the uppermost relation type, it should be possible to find this graph with any of the following queries using FCGs.

```
spec [a person, agent of: (a research, relation+: GCcGU)];
spec [a research, (relation: a thing)+ place: GCcGU)];
spec [a research, relation 3+ (part of: a group)3+ place:GCcGU)];
  //('3+' means that at most 3 relations of the specified type should be traversed).
```

2.4.2.4. Searching categories. The above cited operators can be used for searching categories, not just graphs. For example, 'spec "cat" -depth 3' displays all the categories with name "cat" with their direct relations and their specialization hierarchy on three levels – direct relations are displayed (or not) from each of the specialization depending on the selected presentation format. The command ' ext-spec "cat" ' displays i) all the recorded direct relations from the string "cat" (this includes its direct extended specializations such as the categories having "cat" as name), and ii) the hierarchy of its extended specializations.
Another way to search categories is to specify variables in a query graph, as in the following example:

```
spec [a person ?x, agent of: (a research, place: ?y)]
```

In such cases, instead of returning graphs, the search operators return the set of identifiers of the instances corresponding to these variables in the graphs (e.g., {pm, GCcGU}) or, when an instance is not specified in the graphs, the node is given, as in {pm, [a place]}. There is no difference between such (query) variables and (coreference) variables used for specifying a cycle in the graph; hence, if there is a cycle in the query graph, some variable should be used for both goals.

2.4.2.5. Filtering categories or graphs. Before it is displayed, knowledge may be filtered out. WebKB-2 permits to use ***two filters***: one for which the *given users, categories or graphs* specify that ***only*** knowledge believed by these users and specializing these categories or graphs ***can*** be displayed, and one for which the *given users, categories or graphs* specify that ***all*** knowledge believed by these users and specializing these categories or graphs ***cannot*** be displayed. Both are evaluated on any piece of information to display, and in that order.
Depending on the selected presentation options, a filtered out category or graph is fully hidden or displayed via a simple dot (such dots are useful in long hierarchies of categories or statements in order to better see that some levels have been filtered out). In the near future, this dot will have an associated menu that will show its identifier and – like other categories in this near future – direct relations from/to it and commands that can be applied to it.
Currently, WebKB-2 does not provide a more precise presentation scheme to display certain categories or graphs in smaller/larger font or with particular effects (italic, bold).

2.4.2.6. Comparing categories via search commands. Here are two examples using FL graphs.

```
?  [wn#cat  <  wn#feline];   spec [wn#cat  pm#relation 4+: wn#dog];
```

2.4.2.7. Categorizing operators for knowledge search or generation – future research work. This subsection has proposed five extensions to the classic search of specializations of a query graph, derived 15 operators from the classic "spec" operator, and mentioned other search and presentation options. Various options or parameters can *also* be associated to knowledge generation operators such as a "maximal join". For example, one of these options can specify if such a join is maximal with respect to the number of matched *or* final (concept *or* relation) nodes in the result graph. It would be interesting to categorize all these operators – with their extensions and options – as specializations of pm/km#knowledge_search/retrieval.

2.4.2.8. Organizing long lists of statements – future research work. Even with a KB where all the statements are integrated into a unique network (i.e., where the statements are only isolated by their contexts) and where the statements are organized in specialization and/or partOf hierarchies (that the user can navigate, contract or expand), it may happen that a rather unorganized long list of statements is displayed in answer to a query. Here are two avenues that will be pursued in WebKB-2 to permit an *automatic* organization of such lists.

- A first idea is to let people associate ***presentation directives*** within the definitions of some important conceptual categories, e.g., pm#process and pm#person. This would allow user-defined presentations. More generally, a ***presentation specification language*** (with commands based on the ontology presented in Section 4.5) will be proposed to the user. This presentation language will be complementary to Fresnel [Pietriga et al, 2006]. Indeed, unlike Fresnel, it will only deal with simple presentation details (such as "what kind of information to put in bold characters"), not with graph layout details. On the other hand, it will be integrated in FS and hence the commands will be easy to combine (e.g., via "pipes" as in Unix) and associate to hyperlinks, thus easing the creation of virtual documents for presenting the content of the KB.
- Another approach – which unlike the previous one is user-independent – would be to ***automatically structure the statements of the long list according to various topics to which these statements are related.*** Some examples of topics that can be used to group statements about a certain person are: physical characteristics, administrative details, possessions, work related activities, etc. Such groups (and their sub-hierarchies) can be automatically found based on either the type(s) of the *first* relation of each statement or, in case this type is too common for efficiently partitioning the knowledge, the type of the destination concept node of the first relation. For an efficient partitioning, it may turn out that other complementary information also have to be exploited, e.g., the types of the *second* relations and their destinations.

2.4.3. Examples of Static Interfaces Proposed by WebKB-2 For Search and Presentation

Figure 2.4.3.1 shows one of the earliest interfaces of WebKB-2 for searching graphs. Unprefixed terms refer to category names. The user is `pm'. Menus are proposed for selecting quantifiers and names/identifiers of basic/general categories for concept/relation types. The "Options" listed in this early interface permit to specify a filter on the creators of the retrieved graphs, and to specify that HTML tags (and hence hyperlinks) must be used for the presentation of the formal and informal parts of the generated file. The edit box next to the "Submit to" button shows how the command built via this interface is encoded and sent by it to WebKB-2.
Figure 2.4.3.2 shows the result of this call. The generated page first presents the two categories that share the name "Gold_Coast" (this presentation uses an informal format similar to FL) and mentions that the first category is selected as an interpretation for this word in the query. Then, an FL graph ("with hyperlinked categories") answering the query is presented. Finally, the search time is displayed and various menus for making other searches or changing the default formatting options are given.
Figure 2.4.3.3 shows the menu that WebKB-2 proposed (from 2006 to 2009) for changing these default formatting options.

Figure 2.4.3.1. Query for the specializations of a graph

Figure 2.4.3.2. Result of the query in the previous figure

Figure 2.4.3.3. Menu for changing the default formatting options

Figure 2.4.3.4 shows the menu proposed by WebKB-2 (from 2006 to 2009) for searching categories. Some user has entered the category name "person". Figure 2.4.3.5 shows the beginning of the results for this name. However, to display more information in this figure, these results were manually slightly compacted. In this figure, the default category creator is `wn', the default graph creator is `pm', the format used for displaying the direct and indirect relations is the above mentioned "informal format similar to FL". This figure shows that currently in WebKB-2 some statements are kept separated: not all the relations are integrated as much as possible into a unique network. In these separate statements, node annotations are represented within "(^" and "^)" delimiters. To save spaces, such node annotations are also used in the next subsection. The identifiers of these separate statements were automatically generated except when their creators provided an identifier for them.

Figure 2.4.3.4. Menu for searching categories

Figure 2.4.3.5. Result of the query in the previous figure

2.4.4. Generated Search/Entering Interfaces

The previous figure shows some universally quantified statements about wn#person (with annotations for some of their nodes) and, towards the bottom, an hyperlink for accessing a form to enter statements about a (specialization of) wn#person. The next figure shows the beginning of this form. It is generated based on the above cited "universally quantified statements" (alias, "schemas"). The directives "no inheritance" and "explore" stored in the concept node annotations control the generation of the form. The first directive prevents the use of schemas associated to supertypes of the category. The second leads to the generation of an hyperlink to another form for detailing a related object. In other words, this second directive permits the use of schemas for related objects and hence enables form cascading. Figure 2.4.4.2 illustrates form cascading and Figure 2.4.4.3 illustrates the final result. The second directive ('explore') is also used to control the depth of menus generated using subtype partitions. For example, the categories representing colors and days of the week are organized into hierarchies of subtype partitions; such partitions permit WebKB-2 to generate organized menus which do not display categories likely to be irrelevant for the user.

Figure 2.4.4.1. Generated menu for searching persons

Figure 2.4.4.2. Generated sub-menu of the previous form

Figure 2.4.4.3. Result of the command generated by the two previous menus

Such generated forms can also be used for searching statements about a category. The only difference is that the generated command is not a graph assertion but a query graph preceded by the operator "?". Such forms are generated based on schemas (definitions or universally quantified statements, e.g., with quantifiers such as "any", "most" and "at least 60%") about this category and its supertypes. Such forms ease and guide (and hence normalize) knowledge capture. Currently, in the MSO, schemas are mostly associated to top-level concept types. These schemas are inherited by all types in the ontology that have no overriding schemas. They include the most useful relations from a certain object, thus permitting the user to ignore i) many imprecise relation types imported from certain ontologies, and ii) relation types with structural purpose only (e.g., pm#relation_from_spatial_entity). As Figure 2.4.4.1 shows, each form also has a field to permit the use of relation types not listed in the form.

To guide and facilitate the representation of knowledge by average users, many specialized schemas are required, e.g., about categories for "house", "car", "selling", "renting", etc. Unfortunately, the MSO does not yet include such schemas. However, users may create and associate schemas to any category.

2.4.5. Use or Generation of Scalable Comparison Tables – Example with the Beginning Of an Ontology of CG Tools

For representing certain comparisons of objects, such as the comparison of the features of certain techniques or tools, it is useful to use tables as display supports. Such tables can be formal or semi-formal and can be used as input or outputs. Manually creating detailed tool comparison tables is often a presentation challenge and involves a person's knowledge of which features are difficult or important and which are not. Furthermore, it would be too restricting to use *predefined* tables for easing the entering of tool features and then compare them. Hence, generating tables from the KB is needed. Then, allowing the dynamic modification of these tables, via menus associated to its elements, is a nice way to support knowledge entering.

Fact Guru [Skuce & Lethbridge, 1995] is one of the rare KB servers that generate comparison tables. More precisely, it permits the comparison of two objects by generating a table with i) the object identifiers as column headers, ii) the identifiers of all their features/relations as row headers, and iii) for each cell, either a description of the destination object or a mark to state that the feature/relation does not exist for this object The common generalizations of the two objects are also given. However, Fact Guru's approach is not structured enough to be scalable: the list of features/relations from the compared objects is not structured and the cells are allowed to be informal descriptions of the destinations of the relations.

A more scalable approach is to organize the features of the compared objects into an "extended specialization" hierarchy and to use the cells only for indicating whether each compared object has or has not (or will have and when) each feature. In this document, such tables are called "scalable comparison tables".
Table 2.4.5.1 shows an example of table generation query using an FCG-like syntax followed by its result. The query is composed of: 'compare' (the comparison operator), pm/km#WebKB-2 and pm/km#Ontolingua (the two objects to compare), 'on' (some syntactic sugar), an FCG expression specifying the direct relations on which to compare the objects, and a maximal depth for the specialization hierarchy that permits to show how the specified direct relations share certain kinds of destination objects. The '>' before the query is a query prefix displayed by the system, like the prompt in the Unix shell.
Table 2.4.5.2 shows the FL statements used for generating the result. The reader is invited to compare the content of these two tables in order to understand the generation mechanism. In the cells, '+' means "yes" (the tool has the feature), '?' means that this piece of information has not been represented, and '-' means "no" (see the last line of the table). A maximum depth of automatic exploration is given; past this depth, a manual exploration of certain branches (similar to the opening or closing of sub-folders) should permit the user to give the comparison table a presentation better suited to her interest. More than two tools could be compared. The parsing options remain the same as before.

In the first column of Table 2.4.5.1, one entry is an FL expression for a category representing a certain kind of features that has not yet been named (i.e., no category has yet been entered to represent this particular kind) but that is generated to permit the comparison of the tools.

Table 2.4.5.1. Generation of a scalable comparison table

```
>  compare WebKB-2 Ontolingua on
     (support of: a is#information_search/retrieval,  output_language: a KR_notation,
      part: a wn#user_interface),  maxdepth: 5


                                                                  WebKB-2   Ontolingua
support of:
  is#information_search/retrieval                                   +          +
    is#lexical_search/retrieval                                     +          +
      is#regular_expression_based_search                            +          ?
    knowledge-based_search/retrieval                                +          ?
      structure_only_based_generalization_retrieval                 +          ?
      structure_only_based_specialization_retrieval                 +          ?

output_language:
  KR_notation                                                       +          +
    (*x  < pm#language,  wn#expressiveness: FOL)                    +          +
      FCG                                                           +          ?
      KIF                                                           ?          +
    XML-based_notation                                              +          ?
      RDF/XML                                                       +          ?

part:
  wn#user_interface                                                 +          +
    is#HTML_based_interface                                         +          +
    is#API                                                          +          ?
    is#CGI-accessible_command_interface                             +          ?
    is#graph_visualization_2D_interface                             -          -
```

Table 2.4.5.2. Statements exploited for generating the previous table

```
/* Reminder (see Table 2.4.1):
is#information_search/retrieval
  >  { knowledge-based_search/retrieval
       (is#lexical_search/retrieval  >  is#regular_expression_based_search) };
*/
input_language .(wn#computer_program *c, pm#language *l)
  relation_source:  (*c  support of:  //pm#support___description_support
                          (a wn#parsing *p   input: (a statement  language:  *l)) );
wn#user_interface
  >  is#HTML_based_interface  is#CGI-accessible_command_interface   is#API
     is#graph_visualization_interface;
KR_notation
  <  sumo#language,
  >  (XML-based_notation  >  RDF/XML)
     (KIF  wn#expressiveness:  FOL)  (FCG  wn#expressiveness:  FOL);

Ontolingua
  part:  1..* is#HTML_based_interface
         0 is#graph_visualization_2D_interface,  //no such interface in Ontolingua
  input_language:  a KIF,  //Ontolingua supports at least one version of KIF
  output_language:  a KIF,
  part:  an Ontolingua_library   0 is#DBMS,  //no DBMS in Ontolingua
  support of:  a is#lexical_search/retrieval;  //some lexical search is supported

CG_related_tool   < language/structure_specific_tool,
  instrument of:  Conceptual_Graph_specific_process,
  >  CG-based_KBMS  CG_graphical_editor  NL_parser_with_CG_output;

     CG-based_KBMS  <  KBMS,   >  {CGWorld  PROLOG+CG  CoGITaNT  Notio  WebKB};

         WebKB
           >  {WebKB-1  WebKB-2},
           url: http://www.webkb.org,
           part:  1..* (. is#user_interface_in_WebKB  <  is#user_interface,
                          part:  a is#HTML_based_interface   a is#API
                                 a is#CGI-accessible_command_interface
                                 0 a is#graph_visualization_2D_interface
                       ),
           input_language:  a FCG  a FL  a FE  a RDF/XML,
           output_language:  a FCG  a FL  a FE  a RDF/XML,
           support of:  1..* structure_only_based_generalization_retrieval
                        (structure_only_based_specialization_retrieval
                           //In WebKB, a specialization_structural_retrieval of PCEF graphs
                           //  (positive conjunctive existential formulas; see Table 2.1.3.3)
                           //  via PCEF queries is consistent and complete w.r.t. to
                           //  deduction in 1st order logic:
                          > (PCEF_specialization_structural_retrieval_in_WebKB-2
                              <  complete_knowledge_retrieval  consistent_knowledge_retrieval,
                              parameter: a (. PCEF_query  < pm#PCEF_statement pm#query),
                              object: a (. pm#PCEF_statement  < pm#PCEF_statement))
                        )__[any->1..*];

           WebKB-2  support of:  1..* is#regular_expression_based_search,
                    part:  1 is#FastDB   1 MSO_of_WebKB-2;
```

In the general case, the above approach (where descriptions are put in the rows and organized in a hierarchy) is likely to be more readable, scalable and easier to specify via a command than when the descriptions are put in cells as in Fact Guru. However, for simple cases, putting descriptions into cells may be envisaged as a shortcut: for example {FCG, KIF} may be displayed instead of '+' for the output_language relation.

In addition to generalization relations, 'part' relations could also be used. The '>part' relation between domains or theories (e.g., between the above cited F.O.L and P.C.E.F.) is already covered since it is an "extended specialization" relation.

Notes on the comparison of tools; informal comparison of six CG tools on 160 criteria. To permit the comparison of tools, a lot of information should be represented and the same structures or relations should be used by the various contributors, for example when expressing what the input languages of a tool can be.
There is a demand for the comparison of the dozens existing ontology editing tools since Michael Denny's "Ontology editor survey" [Denny, 2004] attracted interest despite being superficial and poorly structured. These characteristics made this survey often misleading, at least for people who are not knowledge engineers. Hence, to actually permit a genuine comparison of tools, it would be interesting to i) encourage and guide the creators of ontology editors in representing the features of their tools in the MSO, and then ii) support the generation of precise comparisons of tools as above illustrated. This would first require a categorization of the most important knowledge management processes and a categorization of many important features. As a first step towards this goal, within an input file, [Martin, 2004] categorized six CG-related tools according to 160 criteria represented by informal categories that were organized in a specialization hierarchy (although a completely informal one) and grouped into six sections (with one "scalable comparison table" per section). These sections were respectively titled

- "Availability, Accessibility, Interfacing" (see Table 2.4.5.3),
- "Input textual languages",
- "Output textual languages",
- "Data model: the kinds of storable objects",
- "Search and inferencing features",
- "Handling of multiple users/sources/back-ends" (see Table 2.4.5.4), and
- "Ontology-based knowledge entering guidance"

The important number of '?' in the next two tables shows that many important pieces of information are difficult to find in the documentations associated to the tools.
For two years, this work was accessible from the "CG tools" home page on Wikipedia and, on several occasions, the authors of the compared tools were invited to correct or complete this comparison. However, it must be acknowledged that very few updates and no extensions were entered by the creators of these tools despite the fact that i) these creators are researchers in knowledge representation, ii) they participated to workshops about CG tool comparison, and iii) the categorizations were simpler than formal ones would be. After two years of warnings, the selection committee of Wikipedia removed this page (and the "CG tools" home page) because they were "too technical" and because they used "too many tables".

Table 2.4.5.3. Informal scalable comparison table on the availability, accessibility and interfacing of CG tools (CGTNT refers to CoGITaNT, CGWrd to CGWorld, and Chrgr to Charger)

	Amine	CGTNT	CGWrd	ChrGr	Notio	WebKB
source code downloadable	?	+	?	?	?	+
for research purpose	+	+	+	+	+	+
with commercial license	?	?	?	?	?	-
with open-source license	?	+	?	?	?	-
with GPL	?	+	?	?	+	-
with dual licensing system	?	-	?	?	?	-
with MySQL-like licensing	?	-	?	?	?	-
Internet-accessible interface	-	+	?	-	-	+
1 freely accessible server	-	-	+	-	-	+
Web-accessible interface	-	+	+	-	-	+
CGI-like (ASP,Servlets,...)	-	?	+	-	-	+
GET parameters accepted	-	?	?	-	-	+
mostly HTML-based interface	-	?	+	_	-	+
mostly Java applets	-	-	-	-	-	-
via OKBC	-	-	-	-	-	-
via an XML-based protocol	-	?	-	-	-	-
inputs can be Web documents	?	?	?	?	?	+
via unsupervised Web spidering	?	?	?	?	?	-
via a directed search/download	?	?	?	?	?	+
mixing formal/informal content	?	?	?	?	?	+
with NLP/NLU performed	?	?	?	?	?	-
savable on the server machine	?	?	?	?	?	+
outputs can be documents	?	?	?	?	?	+
sent to the caller (browser)	?	?	?	?	?	+
savable on the server machine	?	?	?	?	?	+
graphs can be visualized in 2D	+	+	+	+	-	-
graphs can be edited in 2D	+	+	+	+	-	-
generated from linear format	+	?	?	?	?	-
API	+	+	+	+	+	+
C++ API	-	+	-	-	-	+
Java API	+	-	+	+	+	-
Lisp API	-	-	-	-	-	-
Prolog API	?	-	?	-	-	-
language of the code						
English-based source code	+	-	+	+	+	+
French-based source code	-	+	-	-	-	+
documentation						
documentation in English	+	~	+	+	+	+
documentation in French	-	+	-	-	-	-
language of the user interface						
user interface in English	+	+	+	+	+	+
user interface in French	-	-	-	-	-	-

Table 2.4.5.4. Informal scalable comparison table
on the handling of multiple users / sources / back-ends by CG tools

	Amine	CGTNT	CGWrd	ChrGr	Notio	WebKB
large shared persistent KB	-	-	?	-	-	+
concurrent editing	?	?	?	?	?	+
via KB locking during updates	-	-	-	-	-	+
via module locking	?	?	?	?	?	-
user/group ownership/edition	-	-	-	-	-	+
simple Unix-like system	-	-	-	-	-	-
special protocols	-	-	-	-	-	+
ontology filtering	?	?	?	?	?	+
composition of ontologies	-	-	-	-	-	-
comparison of ontologies	-	-	-	-	-	-
theory translation	-	-	-	-	-	-
different back-ends usable	-	-	-	-	-	~

3. Towards a *General Ontology* for Knowledge Representation, Sharing and Retrieval

3.1. A General Top-level Ontology of Concepts and Relations

3.1.1. Overview and Approach

The approach underlying the MSO follows three points.

- All categories are interesting to integrate but some are rather "arbitrary" in the sense that deciding whether something is of that category depends on personal preferences, goals or background knowledge. For example, the MSO integrates the top-level ontology of [Sowa 2002], one of the basis of which is C.S. Peirce's distinction between i) things that can be seen as "independent of any relationships to other entities" (e.g., a person), ii) things that can be seen as being "in a relationship to some other entities" (e.g., a spouse), and iii) things that "create a relationship to some other entities" (e.g., a marriage). The problem is that almost anything can be seen as being in relation with, or creating relations between, other entities. Thus, these categories are arbitrary in the above mentioned sense. The distinctions "physical vs. abstract", "changing vs. not changing", "divisible vs. indivisible" and "object_bearing_content vs. object_not_bearing_content" are similarly arbitrary and hence are not good choices for being the basis of an ontology (yet, SUMO is partially based on them).
  Amongst all top-level conceptual distinctions, the "situation vs. entity" distinction – which comes from Situation Semantics [Barwise & Perry, 1983] [Barwise et al., 1991] and is the basis of the top-level ontology of [Sowa, 1992] – seems the most intuitive or least arbitrary: most people similarly differentiates a situation (state or process, i.e., something that "occurs") from an entity (i.e., something that can be "involved" in a state or process but cannot directly occur). It is also one of the most general distinction: it is more general and intuitive than DOLCE's distinction between occurrents and perdurants; it is also closely related to – but not a generalization of – John Sowa's distinction between occurrents and continuants and Matthew West's distinction between 3D and 4D things.
  Between entities, the distinctions "spatial vs. non-spatial" is another general and intuitive distinction. The type pm#spatial_object generalizes dl#physical_endurant, sowa#object and cyc#information_bearing_thing while pm#spatial_object generalizes sumo#abstract.
  Here is a reminder of some direct subtype partitions of pm#thing.

```
pm#thing
 > {(pm#situation  pm#entity)}     {(pm#thing_playing_some_role  sowa#independent_thing)}
   {(4D#thing  3D#thing)}          {(sowa#continuant  sowa#occurrent)}
   {(sumo#physical sumo#abstract)} {(pm#indivisible_thing  pm#divisible_thing)}
   {(pm#individual pm#type)}       {(dolce#particular  dolce#universal  dolce#world)}
   {(cyc#partially_tangible cyc#intangible)} {(cyc#temporal_thing pm#non-temporal_thing)}
   {(cyc#partially_intangible cyc#tangible)} {(pm#domain pm#thing_that_is_not_a_domain)},
 pm#closed_exclusion:  owl#nothing,
 =  owl#thing  cyc#thing  akts#thing  sumo#entity  sowa#entity  rdfs#resource;
```

  The most general and least arbitrary types for organizing concept types also determine a good part of the organization of the relation types (see the next three tables) since organizing relation types according to their source or destination arguments is an efficient and non-arbitrary method. John Sowa proposes a classification of roles and relations [Sowa, 2001] (and in particular, case relations [Sowa, 2000b]) according to their ontological nature, first based on the above cited Peirce's distinctions. This classification has not been integrated into the MSO because classifying these relations according to their arguments is simpler, non-arbitrary and easier to search (e.g., see Table 3.1.3.3 to Table 3.1.3.5 for case relations and other "relations from situations").

- The integration of ontologies should be loss-less (as explained in Subsection 1.2.2) and hence the meaning or organization of the categories from the source ontologies should not be modified except for correcting semantic inconsistencies. In other words, re-categorization should be minimized. Some examples are given in the next subsection. Furthermore, as detailed in the previous chapter, the larger the MSO, and hence the more ontologies are tightly integrated into the MSO, the better for knowledge representation and sharing purposes. As explained in Paragraph 2.3.2.10 ("Connecting/adding to large ontologies"), the MSO can be seen as an alternative to other large ontologies but can also be seen as a complement to them since it interconnects their categories and hence can be seen as a useful resource for the Linking Open Data project.
- The lexical and structural normalization rules listed in Section 2.3 (and hence the shared KB edition protocols of Subsection 2.2.6) should be respected. This may imply adding names to categories of an integrated ontology or relations between these categories (in addition to connecting them to categories of other ontologies). Section 3.2 gives examples.
  The MSO includes various general ontologies (e.g., those of Sowa, DOLCE, LIS, NSM, AKT and a good part of the SUMO), language ontologies (e.g., KIF, OWL and XMLSchema) and well-known category lists (e.g., the relation types of the Dublin Core) but discourages the use of many categories from these ontologies so that the lexical and structural normalization rules are respected. Very few categories from OpenCYC are yet integrated to the MSO.

Table 3.1.1.1 and Table 3.1.1.2 give an overview of the content of the MSO. Table 3.1.1.3 shows the first specializations of pm#relation and hence shows various ways to categorize relation types. In current ontologies, relation types are rarely organized in specialization hierarchies, e.g., they are not much organized in DOLCE and the SUMO. In the MSO, they are organized because this eases their retrieval (and hence, their use) and avoids 'duplications'. To fully avoid such 'duplications', the relation type hierarchy should be entirely derived from the relation type hierarchy. This is possible (see Paragraph 2.1.1.14 and Subsection 4.2.13 for details) but this is not yet the case in the MSO.

The ***following subsections of Section 3.1*** present useful general types of concepts, some of their subtypes, and the types of relations using from/to such concepts.
The intertwining of the categories from various ontologies in the specialization hierarchy shows their tight integration, something that can only be done manually and that permits to better understand the meaning of each of these categories and their relationships (e.g., see Table 3.1.6.2).
After the presentation of many top-level categories specializing the distinction {(pm#situation pm#entity)} (subsections 3.2.1 to 3.1.11), other classifications are presented for concept types (Subsection 3.1.12) and for relation types (subsections 3.1.13 and 3.1.24).
As in the previous chapter, bold characters are used for the identifiers of categories that are specialized in a subsequent table. All the above mentioned direct subtype partitions of pm#thing are presented (as well as important relations from these subtypes) except for {(pm#domain pm#thing_that_is_not_a_domain)} which has been presented in Table 2.1.2.1 along with pm#domain_related_relation.

Section 3.2 presents the approach used for converting the noun-related part of WordNet 1.7 into a genuine lexical ontology (only the noun-related part has so far been integrated; from now on, this information will be left implicit).

After this conversion, to integrate WordNet into the MSO, the top-level WordNet categories were inserted into the top-level of the MSO. More precisely, the top-level categories of the MSO were specialized by all the relevant WordNet categories that could easily be found in WordNet, whichever their depth in the WordNet hierarchy. This is shown by the next subsections and some choices are explained. Table 3.1.1.4 shows the first two levels of the WordNet top-level categories for nouns. They are clearly very heterogeneous and unorganized.

Associating codes to certain top-level categories for speeding the comparison of any category inheriting these codes. Table 3.1.1.1 shows that three codes are associated to certain top-level categories (they are indicated within square brackets): a 4-character abbreviation, a 1-character abbreviation, and a binary code. The categories that have a code are those that can be considered the most useful for partitioning or organizing all the other categories. However, the categories that do not have a dot immediately before their codes are, for most purposes, too general to be used directly, they are essentially meant for organizational purposes.

These codes are meant to be inherited by the specializations of these categories in a ***depth first-order*** and only if they do not already have an associated code. The reason is that in Table 3.1.1.1, each category is first subtyped by two exclusive types and then possibly by other subtypes which are not exclusive with the first two subtypes. Thus, comparing the codes of two randomly selected pairs of categories in the MSO, may be sufficient to know if they are exclusive. This speeds up category matching and hence graph matching. When the codes are not sufficient to know if the two categories are exclusive, a path of subtype and/or exclusion relations between the two categories must still be found to check if they are comparable or not (this is a much shorter operation than comparing separate statements but this is a very frequent operation in graph matching and a much longer operation than comparing two codes).

With the *first two kinds of codes, the specialization and exclusion relations between the codes must be stored in a table* (e.g., an hash table). The binary code used below permits to avoid accessing such a table. This code should be read from right to left. The dots are only visual aids to distinguish pairs of bits. The rightmost bit permits to distinguish concept types (0) from relation types (1). *When a category with code X is subtyped, the first subtype has for code 01.X, the second subtype (exclusive with the first) has for code 10.X, and the other subtypes (not exclusive with the first two) have a code ending by 11.X (01.11.X for the first of these subtypes).* Thus, this is a scalable scheme even though not a concise one.

The ***Information Economy Meta Language*** [www-IEML, 2009] is a general ontology that

- associates a binary code to thousands of categories based on their decomposition into a dozen basic categories,
- is claimed to be extendable by anyone, by decomposing the meaning of any new category according to these basic categories and thus finding a code for this category according to the IEML encoding scheme.

If such a decomposition could actually be made in a non-arbitrary way, knowledge sharing, translation and retrieval would be much easier and there would be no need for aligning ontologies or no need for shared KBs: each ontology would simply define its categories in terms of shared basic categories. Unfortunately, the meaning of most categories is much too complex (and of a non-symbolic nature) to be defined by necessary and/or sufficient conditions with respect to their parents and siblings in a specialization hierarchy and hence cannot be defined with respect to basic concept types. (This does not mean that a relation type hierarchy cannot be kept small, especially if it does not 'duplicate' the concept type hierarchy. However, unless certain concept types can be used in relation nodes, trying to reduce the set of usable relation types too much, for example to a dozen relation types as in the knowledge graphs approach [Zhang, 2002], also leads to arbitrary decompositions and knowledge representation impoverishment.) The decompositions of categories in IEML are particularly arbitrary or non-intuitive; here are some examples:

```
salt      = relation from memory to language
sugar     = relation from language to memory
attribute = idea of a sensation towards a name
color     = idea of a name towards a sensation
red       = relation from life to language
green     = relation from language to life
blue      = relation from language to thought
```

On the other hand, mathematical or logical concept/relation types may be decomposed into a few primitives. [Schröder, 1899] decomposed these types into four primitives: `same', `and', `not' and `of'.

Table 3.1.1.1. Beginning of the Table of Content in the MSO input file "Top-level Ontology"

Table 3.1.1.2. Table of Content for Relations in the MSO input file "Top-level Ontology"

Table 3.1.1.3. Beginning of the Ontology of Relations in the MSO input file "Top-level Ontology"

```
pm#relation___related_thing___related_with .(*)
  definition: "type for any relation (unary, binary, ..., *-ary)",
  kind:  pm#relation_type,
  >  pm#relation_from/to_thing_of_common_kind  pm#relation_playing_a_special_role
     pm#wh-/how_relation  pm#relation_with_particular_property;

     pm#relation_from/to_thing_of_common_kind .(*)
       annotation: "type that permits to categorize relations according to their signatures
                    and hence offers i) a concise way to set essential exclusion relations,
                                     ii) a systematic and easy-to-follow categorization",
       >  {pm#relation_from_situation   pm#spatial_relation_from_entity_with_spatial_feature
           pm#relation_from_collection  pm#relation_from_description_content/medium/container
           pm#relation_from_attribute_or_quality_or_measure}
          {pm#relation_to_situation  pm#spatial_relation_to_entity_with_spatial_feature
           pm#relation_to_time  pm#relation_to_collection
           pm#relation_to_attribute_or_quality_or_measure};

     pm#relation_playing_a_special_role .(*)
       annotation: "this type permits to categorize relations according to their roles;
                    this is a traditional but quite subjective way of categorizing relations",
       > {pm#attributive_relation  pm#mereological_relation  pm#intentional_relation
          pm#temporal_relation  (pm#object_relation  > pm#object  pm#domain_object)
         }  dl#conceptual_relation  pm#relation_for_an_application ;
```

Table 3.1.1.4. The WordNet 1.7 top-level concept types for nouns

```
[_ parsing][pm#new_term  pm#default_creator:  wn];

human_action___act___human_activity
  >  action  non-accomplishment  leaning  assumption  rejection  forfeit
     {activity  inactivity}  wearing  judgment  production.human_action
     judgment  stay  residency  laughter  hindrance  stoppage
     group_action  distribution  permissive_waste  communicating  speech_act;

state
  >  skillfulness  cognitive_state  cleavage.state  medium.state
     condition  condition.state  conditionality  state_of_affairs
     relationship  relationship.state  tribalism.state  {utopia  dystopia}
     wild  isomerism  degree.state  office.state  status  {beingness  nonbeing}
     death.state  {employ  unemployment}  {order.state  disorder}  enmity  conflict.state
     illumination  freedom  representation.state  dependence  {motion  motionlessness}
     non-issue  {action.state  inaction.state}  temporary_state  imminence  preparedness
     kalemia  union.state  {matureness  immaturity}  state_of_grace  eternal_damnation
     omniscience  omnipotence  {flawlessness  imperfection}  unity  receivership.state
     ownership.state  end.state  sale.state  turgor  polyvalence;

event
  >  might-have-been  nonevent  happening  social_event  miracle.event  Fall;

phenomenon
  >  natural_phenomenon  levitation  metempsychosis  outcome  luck.phenomenon  luck  process;

entity
  >  self-contained_entity whole_thing living_thing cell causal_agent  holy_of_holies
     physical_object  location  depicted_object  unnamed_thing  imaginary_place
     anticipation  body_of_water  natural_enclosure  expanse  {inessential  essential}
     physical_part  sky  building_block  variable;

group___grouping
  >  arrangement  straggle  kingdom.group  biological_group  biotic_community
     human_race  people  social_group  aggregation  edition.group  electron_shell
     ethnic_group  race.group  association.group  subgroup  sainthood
     citizenry  population.group  masses  circuit  system  series.group;

possession
  >  belongings  territorial_dominion  white_elephant.possession  transferred_property
     circumstances  assets  treasure.possession  liabilities;

psychological_feature
  >  cognition  motivation  feeling;

abstraction
  >  time  space  attribute  relation  measure  set;
```

3.1.2. Minimizing re-categorization – Examples with DOLCE

Example 1. OntoClean/DOLCE [Guarino & Welty, 2002] distinguishes "qualities" (such as size, color, redness, smell and duration) from "regions/quale" (quality regions/spaces, i.e., categories of values for qualities, e.g., according to [Gangemi et al., 2002], wn#red, wn#past_times and wn#Greenwich_Mean_Time). Qualities and regions/quale are categorized under the exclusive types dolce#quality and dolce#region (a subtype of which is dolce#quale). However, in WordNet, categories for qualities or quale (about 8900 categories) are inter-related by specialization relations, e.g., wn#red specializes wn#chromatic_color and wn#color, while wn#past_times specializes wn#time. ***Specializing the exclusive types dolce#quality and dolce#region by WordNet categories***, as suggested in [Gangemi et al., 2002], ***is problematic***:

- this is a *re-categorization* since making a choice between these two supertypes changes the meaning of the WordNet categories; furthermore, this choice has to be made for most of the 8900 categories, *not simply for their most general categories* since "categories that could be interpreted as types for qualities" and "categories that could be interpreted as types for quale" specialize each other in WordNet;
- a great number of WordNet specialization relations have to be *broken*, hence this structuring is *lost*;
- it is often difficult to decide whether a WordNet category should be *interpreted* as a type for a quality or a quale; as opposed to [Gangemi et al., 2002], the MSO considers wn#Greenwich_Mean_Time, wn#work_time and wn#red as quality types; the authors of [Gangemi et al., 2002] provide arguments for the representation of types referring to the meaning of "adjectives for colors" as quale but not for the representation of types referring to the meaning of "nouns for colors" (e.g., wn#red___redness) as quale.

In the integration of WordNet into the MSO, relations were added or refined but *not removed or modified* – except for 306 of them (out of 74,488) in order to fix inconsistencies internal to WordNet. Hence, *the most general types of the above cited 8900 categories were not categorized as specializations of dolce#quality or dolce#region.* Instead, *these most general types, as well as dolce#quality and dolce#region, were generalized by* ***pm#attribute_or_quality_or_measure***. The name of this identifier is due to the fact that the WordNet categories for the things that are called "measures" in the MSO often specialize the WordNet categories that many people would interpret as "attributes", "characteristics" or "qualities" (as noted above, this is an over-interpretation when these last kinds of things are considered as exclusive with "measures"). This approach still permits the users of WebKB-2 to add supertype relations from any of the 8900 categories to either dolce#quality or dolce#region when this does not introduce inconsistencies, i.e., mostly from the leaves of the specialization hierarchy of these 8900 categories. As shown by the next FCG, the specialization relations between WordNet categories for attributes or measures permit simple and intuitive representations:

```
[a wn#car,  wn#color: some wn#red,  wn#weight: 900 wn#kg]
  //"900 kg" can be seen as a numerical quantification of wn#kg or as a way to
  //  specify a measure in wn#kg; Subsection 4.2.12 precises this point
```

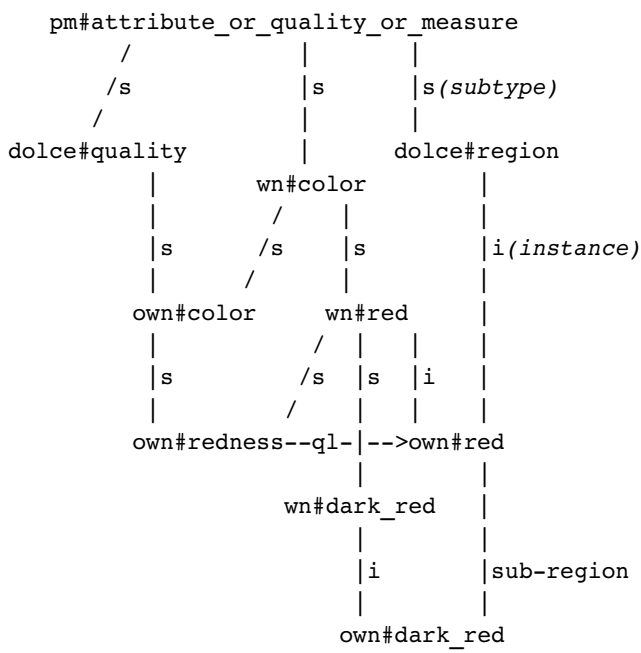
Concept types such as wn#color were also used as relations in Ontoseek [Guarino et al., 1999] who was created by the authors of OntoClean/DOLCE and [Gangemi et al., 2002]. If wn#red and wn#kg were not respectively specializing wn#color and wn#weight but solely categorized as quale (as in OntoWordNet [Gangemi et al., 2003]), more complex statements would have to be written, for example:

```
[a wn#car,  pm#attribute:  (a wn#weight, pm#measure: (a wn#measure, pm#unit: a wn#kg,
                                                                    pm#value: 900 wn#kg))
            pm#attribute:  (a wn#color,  pm#measure: (a wn#measure, pm#unit: a wn#Hertz,
                                                                    pm#value: a wn#red))].
```

Checking this graph would also be more complex and would require additional information on which categories are acceptable as measures for colors and weight. Finally, this approach avoids the introduction and use of relation types specific to a particular ontology (e.g., DOLCE introduces various relations for qualities and for quale), it leads to more normalized and sharable knowledge representations.

OntoWordNet (`own') is a re-interpretation and re-categorization of the top-level categories of WordNet as specializations of the types of DOLCE. OntoWordNet has not been integrated in the MSO but it could be. Table 3.1.2.1 illustrates this with some categories for colors.

Table 3.1.2.1. The way some types from the MSO and from OntoWordNet could be related



Example 2. In [Gangemi et al., 2002], dolce#amount_of_matter is exclusive with dolce#physical_object and subtyped by wn#substance. However, wn#substance has many subtypes which are *also* subtypes of dolce#physical_object. An example is the type wn#olive.relish which specializes wn#fruit (a wn#physical_object) and wn#relish (a wn#condiment and wn#substance). Another example is wn#glass_wool, subtype of wn#artifact (a wn#physical_object) and wn#insulator (a wn#substance). Since these specialization relations in WordNet are not clear mistakes, it seems that in [Gangemi et al., 2002], wn#substance has been over-interpreted (i.e., adapted) to fit the meaning of the type dolce#amount_of_matter. Instead, the MSO categorizes wn#substance (along with other types like wn#building_block and wn#physical_part) as subtypes of ***pm#physical_entity_part_or_substance*** which, like dolce#physical_object and dolce#amount_of_matter, is a direct subtype of pm#physical_entity. Since pm#physical_entity_part_or_substance covers both substances and "physical objects with unity", it may be seen as an adequate supertype for categorizing a "statue of clay". [Gangemi et al., 2002] discusses the dilemma its authors had for categorizing a "statue of clay" but does not discuss the drawbacks of using a type such as pm#physical_entity_part_or_substance. From a knowledge sharing perspective, this approach does not seem to have drawbacks, even for validation purposes. Indeed, as an example, a "statue of clay" can be seen as having the properties of both substances and physical objects. CYC does not hesitate to use similarly general types, e.g., cyc#partially_tangible and cyc#partially_intangible.

Conclusion: imposing precision is not always good for knowledge sharing. This subsection does not deny that the distinctions made by DOLCE and other top-level ontologies are important for knowledge validation purposes and because they lead people to represent knowledge in more precise ways. The precision of DOLCE categories and their associated constraints were intended to ease the automatic matching of categories from ontologies that are independently developed *but that re-use the DOLCE ontology*. However, a systematic use of precise categories (e.g., because general types such as pm#attribute_or_quality_or_measure and pm#physical_entity_part_or_substance are not proposed) is not good for knowledge sharing purposes since this sometimes leads to arbitrary choices in knowledge categorization or representation, and more complex statements that are more difficult to exploit.

3.1.3. General Categories for Situations (States and Processes)

Adoption of a more concise presentation. From now on, because of space constraints in tables for printing purposes, a more concise presentation is used for displaying information related to a category (this is not the presentation used in the input files for the MSO). Here is a list of changes compared to the presentations so far.

- Indentation is smaller.
- More relations are grouped (via the use of parenthesis).
- ***Categories useful for organization purposes only (e.g., pm#relation_from_situation) are declared/displayed only if a sufficient number of categories needs to be grouped (hence, the decomposition is not systematic).***
- The delimiters "(^" and "^)" are used for associating informal annotations to categories instead of using relations of type pm#annotation or pm#definition. An annotation sometimes contains several sentences but they are separated by '.' or ';' (hence; separate annotations can still be generated from them). Certain annotations are not displayed, e.g., the long annotations associated to certain categories from DOLCE or SUMO.
- The signature of a relation type is often not displayed when it is identical to the (most specialized) signature of the supertype(s).
- The creators of the relations are not shown; relations connecting categories from different ontologies are from `pm'.
- The last parsing directive allows WordNet categories to be left unprefixed. However, even though the "wn#" prefix is not used, the "#" is kept because it makes WordNet categories easier to spot.
- The abbreviation of pm#inverse ("-") is used.
- A "&" sign is used for connecting `dl' categories (DOLCE-Lite categories defined in OWL) to their `dolce' counterparts (DOLCE-Full categories defined in a world-based modal logics).
- The relations between some relation types and their "counterpart concept types usable in relation nodes" (e.g., between pm#agent and wn#agent as shown in Table 3.1.3.5) are only rarely shown and only when the concept types has many subtypes.

Table 3.1.3.1. Some general categories for situations

```
pm#situation (^something that "occurs" in a real/imaginary region of time and space^)
 > {(pm#state  pm#process)}  pm#situation_playing_some_role
   //some other categorizations for situations:
   sumo#process  dl#perdurant  sowa#process  sowa#participation
   sowa#situation___physical_mediating_occurrent___spatially_located_situation
   pm#situation_being_the_main_input/object/output_of_another_process;

   pm#state (^situation that is not a process^)
    > #state  #feeling  pm#state_playing_some_role;

   pm#process (^situation commonly considered as making a change during some period of time^)
    > pm#event  pm#problem_solving_process   pm#process_with_particular_feature
      #cognitive_process  #unconscious_process  #human_action  pm#process_playing_some_role
      pm#process_with_at_least_one_main_input/object/output,
    = aktp#Activity;

      pm#event (^process considered instantaneous from some viewpoint;
                classification under this category is application-dependent^);

      pm#problem_solving_process (^cognitive activity to solve a problem^)
       > pm/km#task  pm/km#problem_solving_method;

         pm/km#task (^processes modeled in knowledge acquisition, e.g., a diagnostic^)
          > (pm/km#real_life_task  > pm/km#knowledge_engineering
                                     kads#real_life_process_in_knowledge_acquisition);

      pm#process_with_particular_feature > pm#iterative_process  pm#never-ending_process;
         pm#iterative_process  > pm#iterative_never-ending_process;
         pm#never-ending_process  > pm#iterative_never-ending_process;

   pm#situation_playing_some_role (^e.g., a causal situation^)
     > pm#process_playing_some_role  dl#achievement  dl#accomplishment
       #resultant  #outcome  #evidence.situation;


pm#relation_from_situation .(pm#situation,*)
 > {pm#relation_from_situation_to_time  pm#relation_from_situation_to_situation}
   pm#case_relation  pm#within_group .(pm#situation,pm#collection);

pm#relation_to_situation .(*,pm#situation) (^prefer using relations "from" a situation^)
  > {pm#relation_from_time_to_situation .(pm#time_measure,pm#situation)
     pm#relation_from_situation_to_situation};
```

Table 3.1.3.2. Some other categorizations for situations

```
  sumo#process___physical_situation
    (^thing that happen and have temporal parts or stages; sumo#process is not identical
      to pm#situation because an instance of pm#situation is not necessarily an instance
      of sumo#physical^)
   > sumo#dual_object_process (^process requiring two nonidentical patients^)
     #state  #feeling  #cognitive_process  #unconscious_process  #human_action
     #phenomenon  #event;

  dl#perdurant___occurrence___PD   & dolce#perdurant,
    > {(dl#stative dl#event)},
    dl#p_spatial_location=> 0..* dl#space_region,
    dl#temporal_location=> 0..* dl#temporal_region,
    dl#happens_at=> 0..* dl#time_interval,
    dl#has_quality: 1..* dl#Temporal_location,
    dl#participant=> 1..* dl#endurant,
    dl#has_quality=> 0..* dl#temporal_quality,
    dl#part=> 0..* dl#perdurant,
    dl#constituted_by=> 0..* dl#perdurant;

      dl#stative___STV   & dolce#stative,  > {(dl#state dl#process)};
         dl#state___ST  & dolce#state;
         dl#process___PRO   & dolce#process;
      dl#event___EV  & dolce#event,  > {(dl#achievement dl#accomplishment)};
         dl#achievement___ACH  & dolce#achievement;
         dl#accomplishment___ACC  & dolce#accomplishment,  > dl#activity dl#phenomenon,
            dl#activity  > (dl#transaction  >  dl#commerce);
            dl#phenomenon  > dl#economic_process  dl#biological_process;
               dl#biological_process  dl#participant: 1..* dl#biological_object;
```

Table 3.1.3.3. Some types of relations from situations to time measures

```
pm#relation_from_situation_to_time .(pm#situation, pm#time_measure)
 > pm#time  pm#duration  pm#from_time  pm#until_time  pm#before_time;

  pm#time .(pm#situation,pm#time_measure) > pm#date .(pm#situation -> pm#time_measure);

  pm#from_time .(pm#situation -> pm#time_measure)
   > pm#departure_time .(pm#process -> pm#time_measure);

  pm#until_time___to_time .(pm#situation -> pm#time_measure)
   > pm#arrival_time .(pm#process -> pm#time_measure);
```

Table 3.1.3.4. Some types of relations between situations

```
pm#relation_from_situation_to_situation .(pm#situation,pm#situation)
 > pm#later_situation  pm#before_situation;

   pm#later_situation .(pm#situation,pm#situation)
    > pm#next_situation  pm#ending_situation  pm#postcondition  pm#consequence;

      pm#next_situation .(pm#situation -> pm#situation)  > pm#successor_situation;
      pm#postcondition .(pm#process,pm#situation);

   pm#before_situation .(pm#situation,pm#situation)
    > pm#previous_situation  pm#beginning_situation  pm#precondition  pm#cause;

      pm#beginning_situation .(pm#situation -> pm#situation)  – pm#ending_situation;
      pm#precondition .(pm#process,pm#situation)  – pm#postcondition;

      pm#cause .(pm#situation,pm#situation) – pm#consequence,
       > sumo#causes .(sumo#process,sumo#process);
```

Table 3.1.3.5. Case/thematic relations

```
pm#case_relation___thematic_relation .(pm#situation,*)
 > pm#doer/input/object/result/place
   (pm#experiencer .(pm#situation,pm#causal_entity)
     > sumo#experiencer .(sumo#process,dl#agentive_physical_object))
   pm#relation_from_process_only  pm#participant_in_Dolce_Lite;

   pm#doer/input/object/result/place .(pm#situation,?)
    > pm#doer/object/result
      pm#path_length .(pm#process -> pm#spatial_attribute_or_quality_or_measure)
      pm#place .(pm#situation -> pm#spatial_object);

      pm#doer/object/result .(pm#situation,?)
       > pm#agent  pm#initiator .(pm#situation,pm#causal_entity)  pm#object/result;

         pm#agent___doer .(pm#situation ?s [0..*], pm#entity ?e [1..*])
          >  sumo#agent .(sumo#process,dl#agentive_physical_object),
          :<=> [?s  #agent: ?e];  //#agent and its subtypes can be used instead of pm#agent
          //#agent can be used in relation nodes since:
          //  #agent <  (#causal_agent < pm#thing_playing_some_role);

         pm#input/object/result .(pm#situation, ?)
          > (pm#object___patient___theme .(pm#situation ?s, ?e) :<=> [?s  #object: ?e]
              > (pm#input .(pm#process,?)  > pm#material  pm#parameter)
                (pm#input_output .(pm#process,?)
                  > (pm#modified_object > pm#muted_object)
                    (pm#deleted_object  > pm#muted_object))
                (sumo#patient .(sumo#process,?)
                  > { (sumo#instrument < pm#instrument)
                      (sumo#result < pm#result)
                      sumo#resource .(sumo#process,sumo#object) }) )
            (pm#result .(pm#process ?p, ?e)  :<=> [?p  #resultant: ?e],
              > pm#output  sumo#result)
            (pm#instrument .(pm#situation ?s, pm#entity ?e) :<=> [?s  #instrument: ?e],
              > sumo#instrument .(sumo#process,sumo#object) ));

      pm#from/to .(pm#situation,pm#entity)
       > pm#source .(pm#situation,pm#causal_entity)
         (pm#recipient .(pm#situation,pm#entity)
           > pm#beneficiary .(pm#situation,pm#causal_entity))
         (pm#from/to_place .(pm#process -> pm#spatial_object)
           > pm#from_place  pm#to_place  pm#via_place  pm#path)
         sumo#origin___from .(sumo#process,sumo#object)
         sumo#destination___to .(sumo#process,pm#entity)
```

3.1.4. Organizing Processes w.r.t. their Inputs/Outputs

Processes can be organized according to many viewpoints but the most intuitive, systematic and scalable scheme is to base this organization on their *main* relations to other objects, especially their input/output relations. This scheme is similar to the organization of relation types based on their arguments. The emphasis on "main" refers to the fact that a process can have (many sub-processes that have) relations which are not important to the purpose, role or definition of that process. The next table shows general categories for organizing processes according to their main inputs-outputs. In this table, categories having identifiers in italic bold characters have been used for partitioning KM processes in Table 2.1.4.1. Their subtypes given in Table 2.1.4.1 are not repeated here.

Table 3.1.4.1. Organizing processes according to their main inputs-outputs

```
[_ parsing][pm#new_term  pm#default_creator:  pm];

process_with_at_least_one_main_input/object/output
 main input/object/result: 0..* thing,
 > {process_not_modifying_its_main_inputs  process_modifying_its_main_inputs}
   {process_creating_its_main_outputs  process_modifying_its_main_outputs}
   (process_with_a_main_input/object/output_of_a_particular_kind
     > (pm#process_with_a_process_as_main_input/object/output
         > pm#teaching_a_certain_process) )
   (process_being_the main_input/object/output_of_another_process
     < pm#situation_being_the main_input/object/output_of_another_process,
     > (pm#taught_process  object of:  pm#teaching_a_certain_process __[any->*,1..*<-any]) );

   process_not_modifying_its_main_inputs
    > {process_creating_its_main_outputs_without_modifying_its_main_inputs
       process_not_creating_nor_modifying_anything_using_its_main_inputs}
      {analyzing  synthesizing}  applying;

      process_creating_its_main_outputs_without_modifying_its_main_inputs
       > process_creating_copies/versions/selections_of_its_main_inputs
         process_associating_values_to_its_main_inputs;

         process_creating_copies/versions/selections_of_its_main_inputs
          > reading  sharing  retrieving  extracting  importing  exporting
            translating  federating
            interpreting   extrapolating;

         process_associating_values_to_its_main_inputs
          > annotating  evaluating  validating  comparing  analyzing;

   process_modifying_its_main_inputs
    > {process_creating_its_main_outputs_and_modifying_its_main_inputs
       process_modifying_its_main_outputs_and_modifying_its_main_inputs
       process_modifying_its_main_inputs_without_using_them_for_creating_its_main_outputs};

   process_creating_its_main_outputs
    > {process_creating_its_main_outputs_without_modifying_its_main_inputs
       process_creating_its_main_outputs_and_modifying_its_main_inputs};

   process_modifying_its_main_outputs
    > {process_modifying_its_main_outputs_without_modifying_its_main_inputs
       process_modifying_its_main_outputs_and_modifying_its_main_inputs};

[_ parsing][pm#new_term  pm#default_creator:  wn];
```

3.1.5. General Categories for Entities

Table 3.1.5.1. Some general categories for entities

```
pm#entity  (^something that can be "involved" in a situation^)
 > {(sumo#object  sumo#abstract)}  {(pm#spatial_object  pm#non-spatial_object)}
   dl#endurant  {(pm#indivisible_entity  pm#divisible_entity)}
   pm#entity_playing_some_role;

   sumo#object___entity_with_spatial_feature
    (^spatial object (space, location or physical object) or description medium/container
      (e.g., string, language, image); in a 4D ontology, an object is something whose
      spatio-temporal extent is thought of as dividing into spatial parts roughly parallel
      to the time-axis^)
    > {(pm#spatial_object  sumo#content_bearing_object)}
      sumo#self_connected_object  sumo#collection;

   pm#divisible_entity (^many classifications under this category are application-dependent^)
    > pm#collection  sumo#corpuscular_object  pm#amount_of_matter;
```

Table 3.1.5.2. Categorizing entities according to their roles

```
pm#entity_playing_some_role (^e.g., an agent, an owner^)
 > (pm#owned_entity  > #possession  {#good_point  #bad_point }  #creation_result)
   (pm#entity_part  > pm#physical_entity_part_or_substance)
   (pm#tool  > dl#artifact  #instrument.tool)
   (pm#process_recipient  > #recipient)
   (pm#process_object  > #depicted_object)
   pm#situation_result pm#causal_entity
   (pm#imaginary_entity (^an entity that has been imagined^)
     > (pm#imaginary_spatial_entity (^e.g., a cartoon character^)
         > #imaginary_place  #imaginary_being  #spiritual_being))
   {#essential #inessential}   #self-contained_entity  #anticipation  #representation
   #variable  #unnamed_thing  #holy_of_holies/*it is in the top-level of WordNet*/;

   pm#causal_entity___agent (^something (animal or software agent) able to act^)
    < #causal_agent,
    > pm#goal_directed_agent
      pm#perhaps_goal_directed_causal_entity (^e.g., supernatural forces^)
      pm#without_goal_causal_entity (^non conscious entity and not AI_Agent^)
      dl#agentive_physical_object  dl#agentive_social_object  #computer_code;

      pm#goal_directed_agent
       (^goal directed causal entity, e.g., a problem solver or an interactional agent^)
       > pm#cognitive_agent;

         pm#cognitive_agent (^e.g., an organization, an animal or an AI-agent^)
          > {(sumo#sentient_agent  pm#non-sentient_cognitive_agent)} aktp#Legal-Agent,
          = aktp#Generic-Agent;

            pm#non-sentient_cognitive_agent (^e.g., AI_Agent^)  > #social_group;
```

Table 3.1.5.3. DOLCE's endurants

```
dl#endurant___ED  & dolce#endurant,
 > {(dl#physical_endurant  dl#non-physical_endurant  dl#arbitrary_sum)},
 dl#temporary_part=> dl#endurant,  dl#temporary_part of=> dl#endurant,
 dl#temporary_proper_part=> dl#endurant,  dl#temporary_proper_part of=> dl#endurant,
 dl#e_temporal_location=> dl#temporal_region,  dl#present_at=> dl#time_interval,
 dl#depends_on=> dl#endurant,  dl#depends_on of=> dl#endurant,
 dl#descriptively_depends_on=> dl#non-physical_endurant,  dl#participant_in=> dl#perdurant;

   dl#non-physical_endurant___NPED  & dolce#non-physical_endurant,
    > dl#non-physical_object,
    dl#abstract_location=> dl#abstract_region,  dl#descriptively_depends_on of=> dl#endurant,
    dl#depends_on_spatial_location=> dl#space_region,  dl#part=> dl#non-physical_endurant,
    dl#physically_depends_on=> dl#physical_endurant,  dl#has_quality=> dl#abstract_quality,
    dl#constituted_by=> dl#non-physical_endurant;

      dl#non-physical_object___NPOB   & dolce#non-physical_object,
       > {dl#mental_object  dl#social_object };

         dl#mental_object___MOB (^e.g., a percept, a sense datum^)   & dolce#mental_object;

         dl#social_object___SOB   & dolce#social_object,
          > {(dl#agentive_social_object  dl#non-agentive_social_object)};

            dl#agentive_social_object___ASO   & dolce#agentive_social_object
             > {dl#social_agent  dl#society };

               dl#social_agent___SAG (^e.g., a legal person, a contractant^)
                > dl#socially_constructed_person  dl#social_unit,  & dolce#social_agent;

               dl#society___SC (^e.g., IBM, ONU^)  & dolce#society,  > dl#group;

            dl#non-agentive_social_object___NASO  & dolce#non-agentive_social_object,
             (^e.g., law, asset, currency, economic stuff^)
             > dl#information_object  dl#non-physical_place;

               dl#information_object
                dl#physically_depends_on: 1..* dl#PQ_or_PED_or_PR_or_PD;

               dl#non-physical_place  > dl#political_geographic_object,
                dl#physically_depends_on: 1..* dl#physical_place;

                  dl#political_geographic_object  > dl#country,
                   dl#physically_depends_on: 1..* dl#geographical_object;
```

Table 3.1.5.4. Categorization of entities according to their divisibility

```
sumo#self_connected_object (^object that does not consist of two or more disconnected parts^)
 > {sumo#substance  sumo#corpuscular_object}  sumo#transit_way  sumo#food;

   sumo#corpuscular_object___composite_entity___divisible_entity_with_discrete_parts
    (^self_connected_object whose parts have properties that are not shared by the whole^)
    > sumo#content_bearing_object  sumo#organic_object  sumo#artifact
       sowa#structure___physical_mediating_continuant  #whole_thing;

pm#divisible_entity (^many classifications under this category are application-dependent^)
 > pm#collection  sumo#corpuscular_object  pm#amount_of_matter;
```

3.1.6. General Categories for Spatial Objects (Including Physical Objects)

Table 3.1.6.1. General categories for spatial objects and space areas/regions

```
pm#spatial_object (^object which as a direct spatial location,
                    e.g., physical object, spatial region or shape^)
 > pm#space  dl#physical_endurant
   sowa#object___physical_independent_continuant___independent_spatial_object
   {#mathematical_point  pm#2D_object  pm#3D_object }  cyc#information_bearing_thing;

   pm#space (^point or extent in space^)
    > sumo#region (^topographic location^)
      dl#feature  dl#physical_place  //not dl#Spatial_location since it is a dl#quality
      #space  #location  #natural_enclosure  #expanse  #sky  #shape;

      dl#feature___F (^e.g., hole, gulf, boundary^)  & dolce#feature,
       > dl#dependent_place  dl#relevant_part,
       dl#host=> dl#entity;

      dl#physical_place  > dl#geographical_object;
```

Table 3.1.6.2. General categories for physical objects

```
dl#physical_endurant___PED  & dolce#physical_endurant,
 > {(pm#physical_entity  dl#feature)},
 dl#physical_location=> dl#physical_region,  dl#spatial_location=> dl#space_region,
 dl#physically_depends_on of=> dl#non-physical_endurant,  dl#part=> dl#physical_endurant,
 dl#constituted_by=> dl#physical_endurant,
 dl#has_quality=> 1..* dl#physical_quality;

   pm#physical_entity___object (^spatial entity made of matter^)
    > dl#physical_object  pm#physical_entity_part_or_substance;

      dl#physical_object___POB  & dolce#physical_object,
       > {(dl#agentive_physical_object  dl#non-agentive_physical_object)}
         (cyc#information_bearing_object  < cyc#information_bearing_thing),
       exclusion: sumo#substance;

         dl#agentive_physical_object___APO  & dolce#agentive_physical_object, = sumo#agent,
          > pm#entity_that_can_be_or_was_alive;

            pm#entity_that_can_be_or_was_alive (^e.g., an animal, a cell^)
             > (sumo#sentient_agent > (sumo#cognitive_agent > sumo#human)
               pm#living_physical_entity  #living_thing #cell;

                sumo#human  > {(sumo#man  sumo#woman)}  #person,  = dl#natural_person;
                //Figure 2.4.3.2 showed some universally quantified statements about wn#person

               pm#living_physical_entity (^alive physical entity^)
               //> #living.people, //no: people is a collection
                exclusion: pm#dead_physical_entity;
                   //because of the "undead" zombies, pm#closed_exclusion cannot be used

         dl#non-agentive_physical_object___NAPO  & dolce#non-agentive_physical_object,
          > pm#dead_physical_entity  #physical_object  (dl#artifact  = sumo#artifact)
            dl#biological_object  dl#physical_place  dl#unitary_collection;

            pm#dead_physical_entity (^physical_entity that is no more alive^)
             > pm#dead_person,  exclusion: pm#living_physical_entity;

            dl#biological_object (^an animal is not a biological_object, it is constituted_by
                                    a biological_object which is constituted_by some matter^)
             > dl#organism;

      pm#physical_entity_part_or_substance
       > dl#amount_of_matter   #substance  #physical_part  #building_block;

         dl#amount_of_matter___M (^similar to sumo#substance but need not be connected,
                                    e.g., "the amount of gold currently located in Europe"^)
          > sumo#substance,  & dolce#amount_of_matter;

            sumo#substance___amount_of_matter___divisible_entity_without_discrete_parts
             > {(sumo#synthetic_substance  sumo#natural_substance)}
               {(sumo#pure_substance  sumo#mixture)},
              exclusion: dl#physical_object;
```

Table 3.1.6.3. Some types of relations from entities with spatial features

```
pm#spatial_relation_from_entity_with_spatial_feature .(sumo#object,*)
 > pm#relation_from_spatial_object
   pm#relation_from_self_connected_object  pm#relation_between_objects
   sumo#distance  sumo#larger  sumo#smaller  sumo#connected  sumo#connects
   sumo#mereological_sum_fn  sumo#mereological_product_fn  sumo#mereological_difference_fn
   sumo#hole  sumo#hole_host_fn  sumo#partially_fills  sumo#hole_skin_fn  sumo#orientation;

   pm#relation_from_spatial_object .(pm#spatial_object,*)
    > pm#relation_to_another_spatial_object  pm#facility .(pm#spatial_object,?);

      pm#relation_to_another_spatial_object .(pm#spatial_object,pm#spatial_object)
       > pm#place  pm#spatial_part;

         pm#place .(pm#spatial_object,pm#spatial_object)
          > pm#address  (pm#above > pm#on, – pm#below)  pm#below
            (pm#near > pm#touching, – pm#far_from)  pm#far_from
            (pm#exterior___in  – pm#interior)  pm#interior  pm#before_location___before;

   pm#relation_from_self_connected_object .(sumo#self_connected_object, *)
    > (pm#function_between_self_connected_object
        .(sumo#self_connected_object -> sumo#self_connected_object)
       >  sumo#front_fn  sumo#back_fn)  sumo#member;

   pm#relation_between_objects .(sumo#object,sumo#object+)
    > sumo#traverses  sumo#crosses  sumo#penetrates  sumo#between  sumo#part;

      sumo#part___part_of .(sumo#object,sumo#object)
       (^this type should have been name sumo#part_of to respect the common graph reading
         convention for parameters; all other mereological relations are defined in terms
         of this one; this means that the 1st argument is part of the 2nd; note that,
         since part is a reflexive_relation, every object is a part of itself^)
       > sumo#proper_part___proper_part_of
         sumo#piece___piece_of .(sumo#substance,sumo#substance)
         sumo#component___component_of .(sumo#corpuscular_object,sumo#corpuscular_object)
         sumo#member .(sumo#self_connected_object,sumo#collection),
       exclusion: sumo#contains;


pm#spatial_relation_to_entity_with_spatial_feature .(*,sumo#object)
 (^prefer using relations "from" a spatial object^)
 > (pm#relation_to_spatial_object .(*,pm#spatial_object)
     > (pm#relation_to_another_spatial_object
        sumo#where_fn .(sumo#physical,sumo#time_point -> sumo#region) )
   (sumo#partly_located___partly_located_at .(sumo#physical,sumo#object)
     > (sumo#contains .(sumo#self_connected_object,sumo#object)  exclusion: sumo#part)
       (sumo#located .(sumo#physical,sumo#object)  > sumo#exactly_located) );
```

3.1.7. General Categories for Non-Spatial Objects

Table 3.1.7.1. General categories for non-spatial objects

```
pm#non-spatial_object (^abstraction or description content/medium/container
                       (a description medium that has some spatial feature is both
                       instance of sumo#object and pm#non-spatial_object^)
 > {(pm#attribute_or_quality_or_measure
     pm#non-spatial_object_that_is_not_an_attribute_or_quality_or_measure)}
   sumo#abstract   pm#non-spatial_object_playing_some_role;

   pm#non-spatial_object_that_is_not_an_attribute_or_quality_or_measure
    > pm#description_content/medium/container  pm#non-spatial_collection;
      //the above subtypes are not exclusive!

   sumo#abstract___entity_without_spatial_feature
    (^e.g., knowledge, motivation, measure; properties or qualities as distinguished
      from any particular embodiment of the properties/qualities in a physical medium;
      instances of sumo#abstract can be said to exist in the same sense as mathematical
      objects such as sets and relations, but they cannot exist at a particular place
      or time without some physical encoding or embodiment^)
    > {sumo#Attribute  pm#type}  sumo#quantity  pm#set_or_class  pm#psychological_entity
      sowa#abstract   dl#abstract  pm#description_content;

      sowa#abstract___abstract_entity
       (^distinguished from sumo#abstract because John Sowa does not seem to categorize
         all types of relations, sets, classes, attributes, quantities and graphs under
         this category; see http://www.jfsowa.com/ontology/toplevel.htm and
         http://www.jfsowa.com/ontology/roles.htm^)
       > sowa#form  sumo#proposition  sowa#intention;

      dl#abstract___abstraction___AB
       > {dl#fact dl#set dl#region },
       dl#has_quality=> 0 dl#Spatial_location,
       & dolce#abstract;

   pm#non-spatial_object_playing_some_role
    > (pm#psychological_entity (^feature/product of mental activity, e.g., feeling^)
        > (dl#mental_object > #sense_experience)  #psychological_feature
          pm#unit_of_measure_for_a_psychological_attribute;
```

3.1.8. General Categories for Temporal/Spatial/Physical/Psychological/... Attributes and Measures

Table 3.1.8.1. General concept types for attributes and measures

```
pm#attribute_or_quality_or_measure
 (^e.g., mass, mass unit, 1 kg, frequency, [2-3] hz, color, blue, speed, 1 m/s^)
 > pm#time_measure  pm#physical/spatial_attribute_or_quality_or_measure
   pm#process_attribute_or_quality_or_measure  pm#social_attribute_or_quality_or_measure
   pm#psychological_attribute_or_quality_or_measure
   pm#modality_measure  pm#numeric_attribute  pm#unit_of_measure
   {sumo#Attribute  sumo#quantity} //distinction generalizing {dl#quality  dl#region}
   #measure  #attribute  #property  #magnitude_relation;

   //subtypes of sumo#quantity and dl#region are given in Table 3.1.8.5
   sumo#Attribute
    > {( sumo#internal_attribute (^e.g., shape, color, fragility, etc^)
         (sumo#relational_attribute (^e.g., social roles and positional attributes^)
           > (sumo#normative_attribute (^attribute related to morality, legality, ...^)
               > {sumo#subjective_assessment_attribute  sumo#objective_norm} ) )
      )}
      (dl#quality___Q (^an attribute/dimension of something, e.g., its color,
                       but not the measure of this color^)  & dolce#quality,
        > {dl#temporal_quality  dl#physical_quality
           (dl#abstract_quality___AQ (^e.g., an asset value (not a measure of this value)^)
             dl#q_location=> dl#abstract_region,  dl#has_quality=> dl#abstract_quality,
             dl#inherent_in=> 1..* dl#non-physical_endurant,
             & dolce#abstract_quality) },
         dl#inherent_in=> 1..* dl#entity ,  dl#q_location=> 0..* dl#region,
         dl#has_quale=> 0..* dl#quale,  dl#has_quality=> 0..* dl#quality);

   sumo#quantity (^any specification of how many or how much of something there is^)
    > {pm#atomic_abstract_data_type
       (sumo#physical_quantity  = akts#Physical-Quantity,
         > {( (sumo#function_quantity
                > (sumo#unary_constant_function_quantity > sumo#time_dependent_quantity)
                  sumo#calorie  sumo#British_thermal_unit)
              (sumo#constant_quantity > sumo#time_measure)
           )}
           (dl#region___R (^the measure of an attribute/dimension^)  & dolce#region,
             > (dl#quale  dl#has_quale of=> dl#quality, dl#proper_part<= 0 dl#region)
               dl#space_region
               {dl#temporal_region  dl#physical_region
                (dl#abstract_region___AR (^e.g., 2 US dollars^)  & dolce#abstract_region,
                   dl#abstract_location of=> dl#non-physical_endurant,
                   dl#part=> dl#abstract_region) },
             dl#q_location of=> dl#quality,  dl#generic_location of=> dl#entity) ) };

   pm#unit_of_measure (^ Any kind of unit of measure, meter, dollar, kilogram, etc. ^)
    = akts#Unit-Of-Measure,
    > {pm#unit_of_time_duration  pm#physical/spatial_unit_of_measure
       pm#unit_of_measure_for_a_process  pm#unit_of_measure_for_a_psychological_attribute
       pm#unit_of_measure_for_a_social_attribute}  #unit_of_measurement;
```

Table 3.1.8.2. General relation types from/to attributes and measures

```
pm#relation_from_attribute_or_quality_or_measure .(pm#attribute_or_quality_or_measure,*)
 > pm#relation_to_another_attribute_or_quality_or_measure
   sumo#extension_fn .(sumo#Attribute -> sumo#class) (^a unary_function that maps an
            attribute into the class whose condition for membership is the attribute^);

   pm#relation_to_another_attribute_or_quality_or_measure
    .(pm#attribute_or_quality_or_measure,pm#attribute_or_quality_or_measure)
    > (pm#measure .(pm#attribute_or_quality_or_measure,pm#attribute_or_quality_or_measure)
        (^e.g., [a weight, measure: 75 kg]^))
      (dl#q_location .(dl#quality,dl#region)  – dl#q_location_of,
        > (dl#has_quale___ql .(dl#quality,dl#quale)  – dl#quale_of,  & dolce#ql));


pm#relation_to_attribute_or_quality_or_measure .(*,pm#attribute_or_quality_or_measure)
 > pm#attribute  pm#relation_to_another_attribute_or_quality_or_measure
   akts#has-unit-of-measure .(akts#Physical-Quantity, akts#Unit-Of-Measure);

   pm#attribute .(?,pm#attribute_or_quality_or_measure) (^e.g., [a car, attribute: a weight]^)
    > (sumo#property .(?,sumo#Attribute)
        > {sumo#attribute .(sumo#object,sumo#Attribute)
           sumo#manner .(sumo#process,sumo#Attribute) })
      (dl#qt___quality .(dl#quality_or_endurant_or_perdurant, dl#quality)
        > dl#direct_quality___dqt);
```

Table 3.1.8.3. General categories for temporal attributes and measures

```
pm#time_measure___temporal_attribute_or_quality_or_measure
 (^temporal duration or positions of time_points and time_intervals^)
 > sumo#time_measure  #time  #time_period;

   sumo#time_measure
    > dl#temporal_quality  dl#temporal_region;

      dl#temporal_quality___TQ (^e.g., a date or a duration, but not its measure/value^)
       > dl#Temporal_location,
       dl#q_location=> dl#temporal_region,  dl#has_quality=> dl#temporal_quality,
       dl#inherent_in=> 1..* dl#perdurant,  & dolce#temporal_quality;

         dl#Temporal_location___TL (^$(same normalization as dl#temporal_location)$ ^);

      dl#temporal_region___TR (^a value for a temporal quality, e.g., 24/5/2002, 2 seconds^)
       > dl#time_interval  sumo#time_position,
       dl#temporal_location of=> dl#perdurant,  dl#e_temporal_location of=> dl#endurant,
       dl#part=> dl#temporal_region,  & dolce#temporal_region;

         dl#time_interval___T
          > (pm#unit_of_time_duration (^any unit used to measure time^)  > #time_unit)
            (sumo#time_duration (^any measure of length of time^)  = akts#Duration)  dl#date,
          dl#time_of_presence_of=> dl#endurant,  dl#time_of_happening_of=> dl#perdurant,
          dl#time_of_q_presence_of=> dl#physical_quality,  & dolce#time_interval;

         sumo#time_position (^any time_point or time_interval^)  = akts#Time-Position,
          > {(sumo#time_interval  sumo#time_point)};


pm#relation_to_time .(*,pm#time_measure)
 > pm#relation_from_situation_to_time  pm#relation_from_time_to_time
   pm#relation_from_physical_entity_to_time .(pm#creation_date  > pm#first_release);

   pm#relation_from_time_to_time .(pm#time_measure,pm#time_measure)
    > pm#near_time  (pm#before  — pm#after)  pm#after;

   pm#relation_from_physical_entity_to_time .(pm#physical_entity -> pm#time_measure)
     > pm#check-in  pm#check-out;
```

Table 3.1.8.4. General categories for physical/spatial attributes and measures

```
pm#physical/spatial_attribute_or_quality_or_measure (^e.g., mass/length/color measure^)
 > dl#physical_quality  dl#physical_region
   pm#living_entity_attribute_or_quality_or_measure  menu#physical_characteristic
   pm#spatial_attribute_or_quality_or_measure
   pm#physical/spatial_unit_of_measure  pm#physical_process_attribute_or_quality_or_measure;

   dl#physical_quality___PQ (^e.g., weight, color, as attributes^)  & dolce#physical_quality,
    > dl#Spatial_location,
    dl#q_present_at=> dl#time_interval,  dl#q_location=> dl#physical_region,
    dl#has_quality=> dl#physical_quality,
    dl#inherent_in=> 1..* dl#physical_endurant;

   dl#physical_region___PR (^e.g., 2 meters, 3.5 kg, [2-3] hz^)  & dolce#physical_region,
    > dl#space_region dl#volume,  dl#physical_location of=> dl#physical_endurant,
    dl#part=> dl#physical_region;

      dl#space_region___S  & dolce#space_region,
       > dl#spatio_temporal_region,
       dl#spatial_location of=> 1..* dl#physical_endurant,
       dl#p_spatial_location of=> 0..* dl#perdurant,  dl#part=> 0..* dl#space_region,
       dl#depends_on_spatial_location of=> 0..* dl#non-physical_endurant;

   menu#physical_characteristic (^this category is for menu-generation purposes^)
    > {#visual_property #physical_property };

   pm#living_entity_attribute_or_quality_or_measure
    > pm#date_of_birth___D.O.B.___birthdate (^not a birthday^);

   pm#spatial_attribute_or_quality_or_measure (^e.g., length measure in meters^)
    > (dl#Spatial_location  & dolce#Spatial_location)
      (menu#spatial_characteristic (^this category is mainly for menu-generation purposes^)
        > {#width  #height  #length  #circumference  #diameter  #surface_area  #volume
           #thickness  #tenuity  #angle })
      (pm#spatial_unit_of_measure  > #linear_measure #angular_unit #area_unit #volume_unit);

   pm#physical/spatial_unit_of_measure
    > pm#spatial_unit_of_measure  #metric_unit
      (pm#unit_of_measure_for_a_physical_process
        > #acceleration_unit  #work_unit  #force_unit  #electromagnetic_unit
          #absorption_unit  #explosive_unit  #pressure_unit  #electromagnetic_unit);

   pm#physical_process_attribute_or_quality_or_measure
    > pm#unit_of_measure_for_a_physical_process
      (pm#energy_measure  > #work_unit  #electromagnetic_unit)
      (pm#absorption_measure  > #absorption_unit)
      (pm#radiation_measure  > pm#radioactivity_measure  #langley)
      (pm#explosion_measure  > #explosive_unit)  #game_point;
```

Table 3.1.8.5. Some other general categories for physical/spatial attributes and measures

```
pm#process_attribute_or_quality_or_measure (^e.g., a speed measure^)
 > pm#physical_process_attribute_or_quality_or_measure;

pm#social_attribute_or_quality_or_measure
 > (pm#identifier___id
     > {pm#passport_identifier  pm#social_welfare_identifier }  pm#serial_number
       (pm#URI___Uniform_Resource_Identifier
         > pm#URL___Uniform_Resource_Locator pm#URN___Uniform_Resource_Number))
   (pm#unit_of_measure_for_a_social_attribute  > #monetary_unit);

pm#psychological_attribute_or_quality_or_measure
 > pm#unit_of_measure_for_a_psychological_attribute;

pm#modality_measure (^e.g., pm#never is as an instance^)
 > (pm#temporal_modality_measure instance: pm#never  pm#rarely  pm#often  pm#always)
   pm#physical_possibility;
```

3.1.9. General Categories for Description Content/Mediums/Containers

Table 3.1.9.1. Some general categories for description content/mediums/containers

```
pm#description_content/medium/container
 > {pm#description_container  pm#description}  #communication;

   pm#description_container (^e.g., file, image, ... but not a disk or a piece of paper^)
    > (pm#document_element___document___DE  (^part of a document or whole document^)
        > #document);  //not #written_document, #document.communication, #record

   pm#description___information (^description (content/medium) of an entity or a situation^)
    > pm#description_content  pm#description_medium
      {(pm#data  pm#formal_or_semi-formal_well-formed_statement___knowledge)}
      (sowa#form___abstract_independent_thing
        > sowa#schema___abstract_independent_continuant
          sowa#script___abstract_independent_occurrent);

      pm#description_content___information (^e.g., a narration, an hypothesis^)
       = aktp#Abstract-Information,
       > sowa#proposition   sowa#intention  dl#fact  rdf#description
         pm#narration (^e.g., report, story, biography, etc.^)
         #subject_matter  #written_material  #code.laws  #public_knowledge  #cognitive_factor
         #perception.cognition  #cognitive_content  #history.cognition  #mental_attitude;

         sumo#proposition___abstract_relative_thing___situation_description
          (^piece of information; this piece of information may be represented by
            content_bearing_object(s) such as a string, a sound or an icon^)
          > sowa#description___abstract_relative_continuant
            sowa#history___abstract_relative_occurrent,
          = sumo#proposition;

         sowa#intention___abstract_mediating_thing
          > sowa#reason___abstract_mediating_continuant
            sowa#purpose___abstract_mediating_occurrent;
```

The five tables of Subsection 2.1.3 list the general categories for description mediums. They are not repeated here. Subsection 2.1.3 also explains i) why pm#description_medium was sub-typed instead of pm#description_content (mainly to categorize types such as pm#statement, pm#belief, pm#observation, pm#deduction, pm#assumption and pm#task_description), and ii) why this is not a very important modeling choice. The next table specializes pm#atomic_abstract_data_type and pm#structured_abstract_data_type, mainly with KIF and SUMO types.

Table 3.1.9.2. Some categories for abstract data types

```
pm#atomic_abstract_data_type
 > sumo#number  (pm#boolean instance: kif#true kif#false, = xmls#boolean);

   sumo#number  = kif#number,
    > {sumo#real_number  sumo#imaginary_number  (sumo#complex_number = kif#complex) };

      sumo#real_number  = kif#real owl#real,
       > {(sumo#rational_number sumo#irrational_number)}
         {( (sumo#nonnegative_real_number
              > (sumo#positive_real_number
                  > (sumo#positive_integer = xsd#positive_integer  xmls#non-negative_integer))
                (sumo#nonnegative_integer  = kif#natural  xmls#non-negative_integer,
                  > {(sumo#positive_integer  kif#zero)} ) )
            (sumo#negative_real_number
              > (sumo#negative_integer  = kif#negative  xsd#negative_integer)) )}
         {( (pm#positive_real_number  >  (xsd#positive_integer  = kif#positive))
            (pm#non-positive_real_number
              > (xsd#non-positive_integer > {(kif#zero  sumo#negative_integer)}) ) )}
         sumo#binary_number  xsd#decimal;

         sumo#rational_number  = kif#rational  owl#rational,
          > (sumo#integer  = kif#integer  xsd#integer,
              > {xsd#int xsd#short  xsd#byte
                 xsd#unsigned_long  xsd#unsigned_int  xsd#unsigned_short  xsd#unsigned_byte}
                {(sumo#nonnegative_integer  sumo#negative_integer)}
                {(xsd#non-positive_integer  sumo#positive_integer)}
                {( (sumo#even_integer = kif#even)  (sumo#odd_integer = kif#odd) )}
                sumo#prime_number);
         //all the atomic abstract data types of OWL 2 are listed above

pm#structured_abstract_data_type
 > sumo#list  pm#number_container  (pm#array  > pm#string)  pm#queue  pm#stack
   pm#keyed_collection_ADT  (pm#graph_ADT  > sumo#graph___connected_graph)
   wn#lattice  xmls#block_set  owl#data_range;

   sumo#list  = kif#list,
    > sumo#unique_list  kif#double___list_with_2_elements  kif#triple___list_with_3_elements
      (rdfs#container > {(rdf#bag rdf#seq rdf#alt)});

         rdf#seq > (rdf#list  = owl#collection,  > daml#disjoint___disjoint_list_of_classes);
         rdf#alt > pm#or_bag  pm#xor_bag;
```

Table 3.1.9.3. Some types of relations from description content/mediums/containers

```
pm#relation_from_description_content/medium/container
 .(pm#description_content/medium/container,*)
 > pm#relation_from_description
   (pm#relation_from_string .(pm#string,*)
      > (sumo#names .(pm#string,?)  > sumo#unique_identifier)
        (sumo#related_external_concept .(pm#string,?,sumo#language)
          > {sumo#synonymous_external_concept  sumo#subsumed_external_concept
             sumo#subsuming_external_concept}) )
   (pm#version .(pm#description_content/medium/container,
                 pm#description_content/medium/container)  > pm#ascii_content)
   pm#version_id .(pm#description_content/medium/container,pm#string)
   dc#Coverage  dc#Contributor  dc#Source  dc#Publisher  dc#Rights
   (pm#authoring_time .(pm#description_content/medium/container -> pm#time_measure)
     > pm#authoring_date)
   (pm#author .(pm#description,pm#causal_entity)  > pm#main_author pm#co-author)
   (dc#Language .(pm#description_content/medium/container -> pm#entity)
     > pm#language .(pm#description_content/medium/container -> pm#description_medium) )
   (dc#Format .(pm#description_content/medium/container -> pm#entity)
     > pm#format .(pm#description_content/medium/container ->
                   pm#description_content/medium/container) )
   (pm#description_instrument___descr_instrument .(pm#description_content/medium/container,
                                                   pm#description_medium)  > pm#language)
   (pm#description_object___descr_object .(pm#description_content/medium/container,?)
     – pm#descr)
   (pm#description_place .(pm#description_content/medium/container,
                           pm#description_content/medium/container)
   pm#physical_support___description_physical_support___descr_physical_support
     .(pm#description_content/medium/container,pm#physical_entity)
   pm#rhetorical_relation  pm#argumentation_relation;

   pm#relation_from_description .(pm#description,*)
    > pm#descr_container  pm#logical_relation  pm#truth_restricting_contextualizing_relation
      (pm#probability_relation .(sumo#formula,?)
        > sumo#probability_fn .(sumo#formula -> sumo#real_number)
          sumo#conditional_probability .(sumo#formula,sumo#formula,sumo#real_number)
          sumo#increases_likelihood___increases_likelihood_of .(sumo#formula,sumo#formula)
          sumo#decreases_likelihood___decreases_likelihood_of .(sumo#formula,sumo#formula)
          sumo#independent_probability .(sumo#formula,sumo#formula) );
      (pm#extended_specialization_from_statement .(pm#description, pm#description)
        > (pm#specializing_statement  pm#abbreviation: ".>",
            pm#inverse:  (pm#generalizing_statement  pm#abbreviation: "<") )
          (pm#corrective_specialization  >  pm#corrective_existential_specialization)
          pm#overriding_specialization);

      pm#logical_relation .(pm#description,pm#description)
       > pm#and
         (pm#truth_restricting_logical_relation .(pm#description,pm#description)
           > pm#or  pm#xor
             (pm#implication___logical_deduction___necessary_condition___if
               .(pm#description,pm#description)
               > sumo#entails .(sumo#formula,sumo#formula)   pm#generalizing_statement,
                 (pm#equivalence___iff  pm#abbreviation: "<=>"),
               pm#abbreviation: "=>",
               pm#inverse: (pm#sufficient_condition  pm#abbreviation: "<=") );
```

Table 3.1.9.4. Some types of truth-restricting contextualizing relations

```
     pm#truth_restricting_contextualizing_relation .(pm#description,*)
      > pm#truth_restricting_logical_relation
        pm#believer .(pm#description,pm#causal_entity);
        pm#modality .(pm#description,pm#modality_measure)  pm#overriding_specialization
        (pm#corrective_statement .(pm#description,pm#description)
          > (pm#corrective_specialization  – pm#corrective_generalization,
              > pm#corrective_existential_specialization  pm#extended_instantiation)
            (pm#corrective_generalization  > pm#corrective_existential_generalization)
            pm#corrective_reformulation  pm#correction);
```

Table 3.1.9.5. Some types of rhetorical and argumentation relations

```
pm#rhetorical_relation .(pm#description_content/medium/container,
                         pm#description_content/medium/container)
 (^main sources: the Rhetorical Structure Theory (RST) and the PENMAN ontology;
   DO NOT USE such fuzzy relations: instead, use relations from/to situations^)
 > (rst#presentational_rhetorical_relation
     > {rst#enablement  rst#background  rst#motivation  rst#evidence  rst#justify
        rst#antithesis  rst#concession });
   (rst#subject_matter_rhetorical_relation
     > {rst#circumstance  rst#solution
        (rst#elaboration
          > {rst#subtype  rst#instance  rst#specialization  rst#illustration  rst#subtask
             (rst#attributive_relation  > {rst#property rst#attribute rst#possession })
             rst#part})
        (rst#cause  > {rst#volitional_cause rst#non-volitional_cause rst#purpose })
        (rst#effect > {rst#volitional_result rst#non-volitional_result })
        rst#definition  rst#comparison  rst#means  rst#condition  rst#otherwise
        (rst#interpretation  > rst#evaluation)  rst#restatement  rst#summary  rst#theme
        (rst#contrast > rst#antithesis) })
   (rst#symmetric_rhetorical_relation  > {rst#restatement rst#contrast })
   pm#opposition  pm#negative_consequence;

pm#argumentation_relation .(pm#description_content/medium/container,
                            pm#description_content/medium/container)
 >  pm#answer  pm#contribution  pm#replacement  pm#confirmation  pm#reference
    (pm#argument > pm#weak_argument .(pm#strong_argument  > pm#proof)  pm#illustration
                   pm#argument_by_authority  pm#argument_by_popularity);
    (pm#contradiction  > pm#objection);
```

3.1.10. General Categories for Collections and Types

Table 3.1.10.1. General categories for collections

```
pm#collection  (^something gathering separated things (entities/situations)^)
 > {(pm#spatial_collection  pm#non-spatial_collection)}
   sumo#collection___physical_collection  pm#bag  pm#sequence  #group  #set;

   pm#spatial_collection
    > sumo#collection___physical_collection (^such a collection has members like a class,
         but unlike a class, it has a position in space-time and members can be added and
         subtracted without thereby changing the identity of the collection; some examples
         are toolkits, football teams, and flocks of sheep^);

   pm#non-spatial_collection___collection_of_categories_or_statements
    (^something gathering separated entities or situations and that is not a spatial object^)
    > pm#domain  pm#structured_ADT   pm#type  pm#set_or_class
      dl#arbitrary_sum___AS .(rdfs#constraint_resource  > rdfs#constraint_property);

      //See the tables and examples of Subsection 2.1.2 for direct relations from pm#domain,
      //specializations of pm#domain, and types of relations from pm#domain

      pm#set_or_class (^like sumo#set_or_class but including rdfs#class too^)
       > rdfs#class
         (sumo#set_or_class (^any instance of sumo#abstract that has elements or instances^)
           > {( (sumo#set___bag  > dl#set)  sumo#class)} );
```

As illustrated by the next table, in the MSO, relation type names from SUMO and KIF are complemented because the original names do not follow the graph-oriented reading convention for relations. For example, the next table includes the category identifiers kif#member___member_of and sumo#sub_list___sub_list_of. If relation creators were displayed (i.e., if a concise presentation had not been adopted), the next table would show the name relations as in the following statements.

```
kif#member  name: "member_of" __[pm];     sumo#sub_list  name: "sub_list_of" __[pm];
```

Such names were added only when problematic identifiers were not connected to un-problematic identifiers via an identity relation. Example in Table 3.1.10.2:

```
pm#relation_type  = sumo#relation;
```

Similarly, "bag" was added as a name to sumo#set (see the last line of previous table) since what SUMO call "sets" may contain duplicate elements. This is not the case for an instance of kif#set, i.e., what is referred to in this document with the word "set". Types (alias "classes") are like sets except that i) they are not assumed to be extensional (i.e., distinct classes may have exactly the same instances) and ii) they have implicit/explicit partial/total definitions. The type pm#collection is a supertype for all kinds of collections: extensional or not, with duplicate or not, ordered or not, etc.

Table 3.1.10.2. Some types of relations from collections

```
pm#relation_from_collection .(pm#collection,*)
 > pm#member  kif#nthrest .(kif#list,kif#natural -> kif#list)
   sumo#list_order_fn .(kif#list,sumo#positive_integer -> ?)
   sumo#list_length_fn .(kif#list -> sumo#nonnegative_integer)
   pm#relation_from_collection_to_number
   (pm#relation_from_an_ontology
     > (pm#relation_to_another_ontology .(pm#ontology,pm#ontology)
         > owl#backward_compatible_with owl#incompatible_with owl#prior_version) )
   pm#relation_between_collections  pm#relation_from_type;

   pm#member .(pm#collection,*)
    > (pm#domain_object .(pm#domain,pm#thing_that_is_not_a_domain)  > pm#core_domain_object)
      kif#member___member_of .(kif#set,?)
      rdf#li .(pm#collection,*)
      (pm#list_member .(kif#list,?)
        > (pm#item  — kif#item)  (kif#first > rdf#first .(rdf#list->?))
          kif#last  kif#butlast)
      kif#nth .(kif#list,kif#positive -> ?);

   pm#relation_from_collection_to_number .(pm#collection -> kif#number)
    > (pm#size___number_of_elements .(pm#collection -> kif#natural)
        > kif#length .(kif#list -> kif#natural) )
      pm#percentage  pm#minimal_size .(pm#collection -> kif#natural)
      pm#maximal_size .(pm#collection -> kif#natural)
      pm#average .(pm#number_container -> kif#number);

   pm#relation_between_collections .(pm#collection,pm#collection +)
    > (pm#sub_collection > { (pm#ending_collection > pm#final_segment)  kif#sublist }
      pm#relation_from_type_to_collection  pm#relation_to_another_set_or_class;

      (pm#sub_collection_of .(pm#collection,pm#collection)
        > sumo#sub_collection___sub_collection_of .(sumo#collection,sumo#collection)
          (sumo#sub_list___sub_list_of .(kif#list,kif#list)  > sumo#initial_list)
          (pm#final_segment_of  — pm#final_segment,  > kif#sublist___final_segment_of) )
      pm#overlapping_collection
      (pm#not_overlapping_collection  > pm#collection_complement)
      (pm#relation_between_kif_lists .(kif#list+, kif#list)
         > (pm#function_between_kif_lists .(kif#list+ -> kif#list)
             > (pm#mono_parameter_function_between_kif_lists .(kif#list -> kif#list)
                 >  kif#reverse  (kif#rest  > rdf#rest) )
               (pm#bi_parameter_function_between_kif_lists .(kif#list,kif#list -> kif#list)
                 > (kif#append =  sumo#list_concatenate_fn)  kif#revappend) ) );
```

Table 3.1.10.3. Some general second-order types

```
pm#type (^second-order type or more^)
 > rdfs#class  dolce#universal  {pm#1st_order_type  pm#2nd_order_type }
   {( pm#relation_type
      (pm#type_that_is_not_a_relation_type  > pm#concept_type  pm#2nd_order_type) )};


   rdfs#class (^rdfs#class has pm#binary_relation_type as instance and hence is
               different from sumo#class^)  = owl#class,
    > sumo#class,
    instance: pm#binary_relation_type;

      sumo#class
       > rdfs#datatype  owl#restriction  owl#all_different  owl#deprecated_class
         dl#rigid___RG (^"all" the instances of a rigid type must "necessarily" be of this
           type at all times; role types such as #student or pm#tired_person are "non-rigid"
           and even "anti-rigid" since it is always possible for "any" student or
           tired person to cease being student or tired without loosing its identity^);
         dl#leaf_type___L
         (dl#non-empty___NEP  > dl#strongly_non-empty_perdurant)
         pm#situation_class
         (pm#attribute_class  instance=> (any *a  supertype_or_equal: sumo#Attribute)
         (pm#substance_class  instance=> (any *s  supertype_or_equal: sumo#substance)
         (pm#virtual_class  > pm#virtual_relation_type),
       instance: pm#class_of_inheritable_relation_type  pm#thing  pm#nothing;

         pm#situation_class (^pm#situation and any subtype of it is instance of this class^)
          > (pm#sumo_process_class  instance: sumo#process)
            dl#strongly_non-empty_perdurant_class___NEP.S
              (^type of perdurant with at least two instances not related by a part relation^)
            dl#cumulative_perdurant_class___CM
              (^a "sum" of instances of this kind of perdurant is of this kind of perdurant^)
            dl#anticumulative_perdurant_class___CM~
              (^a "sum" of instances of this kind of perdurant is of this kind of perdurant^)
            dl#homeomerous_perdurant_class___HOM (^any part is of the same type^)
            dl#anti-homeomerous_perdurant_class___HOM~ (^no part is of the same type^)
            dl#atomic_perdurant_class___AT (^no perdurant of this kind has a proper part^)
            dl#anti-atomic_perdurant_class___AT~
              (^any perdurant of this kind has a proper part^),
          instance: (any *s  supertype_or_equal: pm#situation);
                    //any type *s which has for supertype_or_equal pm#situation

   dolce#universal___UNIVERSAL
    > (dolce#rigid_universal___X
        instance: //most categories of DOLCE are "rigid"; hence, they are not listed here
                  (dolce#particular  > is#instance_of_first_order_class) );

   pm#first_order_type___type1___type_for_all_1st_order_types
    (^all 1st order types are implicitly or explicitly instance of this 2nd-order type^)
    > {(pm#relation_type  pm#concept_type)};
```

All tables in this chapter are only meant to give an idea of how the MSO can organize types from many ontologies and which important types exist. This is even more true for the next table which omits many subtypes and provides almost no instances but gives an idea of what kinds of second-order types for relation types are important for KM and need to exist in the MSO since they exist in ontologies such as SUMO and OWL.

Table 3.1.10.4. Some general second-order types for relation types

```
pm#relation_type  = sumo#relation,
 > {(pm#predicate_type pm#function_type sumo#list)}  pm#single_valued_relation_type
   {(pm#total_valued_relation_type  pm#partial_valued_relation_type)}
   {pm#binary_relation_type  pm#ternary_relation_type  pm#quaternary_relation_type
    pm#quintary_relation_type  pm#variable_arity_relation_type }
   pm#many_to_many_relation_type  pm#many_to_one_relation_type  pm#one_to_many_relation_type
   pm#type_of_relation_extended_to_quantities  pm#probability_relation_type
   pm#spatial_relation_type  pm#temporal_relation_type  pm#intentional_relation_type,
 instance: (any *s  supertype_or_equal: pm#relation);

   pm#predicate_type (^a sentence-forming relation with each tuple being a finite,
                      ordered sequence of objects^)  = sumo#predicate,
    > pm#binary_predicate_type pm#ternary_predicate_type pm#quaternary_predicate_type
      pm#quintary_predicate_type,
    instance: sumo#disjoint_relation  sumo#contrary_attribute  sumo#exhaustive_attribute
              sumo#exhaustive_decomposition  sumo#disjoint_decomposition  sumo#partition
              sumo#holds;

   pm#single_valued_relation_type  = sumo#single_valued_relation,
    > (pm#function_type  = sumo#function,
        > (pm#continuous_function_type  = sumo#continuous_function,
            > pm#time_dependent_quantity_type)
          pm#function_quantity_type
          (pm#unary_function_type
            > pm#unary_constant_function_quantity_type
              (pm#one_to_one_function_type  > pm#sequence_function_type),
            = sumo#unary_function  owl#functional_property  daml#unique_property)
          (pm#binary_function_type  = sumo#binary_function,
            > pm#associative_function_type  pm#commutative_function_type,
          (pm#ternary_function_type  = sumo#ternary_function)
          (pm#quaternary_function_type  = sumo#quaternary_function);

   pm#binary_relation_type  = sumo#binary_relation  rdf#property  tap#property_type,
    > pm#unary_function_type  pm#binary_predicate_type  pm#injective_binary_relation_type
      pm#reflexive_relation_type  pm#irreflexive_relation_type  pm#symmetric_relation_type
      pm#trichotomizing_relation_type  pm#antisymmetric_relation_type
      pm#transitive_binary_relation_type  pm#intransitive_binary_relation_type
      owl#annotation_property  owl#deprecated_property
      (owl#ontology_property  instance: pm#relation_to_another_ontology)
      rdfs#constraint_property  (rdfs#container_membership_property  instance: rdfs#member)
      {owl#datatype_property  owl#object_property },
    instance: (any *s  supertype_or_equal: pm#binary_relation);

      pm#injective_binary_relation_type
       instance: pm#injective_binary_relation,
       = owl#inverse_functional_property  daml#unambiguous_property;

      pm#symmetric_relation_type  = owl#symmetric_property  sumo#symmetric_relation,
       > pm#equivalence_relation_type;

      pm#transitive_binary_relation_type  = sumo#transitive_relation  owl#transitive_property,
       > pm#equivalence_relation_type  pm#partial_ordering_relation_type,
       instance: pm#relation_instance_of_transitiveProperty_unless_directly_overrided;
                 //see Paragraph 2.1.1.19 for explanations abut this type
```

Table 3.1.10.5. Some types of relations from types

```
pm#relation_from_type .(pm#type,*)
 > (pm#type_specialization_or_equal .(pm#type, pm#formal_term)
     > (pm#instance  – pm#kind)
       (pm#subtype_or_equal .(pm#type,pm#type)  = dl#subsumes___SB,
         > (pm#subtype___strict_subtype  = dl#properly_subsumes___PSB,
             > (dl#properly_subsumes_leaf  < dl#subsumes_leaf),
             – pm#supertype)
           dl#subsumes_leaf
           (pm#same_type_as .(pm#type,pm#type) = dl#equal___EQ,
             > {owl#equivalent_class owl#equivalent_property}),
        – pm#supertype_or_equal),
     – pm#type_generalization_or_equal)
  (pm#supertype_or_equal .(pm#type, pm#type)
     > (pm#supertype___strict_supertype
          > rdfs#sub_class_of
            (sumo#subrelation .(pm#relation_type,pm#relation_type)  = cyc#genl_preds,
              > rdfs#sub_property_of .(pm#binary_relation_type,pm#binary_relation_type) ))
       pm#same_type_as)
  (pm#exclusive_type .(pm#type,pm#type) = dl#disjoint___DJ,
    > pm#exclusive_class
      (pm#closed_exclusion .(pm#type -> pm#type)
       > (pm#complement_type  = owl#complement_of,  > pm#complement_class)))
  (corresponding_relation_type .(pm#concept_type, pm#relation_type)
    > manually_set_corresponding_relation_type)
  (pm#relation_from_class .(rdfs#class,*)
    > pm#relation_from_class_to_collection
      sumo#abstraction_fn .(sumo#class -> sumo#Attribute)
      (pm#relation_from_sumo_process_class .(pm#sumo_process_class,*)
        > sumo#causes_subclass .(pm#sumo_process_class,pm#sumo_process_class)
          sumo#capability .(pm#sumo_process_class,pm#case_relation_type,sumo#object)
          sumo#has_skill .(pm#sumo_process_class,dl#agentive_physical_object))
      (pm#relation_from_attribute_type .(pm#attribute_class,*)
        > sumo#contrary_attribute .(pm#attribute_class,pm#attribute_class +)
          sumo#exhaustive_attribute .(pm#attribute_class,pm#attribute_class +))
      (pm#relation_from_restriction .(owl#restriction,*)
        > owl#on_property .(owl#restriction,pm#binary_relation_type)
          (owl#has_value .(owl#restriction,?)  = owl#to_value)
          (pm#relation_from_restriction_to_class .(owl#restriction,rdfs#class)
            > owl#all_values_from  owl#some_values_from daml#has_class_q)
          (pm#function_from_restriction_to_natural_number .(owl#restriction -> kif#natural)
            > owl#cardinality  daml#cardinality_q  owl#min_cardinality
              daml#min_cardinality_q  owl#max_cardinality  daml#max_cardinality_q) )
      pm#WordNet_object .(rdfs#class,?)
      pm#WordNet_noun_type .(rdfs#class,?) )
  sumo#material___material_type_of .(pm#substance_class,sumo#corpuscular_object)
  pm#relation_from_relation_type
  pm#relation_from_type_to_collection;
```

Table 3.1.10.6. Some types of relations from relation types

```
pm#relation_from_relation_type .(pm#relation_type,*)
 > (pm#relation_from_binary_relation_type .(pm#binary_relation_type,*)
     > (pm#relation_to_another_binary_relation_type
         .(pm#binary_relation_type,pm#binary_relation_type)
         > owl#equivalent_property  rdfs#sub_property_of
           (pm#inverse___reverse .(pm#binary_relation_type -> pm#binary_relation_type)
              = sumo#inverse  owl#inverse_of) )
      rdfs#domain .(pm#binary_relation_type,rdfs#class)
      rdfs#range .(pm#binary_relation_type,rdfs#class) )
   sumo#domain .(pm#relation_type,sumo#positive_integer,sumo#set_or_class)
   sumo#domain_subclass .(pm#relation_type,sumo#positive_integer,sumo#set_or_class)
   sumo#range .(pm#function_type,sumo#set_or_class)
   sumo#range_subclass .(pm#function_type,sumo#set_or_class)
   sumo#valence .(pm#relation_type,sumo#positive_integer)
   sumo#disjoint_relation .(pm#relation_type +)
   sumo#holds .(pm#relation_type,*)
   sumo#assignment_fn .(pm#function_type,*)
   sumo#distributes .(pm#binary_function_type,pm#binary_function_type);
```

Table 3.1.10.7. Some types of relations from types to collections

```
pm#relation_from_type_to_collection .(pm#type,pm#collection)
 > dl#partition___PT  pm#instances  pm#subtypes  pm#relation_from_class_to_collection;

   pm#relation_from_class_to_collection .(rdfs#class,pm#collection)
    > (pm#relation_from_class_to_list .(rdfs#class,rdf#list)
        > owl#union_of  owl#intersection_of  owl#one_of)
      owl#distinct_members .(owl#all_different,rdf#list)
      pm#relation_to_another_class;

      pm#relation_to_another_class .(rdfs#class,rdfs#class +)
       > (pm#binary_relation_to_another_class .(rdfs#class,rdfs#class)
           > rdfs#sub_class_of  owl#equivalent_class  pm#exclusive_class  daml#restricted_by)
         (sumo#disjoint_decomposition .(sumo#class,sumo#class +)  > sumo#partition)
         (sumo#exhaustive_decomposition .(sumo#class,sumo#class +)  > sumo#partition);

pm#relation_to_another_set_or_class .(pm#set_or_class,pm#set_or_class +)
 (^this category is needed to group SUMO relations between classes which cannot be subtype
   of pm#relation_from_type because their signatures curiously also involve collections^)
 > pm#disjoint  pm#subclass_of_or_equal
   sumo#power_set_fn .(sumo#set_or_class -> sumo#set_or_class)
   pm#relation_to_another_class;

   pm#disjoint .(pm#set_or_class,pm#set_or_class)
    > sumo#disjoint .(sumo#set_or_class,sumo#set_or_class)
      (pm#exclusive_class .(rdfs#class,rdfs#class)  = owl#disjoint_with,
        > (pm#complement_class  = owl#complement_of) );

   pm#subclass_of_or_equal .(pm#set_or_class,pm#set_or_class)
    > (sumo#subclass___subclass_of  > sumo#immediate_subclass___immediate_subclass_of)
      rdfs#sub_class_of;
```

3.1.11. Things w.r.t. to Their Roles

Table 3.1.11.1. Some general role types

```
pm#thing_playing_some_role (^category to classify things according to roles/viewpoints^)
 > (pm#thing_that_can_be_seen_as_a_relation
     > (pm#thing_that_can_be_seen_as_a_function
         > #employer  #seller  #price  #license;
           (pm#contact_point
             > pm#fax_No  pm#email_address  #address
               (pm#phone_No > {pm#mobile_phone_No  pm#home_phone_No  pm#business_phone_No})))
       pm#attribute_or_quality_or_measure
       pm#entity_playing_some_role  #relation #psychological_feature #information #facility)
   pm#situation_playing_some_role
   (pm#created_thing  > sumo#artifact #artifact #creation_result #resultant)
   (pm#processing_thing
     > pm#goal_directed_agent;
       (pm#process_or_process_description
         > pm#process  sumo#process #computer_code))
   #causal_agent  pm#error  pm#failure  pm#fault
   { (sowa#mediating_thing (^Peirce/Sowa's notion of "thirdness"^)
       > sowa#intention
         (sowa#nexus___physical_mediating_thing  > sowa#structure  sowa#situation) )
     (sowa#relative_thing (^Peirce/Sowa's notion of "secondness"^)
       > sowa#proposition;
         (sowa#prehension___physical_relative_thing
           > sowa#juncture___physical_relative_continuant
             sowa#participation___physical_relative_occurrent) )
   };
```

3.1.12. Some Other Categorizations For Things: Continuants/Occurrents, Divisible/Indivisible, ...

The next two tables show concept types that are not subtypes of pm#entity or pm#situation.

Table 3.1.12.1. Some additional types related to DOLCE

```
pm#individual___particular___supertype_of_1st_order_types (^all individuals (for concepts or
                          relations) are implicitly or explicitly instance of that type^)
 > {pm#spatial_object  pm#description_content  pm#description_container
    pm#attribute_or_quality_or_measure }
   dl#entity  dolce#entity  dolce#particular;

   dl#entity (^a category from DOLCE or generalized by a category from DOLCE^)
    > dl#atom  dl#abstract_or_perdurant  dl#quality_or_endurant_or_perdurant,
    = dl#particular___PT,
    dl#conceptual_relation=> 0..* dl#entity ,
    dl#has_quality=> 0..* dl#quality  ,
    dl#host of=> 0..* dl#feature,
    dl#generic_location=> 0..* dl#entity ,
    & dolce#entity  dolce#particular;
    //in dolce the endurant/perdurant distinction does not involve qualities

      dl#atom___At
       > (dl#temporary_atom   & dolce#temporary_atom,
            dl#temporary_proper_part<= 0 dl#entity,
            dl#temporary_atomic_part of=> 0..* dl#entity),
       dl#proper_part<= 0 dl#entity,  dl#atomic_part of=> 0..* dl#entity;

      dl#abstract_or_perdurant___AB_or_PD  > {(dl#perdurant dl#abstract)};

      dl#quality_or_endurant_or_perdurant___Q_or_ED_or_PD
       > {(dl#quality  (dl#endurant_or_perdurant___ED_or_PD  > {(dl#perdurant dl#endurant)})
         )}
         (dl#PQ_or_PED_or_PR_or_PD
           > {(dl#physical_quality dl#physical_endurant dl#physical_region dl#perdurant)} );
```

Table 3.1.12.2. Some additional top-level distinctions

```
pm#indivisible_thing (^thing with no proper part^)  > dl#atom  pm#indivisible_entity;
pm#divisible_thing  > pm#divisible_entity;


sowa#independent_thing (^Peirce/Sowa's notion of "firstness"^)
 > sowa#form  (sowa#actuality___physical_independent_thing  > sowa#object  sowa#process);

sowa#continuant___spatial_entity_with_time_independent_identity
 > sowa#object sowa#schema  sowa#juncture  sowa#description  sowa#structure  sowa#reason;

sowa#occurrent___thing_without_temporally_stable_identity
 > sowa#process  sowa#script  sowa#participation  sowa#history  sowa#situation  sowa#purpose;


3D#thing (^an object seen from a 3D (or endurantist) perspective, i.e. where a spatial entity
           may have a time independent identity, as opposed for example to the
          4D perspective where each spatial entity has an associated time frame^)
 > dl#endurant;
4D#thing (^an object seen from a 3D (or perdurantist) perspective^);


sumo#physical___physical_thing (^an entity that has a location in space-time;
        locations are themselves understood to have a location in space-time^)
 > {(sumo#object  sumo#process)}  sowa#actuality  sowa#prehension  sowa#nexus,
 = sowa#physical_thing;


cyc#partially_tangible (^A subcollection of cyc#SpatialThing-Localized and cyc#TemporalThing.
   Each instance of cyc#PartiallyTangible has a tangible (i.e. material) part and a temporal
   extent (i.e. it exists in time). It might or might not also have an intangible part.
   For example, a particular copy of a book is made of matter, has temporal extent, and also
   has an intangible part: the information content of the text markings on its pages.^)
 = akts#Tangible-Thing;

cyc#intangible (^The collection of things that are not physical -- are not made of, or
   encoded in, matter. Every cyc#Collection is a cyc#intangible (even if its instances
   are tangible), and so are some cyc#individuals.  Caution: do not confuse `tangibility'
   with `perceivability' -- humans can perceive light even though it's intangible --
   at least in a sense.^)  = akts#Intangible-Thing,
 > cyc#mathematical_or_computational_thing  cyc#intangible_individual  pm#non-spatial_object;

cyc#partially_intangible (^The collection of things that either are wholly intangible
   (see cyc#Intangible) or have at least one intangible (i.e. immaterial) part
   (see cyc#intangibleParts). This includes intangible individuals, such as instances of
   cyc#Number-General  or cyc#Agreement, as well as non-individuals (all of which are
   intangible), i.e. instances of cyc#SetOrCollection.  It also includes things that have
   both tangible and intangible components (see cyc#CompositeTangibleAndIntangibleObject),
   such as a printed copy of a newspaper (as its information content is intangible) or a
   person (as her mental states are intangible).^)
 > cyc#intangible;

cyc#tangible (^Something which is not intangible, something which is physical, made of matter.
   It does not matter whether things are real of imaginary. Therefore we consider
   Mickey Mouse's car and a hippogriff as tangible things^)
 > pm#physical_entity;
```

3.1.13. Categorization of Relations w.r.t. their Roles or Ontological Nature

Table 3.1.1.3 introduces pm#relation_playing_a_special_role, the supertype used for categorizing relation types according to their roles. Its direct subtypes are respectively specialized in each of the next tables. Since attributive/mereological/intentional relation types often 'duplicate' certain "concept types that can be used as relation nodes", there is often no point to declare or use such relation types. However, the subtypes of pm#relation_playing_a_special_role are sometimes handy to organize relation types from various ontologies, especially in the rare cases when these relation types could not be categorized elsewhere. This is the reason for the next tables. No attempt was made to be systematic in re-categorizing the relation types shown in all the previous tables under pm#relation_playing_a_special_role since doing so is rather arbitrary.

Table 3.1.13.1. Some attributive relation types

```
pm#attributive_relation .(*)
 > {rst#attributive_relation  (pm#purpose .(?,?)  > pm#goal rst#purpose)
    (pm#owner .(?, pm#causal_entity)  > pm#sole_owner,  —  pm#owner_of)
    (pm#owner_of .(pm#causal_entity, ?)
      > sumo#possesses .(dl#agentive_physical_object,sumo#object)
        sumo#property_fn .(dl#agentive_physical_object -> sumo#set) )
    sumo#leader .(sumo#human,dl#agentive_physical_object)
    (pm#creator .(pm#entity,pm#entity)  > (dc#Creator  > pm#author) )
    sumo#exploits .(sumo#object,dl#agentive_physical_object)
    sumo#has_purpose .(sumo#physical,sumo#formula)
    sumo#has_purpose_for_agent .(sumo#physical,sumo#formula,sumo#cognitive_agent)
    pm#measure  pm#attribute
    (pm#name .(?,?)
      > (dc#Title___title .(? -> pm#entity) = tap#title)
        (dc#Identifier___identifier .(? -> pm#string)  > dl#identifier)
        (rdfs#label .(?,pm#string)  > dl#name  akts#has-pretty-name  akts#has-variant-name)
        tap#plural .(?,pm#string)  tap#singular .(?,pm#string) )
    (dc#Date___date .(? -> pm#entity)  > pm#date pm#authoring_time pm#publish_date)
    rdf#value .(?,?)
    pm#rdf_reification_relation .(?,?)
     > (rdf#predicate .(rdf#statement -> pm#binary_relation_type)
       rdf#subject .(rdf#statement -> ?)   rdf#object .(? -> ?)
    pm#support .(?,?)
   };
```

Table 3.1.13.2. Some mereological relation types

```
pm#mereological_relation .(?,*)
 > pm#binary_mereological_relation  pm#part_in_Dolce_Full;

  pm#binary_mereological_relation  .(?,?)  //see Paragraph 2.1.1.17 for explanations
   pm#relation_source:  (*x  pm#kind:  *t __[.<->?]) __[.->?],
   pm#relation_destination:  (*x  pm#kind:  *t __[.<->?]) __[.->?];
   > pm#direct_part_or_equal  pm#part_or_equal  pm#part_or_equal_of
     (pm#overlap_with .(?,?)
       > dl#overlap___O .(dl#abstract_or_perdurant,dl#abstract_or_perdurant) );

     pm#direct_part_or_equal .(?,?)
      > (pm#direct_part .(?x,?y) := [?x pm#direct_part_or_equal: (?y != ?x)]);

     pm#part_or_equal .(?,?)  – pm#part_or_equal_of,
      < (pm#relation_instance_of_transitiveProperty_unless_directly_overrided
          kind: owl#transitive_property,
          type:  pm#type_instance_of_a_certain_second_order_type_unless_directly_overrided),
      > sumo#part
        (pm#part
          := [ [?x direct_part ?y] or: [?x part: ?y] ];  //or:
        //:= [?x (direct_part: a description_medium)* direct_part: ?y],
          > {pm#sub_situation  pm#spatial_part  pm#sub-attribute  pm#sub-description
             pm#subdomain}  pm#sub_collection  pm#parts .(?, pm#collection);
             pm#part_in_Dolce_Lite);

           pm#sub_situation .(pm#situation,pm#situation)
            > pm#sub_process  pm#substate
              dl#temporal_part___P.T .(dl#perdurant,dl#perdurant)
              dl#spatial_part___P.S .(dl#perdurant,dl#perdurant);

           pm#spatial_part .(pm#spatial_object, pm#spatial_object)
            > {(pm#physical_sub-area .(pm#physical_entity, pm#physical_entity)
                pm#non-physical_sub-area .(pm#spatial_object, pm#spatial_object) )}
              (pm#physical_part .(pm#physical_entity, pm#physical_entity)
                >  pm#matter___stuff
                   (pm#physical_sub-area
                     >  pm#attached_physical_component  pm#removed_physical_piece) );

           pm#part_in_Dolce_Lite .(dl#entity,dl#entity)
            > (dl#part (dl#entity,dl#entity)   – (dl#part_of < pm#part_or_equal_of),
                > (dl#component .(dl#entity,dl#entity)  – dl#component_of)
                  (dl#atomic_part .(dl#entity,dl#atom)  – dl#atomic_part_of,
                    > (dl#temporary_atomic_part___AtP .(dl#entity,dl#atom)
                        – dl#temporary_atomic_part_of) )
              (dl#proper_part .(dl#entity,dl#entity)  – dl#proper_part_of)
              (dl#temporary_proper_part .(dl#endurant,dl#endurant)
                – dl#temporary_proper_part_of,
                & dolce#temporary_proper_part)
              (dl#temporary_part .(dl#endurant,dl#endurant)  – dl#temporary_part_of,
                > (dl#temporary_component  – dl#temporary_component_of) )
              dl#constant_part
              (dl#sibling_part .(dl#entity,dl#entity)  – dl#sibling_part);
```

Table 3.1.13.3. Part relation types in Dolce Full

```
  pm#part_in_Dolce_Full .(dolce#world,dolce#particular,dolce#particular +)
   > (dolce#part___P .(dolce#world,dolce#particular,dolce#particular,dolce#particular)
       & dl#part_of)
     (dolce#atomic_part___AtP .(dolce#world,dolce#particular,dolce#particular)
       & dl#atomic_part_of)
     (dolce#temporary_atomic_part___AtP .(dolce#world,dolce#particular,dolce#particular,
                                          dolce#particular)  & dl#temporary_atomic_part_of)
     (dolce#proper_part___PP .(dolce#world,dolce#particular,dolce#particular)
       & dl#proper_part_of)
     (dolce#temporary_proper_part___PP .(dolce#world,dolce#particular,dolce#particular,
                                        dolce#particular) & dl#temporary_proper_part_of)
     (dolce#temporary_part___P .(dolce#world,dolce#particular,dolce#particular,
                                 dolce#particular)  & dl#temporary_part_of) );
```

Table 3.1.13.4. Some intentional relation types

```
pm#intentional_relation .(sumo#cognitive_agent,?)
 (^relations between an agent and one or more entities, where the relation requires that
   the agent has awareness of the entity^)
 > sumo#prefers .(sumo#cognitive_agent,sumo#formula,sumo#formula)
   sumo#in_scope_of_interest .(sumo#cognitive_agent,?)
   (pm#propositional_attitude_relation .(sumo#cognitive_agent,sumo#formula)
     >  sumo#desires .(sumo#cognitive_agent,sumo#formula)
        sumo#considers .(sumo#cognitive_agent,sumo#formula)
        sumo#believes .(sumo#cognitive_agent,sumo#formula)
        sumo#knows .(sumo#cognitive_agent,sumo#formula) )
   pm#object_attitude_relation .(sumo#cognitive_agent,sumo#physical)
    (^intentional_relations where the agent has awareness of an instance of sumo#physical^)
    > sumo#needs .(sumo#cognitive_agent,sumo#physical)
      sumo#wants .(sumo#cognitive_agent,sumo#physical);
```

Table 3.1.13.5. Some temporal relation types

```
pm#temporal_relation .(?,?)
 > pm#relation_from_time_to_situation  pm#relation_to_time
   sumo#time .(sumo#physical,sumo#time_position)
   sumo#temporal_part  sumo#begin_fn  sumo#end_fn  sumo#starts  sumo#finishes  sumo#before
   sumo#before_or_equal  sumo#temporally_between  sumo#temporally_between_or_equal
   sumo#overlaps_temporally  sumo#meets_temporally  sumo#earlier  sumo#cooccur
   sumo#time_interval_fn  sumo#recurrent_time_interval_fn
   sumo#when_fn .(sumo#physical -> sumo#time_interval)  sumo#past_fn  sumo#immediate_past_fn
   sumo#future_fn  sumo#immediate_future_fn
   sumo#year_fn  sumo#month_fn  sumo#day_fn  sumo#hour_fn  sumo#minute_fn  sumo#second_fn
   sumo#temporal_composition_fn  sumo#relative_time_fn
   sumo#holds_during .(sumo#time_position,sumo#formula);
```

Table 3.1.13.6. Some conceptual relation types

```
dl#conceptual_relation .(dl#entity,dl#entity)  — dl#conceptual_relation,
 > (dl#immediate_relation .(dl#entity,dl#entity)
    > dl#q_location  dl#part  dl#participant
      (dl#inherent_in .(dl#quality,dl#entity)  — dl#has_quality)
      (dl#constituted_by___substance___K .(dl#entity,dl#entity)  — dl#constitutes,
        > (dl#has_member .(dl#entity,dl#entity)  — dl#member_of),
        & dolce#constitution) )
  (dl#mediated_relation .(dl#entity,dl#entity)  — dl#mediated_relation,
    > dl#sibling_part dl#proper_part
      (dl#present_at .(dl#endurant,dl#time_interval)  — dl#time_of_presence_of)
      (dl#q_present_at .(dl#physical_quality,dl#time_interval)  — dl#time_of_q_presence_of)
      (dl#happens_at .(dl#perdurant,dl#time_interval)  — dl#time_of_happening_of)
      (dl#overlaps .(dl#entity,dl#entity)  — dl#overlaps)
      (dl#generic_location .(dl#entity,dl#entity)  — dl#generic_location_of,
        > (dl#exact_location .(dl#entity,dl#region)  — dl#exact_location_of,
            > (dl#location .(dl#entity,dl#region)  — dl#location_of,
                > (dl#physical_location .(dl#physical_endurant,dl#physical_region)
                    > (dl#spatial_location .(dl#physical_endurant,dl#space_region)
                        — dl#spatial_location_of) )
                  (dl#temporal_location .(dl#perdurant,dl#temporal_region)
                    > dl#duration,  — dl#temporal_location_of)
                  (dl#duration .(dl#perdurant,dl#temporal_region)  — dl#duration_of)
              (dl#e_temporal_location .(dl#endurant,dl#temporal_region)
                — dl#e_temporal_location)
              (dl#p_spatial_location .(dl#perdurant,dl#space_region)
                — dl#p_spatial_location_of)
              (dl#abstract_location .(dl#non-physical_endurant,dl#abstract_region)
                — dl#abstract_location_of)
              (dl#depends_on_spatial_location .(dl#non-physical_endurant,dl#space_region)
                — dl#depend_on_spatial_location_of) ) )
      (dl#depends_on .(dl#endurant,dl#quality_or_endurant_or_perdurant)  — dl#depend_on_of,
        > (dl#physically_depends_on .(dl#endurant,dl#PQ_or_PED_or_PR_or_PD)
            — dl#physical_depend_on_of)
          (dl#descriptively_depends_on .(dl#endurant,dl#non-physical_endurant)
            — dl#descriptive_depend_on_of) )
      (dl#host .(dl#feature,dl#entity)  — dl#host_of) );
```

3.1.14. Categorization of Relations w.r.t. What/Who/Why/.../How Questions

It is tempting to think that information can be intuitively and systematically acquired and organized based on who/what/why/where/when/how questions. Several research works in knowledge acquisition have attempted to do so and published a lists of generally one or two dozens of general template questions [Shadbolt & Burton, 1995] [Liu, 2005]. However, as the next table shows, these kinds of questions i) refer to very different types of relations and hence do not permit to differentiate them, ii) do not permit to refer to many relations compared to the number of basic relations that have been presented above for each important conceptual distinction (e.g., the relations from a situation; Table 3.1.3.3 to Table 3.1.3.5), and iii) do not really permit to organize these relations. To conclude, in order to systematically elicit, search and organize knowledge, it seems more efficient to use the hierarchy of concept and relation types of the tables 3.1.3.3, 3.1.3.4, 3.1.3.5, 3.1.6.3, 3.1.8.3, 3.1.8.4 and 3.1.8.5 than general template questions. Given the number of relation types that the next table indirectly refers to (since some of the types listed here have many subtypes listed in previous tables), this section may give the largest collection of relation types related to who/what/why/where/when/how questions that has been published so far.

Table 3.1.14.1. Some relations related to who/what/why/where/when/how questions

```
pm#wh-/how_relation .(*)  (^this type permits to categorize relations based on
 who/what/why/where/when/how questions; this is a subjective and ineffective way of
 categorizing relations^)
 > pm#who_relation  pm#what_relation  pm#why_relation  pm#where_relation  pm#when_relation
   pm#how_relation;

   pm#who_relation
    > pm#agent  pm#initiator  pm#experiencer  pm#owner  pm#generator  pm#creator;

   pm#what_relation
    > pm#object/result  pm#process_attribute  pm#mereological_relation  pm#method
      pm#relation_from_collection  pm#relation_to_collection  pm#contextualizing_relation;

   pm#why_relation
    > pm#cause  pm#consequence  pm#method  pm#goal  pm#triggering_event  pm#ending_event
      pm#precondition  pm#postcondition  pm#purpose;

   pm#where_relation (^where, from/to where, ...^)
    > pm#from/to  pm#place  pm#path_length  pm#within_group
      pm#relation_to_another_spatial_object  pm#spatial_origin;

   pm#when_relation .(?,?)
    > {pm#relation_to_time  pm#relation_from_situation_to_situation  pm#temporal_relation };

   pm#how_relation
    > pm#instrument  pm#method  pm#sub_process
      (pm#how_much_relation
        > pm#duration  pm#relation_to_attribute_or_quality_or_measure
          pm#relation_from_collection_to_number);
```

3.2. Integrating WordNet-like Resources

Need for a comprehensive lexical ontology. People representing knowledge in a shared KB cannot be asked to declaring and categorize or define *most of* the terms they use. Indeed, this task would be too time-consuming and people would not do it in a correct way for knowledge sharing purposes or, more probably, would not do it at all. A "lexical ontology" for English or another natural language should *at least* be provided to ease, check and guide knowledge entering and permit knowledge sharing and retrieval. A lexical ontology connects the words of a language to the categories representing the main meaning of these words, and connects these categories via some specialization relations and only a few other kinds of conceptual relations (typically, some mereological relations): unlike in foundational ontologies, no more complex definitions are generally provided. However, for important categories such as the top-level ones and the often used ones, "schemas" (definitions or universally quantified statements) representing all the relations that are commonly used from instances of these important categories, are also necessary: they are useful for Natural Language Understanding or the generation of combinable menus that guide and normalize knowledge representation.

The many wordnets. WordNet (the main English wordnet, since its inception [Miller, 1995]) is a lexical database that connects English words to "synonym sets" (each "synset" officially represents the shared meaning of the words in the set and, actually, one of their shared meanings) and organizes the synsets by semantic relations, e.g., specialization and partOf relations. Because of these relations, since its first version, WordNet has been (and increasingly is) interpreted and exploited as a lexical ontology (i.e., a set of categories connected by relations having a formal semantics) despite its shortcomings for this purpose. For example, "WordNet as a lexical ontology" has been used for purposes such as precision-oriented information retrieval [Mihalcea & Moldovan, 2000], query expansion and answering [Smeaton et al, 1995] [Kwok et al., 2001], support for machine translation (creation of the Sensus ontology [Knight & Luk, 1994]), and knowledge representation, sharing or brokering [Guarino et al., 1999] [www-AI-Trader, 2003]. Because of this success, WordNet has been extended (e.g., by the "eXtended WordNet" project) and emulated: the "Global WordNet" project attempts to coordinate the production and linking of "wordnets" for all languages (e.g., wordnets from the EuroWordNet, BalkaNet and MultiWordNet projects) [http://wordnet.princeton.edu/wordnet/related-projects/]. Hence, methods exploiting or extending one wordnet may be adapted and re-used on other ones.

Need for many ontological additions and corrections. WordNet has not been built for knowledge representation purposes, nor apparently according to basic "taxonomy building principles" and with consistency checking tools. As noted in [Gangemi et al., 2002], types and individuals are not distinguished, the annotation of a category is not to be relied on as it may be contradicted by specializations of this category, direct specializations of categories often have heterogeneous levels of generality, role types (e.g. wn#student) are not distinguished from natural types e.g. wn#person) and may generalize them. The integration of WordNet into the MSO also permitted to find that in WordNet i) specialization relations are sometimes used where "location" or "similar" relations should be used, ii) the "part" and "member" relations between types are not used in a consistent way (most of these relations seem to mean that all instances of the source type have for part/member at least one instance of the destination type, but this is not *always* the case), iii) some of these transitive relations are redundant (and there were even a few directed cycles in previous versions of WordNet), iv) exclusion relations (the rare constraints that WordNet provides to check its taxonomy) are sometimes broken, i.e. some exclusive categories have common specializations, and v) there are lexical problems in its annotations. Furthermore, WordNet has no intuitive (short) category identifiers. Table 3.1.1.4 shows the WordNet 1.7 top-level concept types for nouns. Even for the top levels, the lack of structure is clear.

The integration of WordNet into the MSO. By 2010, this integration was still, by far, the work that detected, corrected and published the largest number of errors in WordNet: 362 lexical corrections, 338 semantic corrections and 160 domain-independent specializations of a WordNet category (in addition to more than 3000 specializations of WordNet categories that were created for specific applications or domains, e.g., for information technology related domains). These corrections are published in a format that allows them to be automatically performed in a subsequent version of WordNet [Martin, 2003c]. OntoWordNet [Gangemi et al., 2003] seems to be the second largest ontological work on WordNet (although "based on WordNet" is a more exact expression since the corrections cannot be integrated back into WordNet). The integration of WordNet into the MSO is also the only work on WordNet to have

- isolated thousands of individuals (about 6200, even though using individuals instead of types was avoided),
- generated intuitive category identifiers,
- changed the meaning of the WordNet categories only for solving inconsistencies internal to WordNet, and
- permitted Web users to further extend and correct this ontology. No attempt was made to bring more structure to the whole of WordNet, as this would probably take many years of work.

This section introduces extensions and corrections of the noun-related part of WordNet 1.7 that are needed to transform it into a lexical ontology usable for knowledge-based applications and, especially, the *manual* representation of natural language sentences. Much more would be needed to support natural language parsing. No claim is made that the resulting ontology is sufficient to support the inter-operation of fully automatic software agents, e.g., for e-commerce or database integration purposes. [Colomb, 1997] shows that such inter-operations have strong requirements and, in the general case, are not likely to be fully supported by ontologies anytime soon. Conversely, this work may be re-used to enhance brokering applications such as metadata registries and Yellow-pages like catalogs.

Availability in various formats. The input files for the MSO are currently in FL and FCG. The input files for the top-level and WordNet related part of the MSO are stored in files that respectively are 0.3Mb long and 10.3Mb long. These two files were also translated into a 14.1Mb CGIF file, a 35Mb DAML+OIL compliant RDF file, and in two files using the WordNet format (a 12.9Mb file named "data.noun" and a 4.3Mb file named "index.noun"). These files are accessible from [Martin, 2003c]. User-defined parts of the MSO are also available in various formats (currently, FL, FCG, FE, CGIF and RDF) by issuing queries.
Whenever some statements or relations cannot be exported into the selected destination KRL (because this KRL is not expressive enough or the export procedure is not fully implemented), a translation in FL or FCG is given within comments. To avoid doing this too often, ad-hoc translations are used, especially when exporting to RDF+OWL/XML. However, none of this can be done with the WordNet format. Indeed, it does not support comments (since in data.noun the WordNet internal identifier for a category is its byte offset in this file) nor other relation types than the WordNet ones (e.g., there is no "place" relation; there is also no quantifiers nor context). Hence, only WordNet kinds of knowledge is exported when the WordNet format is used. Since WordNet is currently the most re-used lexical/ontological resource and since the offset-based WordNet format is neither readable nor expressive, and less efficient to handle than a database with any DBMS, it would be good for knowledge sharing purposes if WordNet-like projects used a language such as FL and an (object oriented) DBMS based architecture as in WebKB-2.

3.2.1. Generating Intuitive Identifiers for WordNet Categories

WordNet has at least two internal identifiers for each category, e.g., the category for "Friday" has for identifiers '12558316' and 'Friday%1:28:00::'. While some applications re-use them, others (such as [Gangemi et al., 2002]) generated their own identifiers by concatenating names or using suffixes, e.g., 'Inessential$Nonessential' and 'Cell_1'. However, for knowledge representation, exchange and sharing purposes (within KBs or on the Web), category identifiers should permit concise and clear knowledge representations, including via *controlled languages*. Hence, for these purposes, each category should have at least one identifier composed of a common but rather unambiguous

word or expression, and *as little else as possible*. This means that one of the category names should be used as "key name" (the name used in the identifier), if possible with no suffix. Its capitalization should not be modified, in order to ease its use in controlled languages and avoid the addition of another name for specifying the exact capitalization. Other reasons are given in Section 2.3.1 ("Lexical Normalization").

In WordNet, the most common name for a category is supposed to be the first in its synset. This information is not very reliable and less ambiguous names may appear after the first. When one of the other names is a compound name beginning or ending with the first name (e.g., "Steve_Martin" in one of the synsets having "Martin" as first name), it constitutes a better choice for a key name than the first name. Hence, here is the first set of rules (ordered by decreasing order of priority) that were used for generating key names.

- When the first name of a category begins or ends with one of the other names, select this other name as key name, unless it is shared by another category that has no generated key name yet.
- Select the first name of a category as key, unless it is shared by another category that has no generated key name yet.
- Try the first two rules on the second name instead of the first.
- Try the first two rules on the third name instead of the first.
- ...

To respect the decreasing order of priority on these rules, the knowledge base (KB) was scanned many times (each time, testing all remaining categories without a key name) and allowed the test of a lower priority rule only when the application of rules of greater priority did not lead to any more change. (The order of the rules was also respected when testing each category). This simple approach was efficient enough given that WebKB-2 could scan the whole KB quite quickly (0.45 second in average). The application of the first two rules (i.e., trying to use only the first name of each category) permitted the assignment of key names to 75% of categories (56,074 out of 74,488). The use of the other rules, i.e., of the other names, permitted the assignment of key names to 84% of categories. This means that in the remaining 16% categories, each of them had *all* its names in common with another category.

Hence, to go further, suffixes had to be generated. When WordNet 1.6 was integrated in an older version of the MSO, numbers were used as suffixes, but the use of such categories in knowledge statements proved not user-friendly enough. A better option was to use the key name of the first generalization. Such a suffix often helps people guess the meaning of a category without having to access its generalizations. However, giving a key name with a suffix to *all* remaining unassigned categories would also have been cumbersome. Hence, the following rules (by decreasing order of priority and with a lower priority than the previous rules) were used to select the categories to which key names with a suffix would be assigned.

- Select the category with a frequency-of-use number *far lower* than those in other categories sharing all the same names. This number is provided by WordNet and represents the frequency of occurrence of the category in a some concordance documents; it is an indication but not of great importance. The value for "far lower" was first set to 30 and then to decreasing values.
- Select the category with a far lower number of subtypes than the other categories sharing all the same names.

More precisely, the implementation of these last two rules used combinations of gradually decreasing values of frequency-of-use and number of subtypes. Furthermore, the assignment of suffixes to subtypes of wn#action was penalized since these types are more frequently used than others in knowledge statements.

After several more scans of the KB with all the rules, there were still a few dozens of unassigned categories. To fix this, more precise names were sometimes manually added to these categories, or names were sometimes re-ordered. Some suffix attributions and key name choices were also manually corrected. For example, in application of the first rule, "Republic_of_Singapore" had been selected instead of "Singapore" as key name, but wn#Singapore is a more convenient identifier; it also seems that the island of Singapore and the capital of Singapore are better referred via wn#Singapore.island and wn#Singapore.capital than via wn#Singapore. To fix such problems, before re-running the whole "key name assignment" procedure from scratch, suffixes were semi-automatically attached to many key names, especially for the specializations of the category wn#location. For example, the suffixes ".capital", ".city", ".island",

".country" and ".colony" made many key names unambiguous. Sometimes, instead of using the generalizing category for the suffix, the partOf relation was used. As an illustration, in WordNet 1.7, since wn#town has three categories with only name "Bangor" which are part of different regions, the following identifiers was respectively given to these three categories: wn#Bangor.Wales, wn#Bangor.Northern_Ireland and wn#Bangor.Maine.

Thus, *only* 5944 WordNet categories have been given a key name with a suffix. The list of these categories is accessible from [Martin, 2003c], i.e., http://www.webkb.org/doc/wn/.

This method of generating unambiguous and readable identifiers for WordNet could be re-used on some other linguistic ontologies.

3.2.2. Distinguishing Types from Individuals

Distinguishing types from individuals (instances of first order types) is important for knowledge representation, inferencing and checking since individuals cannot have subtypes or instances. Certain individuals, often called continuants or endurants [Gangemi et al., 2002], can change in time without being viewed as different individuals (i.e. without loosing their identity), e.g., individuals for persons or cities. Specializing such individuals according to time (e.g., pm#Paris_in_1995) might be tempting (and is possible using an extended specialization relation) but can be avoided by using contexts on statements.

Distinguishing types from individuals is not always obvious. For example, [Gangemi et al., 2002] asserts that the WordNet category wn#karate (a specialization of wn#activity) should be an individual. However, this is neither a good nor largely shared choice since there are various ways to practice karate and since each individual practice of karate may be considered as an instance. In accordance with one of the normalization rules of Subsection 2.3.2, anything which may be subtyped, or has various occurrences, or comes in different variants or versions should be represented as a type, rather than an individual; otherwise, knowledge representation possibilities are reduced. For example, any doctrine, book, language, alphabetic character, code, diploma, sport or recurring situation should rather be represented as a type.

[Martin, 2003c] lists the 6211 individuals that were manually isolated: typically, time periods, persons, organizations, places and battles. To do so, all WordNet specialization relations were first translated as subtype relations, within an input file. Then, since WordNet categories are grouped by theme within the WordNet database files and since the input file for WordNet keeps this organization, a careful but relatively quick "search and replace" of subtype relations into instance relations was made in the zones where individuals could appear. This work was checked via the extraction and examination of all direct relations from categories having a name with a capitalized first letter, as in the following example.

```
wn#Neolithic_Age
  kind: wn#time_period,
  part of: wn#Stone_Age;  //according to WordNet (nowadays, scientists would not assert this)
```

3.2.3. Correcting Lexical and Semantic Problems

By 2005, ***362 lexical corrections*** to WordNet 1.7 were made: 29 modifications of category annotations, 262 additions of category names and 77 manual re-orderings of category names. The corrections were documented in a (usually not displayed) part of the category annotations, a part which is isolated via the delimiters "$(" and ")$". The same format/phrasing was used for writing all these sub-annotations. Below are two examples for each of the above cited three kinds of lexical corrections; in each example, the identifier between the '#' and '|' characters refer to one of the WordNet identifiers for the category.

```
wn#wn06603188|Department_of_Energy___Energy_Department___Energy
  (^ $("in USA" added in this annotation)$  the federal department responsible for
    maintaining a national energy policy; in USA, created in 1977^);
wn#wn00017399|feeling
  (^ $("the psychological feature of" replaced by "the state of" in this annotation;
       '< #psychological_feature' replaced by '< pm#state')$
    state of experiencing affective and emotional states; "he had a feeling of euphoria"^);
wn#wn00028506|sword_play___play
  (^ $("sword_play" added as key name)$  the act using a sword (or other weapon)
    vigorously and skillfully^);
wn#wn00131140|resolution.action
  (^ $(".action" added as key name extension)$  a decision to do something or to
    behave in a certain manner; "he always wrote down his New Year's resolutions"^);
wn#wn02487798|jalopy___bus___heap
  (^ $("jalopy" set as key name)$  a car that is old and unreliable;
    "the fenders had fallen off that old bus"^);
#wn00022035|possession
  (^ $("possession" set as key name)$  anything owned or possessed^);
```

The second example shows that two changes are documented: a lexical one and a semantic one. The last two examples show how the manual selection of certain names to be key names is recorded to be taken into account by the procedure of key name assignment which must for example be executed when a new version of WordNet has to be integrated (this is not needed if this procedure changes none of the key names of the categories already existing in the old version but only assigns key names for the new categories). The list of all lexical and semantic corrections or additions is accessible from [Martin, 2003c].

By 2005, ***338 semantic corrections*** to WordNet 1.7 were made: 120 removals of relations between categories, 146 modifications of types of relations between categories, and 72 modifications of relation destinations. Table 3.2.3.1 shows two examples for each of these three kinds of corrections. Out of the 338 corrected relations, 41 were redundant and about 230 were inconsistent with other relations. While some inconsistencies were manually detected, most of them were detected by WebKB-2 because exclusion relations at the top-level of the MSO were violated. For example, some categories in WordNet were classified as both

- human action *and* either causal agent, instrument or result of action (this was for example the case of wn#relaxant and wn#interpretation),
- human action *and* communication medium/content (e.g., this was the case of wn#epilog and wn#thanksgiving),
- communication medium/content *and* either a physical entity (e.g., wn#book_jacket) or an attribute (e.g. wn#academic_degree).

Some specialization relations in WordNet were also used where "member" relations should have been used, e.g., between categories for species and genus of species. Similarly, WordNet does not have 'location', 'similar' and 'identity' relations, and hence uses subtype relations instead of

- 'location' relations (e.g., many city/regions where battles occurred were categorized as both city/regions and battles),
- 'similar' relations (e.g., for a Greek god and its Roman counterpart), and
- 'identity' relations (WordNet sometimes introduces a different category to represent obsolete names).

For each transitive relation type in WordNet ('specialization', 'part' and 'member'), WebKB-2 automatically detected redundancies in its related hierarchy. WebKB-2 also exploited the combination of exclusion and specialization

relations to detect inconsistencies or redundancies. However, no other combination was exploited. Hence, more inconsistencies could be detected. For example, the rule "if t2 specializes t1, and t1 is member of t0, then t2 is member of t0" could be exploited to detect more redundant relations; this would for example permit to detect that not only wn#dog but also its subtype wn#hound_dog are related to wn#pack.animal_group by a member_of relation. Negative constraints such as "if t2 specializes t1, then t2 cannot be linked by any other kind of relation to t1" could also be used. However, it does not seem that WordNet has many problems of this kind.

Table 3.2.3.1. Examples of semantic corrections

```
//Examples of removals of relations between categories
wn#wn12347769|Payne's_gray
  (^ $('< wn#blue' removed since wn#blue is exclusive with wn#pigment)$
    any pigment that produces a grayish to dark grayish blue^)
  < wn#pigment;  //'< wn#pigment  wn#blue' was/is in the source WordNet
wn#wn00918115|Actium.naval_battle
  (^ $('< #town' removed since a battle is not a town)$
    the naval battle in which Antony and Cleopatra were defeated by Octavian's fleet
    under Agrippa in 31 BC^)
  kind: wn#naval_battle;

//Examples of modifications of the types of relations between categories
wn#wn07130190|Anglia
  (^ $('< wn#England' replaced by '= wn#England')$  the Latin name for England^)
  = wn#England;
wn#wn07799755|Mancunian
  (^ $('< Manchester' replaced by 'place: wn#Manchester')$  a resident of Manchester^)
  < wn#English_person,
  place: wn#Manchester;

//Examples of modifications of destinations of relations between categories
wn#wn05168522|transmission
  (^ $('< wn#communicating' replaced by '< wn#communication' since the subtypes of
       wn#transmission indicate that it represents a transmission medium, not a process)$
    communication by means of transmitted signals^)
  < wn#communication;
wn#wn02665816|creation_result___creation
  (^$('< #artifact' replaced by '< pm#created_thing' since some subtypes are not physical;
      "creation_result" added as key name)$  an artifact brought into existence by someone^)
  <  pm#created_thing
```

3.2.4. Making Some Domain-independent Additions

By 2005, ***160 domain-independent specializations of a WordNet category*** were created: 18 during the integration of WordNet to the MSO and 142 later when using the ontology for representing knowledge. Among these specializations, 65 are WordNet categories and 90 are newly created categories (which are not part of the MSO top-level). Here are examples of these specialization relations. Their author (`pm') is not shown.

```
wn#yellow  > pm#blond_color;
wn#name  > pm#previous_surname  pm#middle_name;
wn#agency  > pm#real_estate_agency;
wn#region  > wn#dry_land;
wn#mass  > wn#mass_unit;
wn#city  > wn#capital_city;
wn#male  > wn#male_person;
wn#Tasmania  place: wn#Tasmanian_Island;
wn#Great_Britain  place: wn#Wales;
wn#acceleration  > wn#acceleration_unit;
wn#length  > wn#distance  wn#distance.size;
wn#Venus.Roman_deity  instance: wn#Aphrodite;
```

Sub-annotations were used to store certain information for guiding or checking knowledge entering. It is now clear that subtype/instance relations should have been used instead. This will be fixed in the future.

- For example, instead of using the approach shown in Table 3.1.2.1, sub-annotations were used to distinguish between qualities and quale: the sub-annotation "$(value)$" was added to 1300 subtypes of pm#attribute_or_quality_or_measure. This distinction is important to make since, unlike with types for qualities, it does not make much sense to use these subtypes in relation nodes even though pm#attribute_or_quality_or_measure is subtype of pm#thing_that_can_be_seen_as_a_relation. Hence, these subtypes are not proposed in the menus generated by WebKB-2 based on relations associated to a category via definitions or universally quantified statements. The same is true for individuals that are instances of pm#thing_that_can_be_seen_as_a_relation (only it subtypes can be used in relation nodes).
- The sub-annotation "$(artificial)$" was added to all WordNet categories that were found unfit for knowledge representation purposes, generally because they had a lexical rather than semantic character.

Here some examples of "$(value)$" or "$(artificial)$" sub-annotations.

```
wn#dark_red
  (^ $(value)$  a red that reflects little light^);
wn#gram___gramme___gm___g
  (^ $(value)$  a metric unit of weight equal to one thousandth of a kilogram^);
wn#west_by_south___WbS
  (^ $(value)$  the compass point that is one point south of due west^);
wn#andante
  (^ $(value)$  a moderately slow tempo^);
wn#Monday___Mon
  (^ $(value)$  the second day of the week; the first working day^)
wn#mealtime
  (^ $(value)$  the time for eating a meal^)
wn#thing.action
  (^ $(artificial)$  an action; "how could you do such a thing?"^)
wn#thing.happening
  (^ $(artificial)$  an event: "a funny thing happened on the way to the..."^)
wn#tonight
  (^ $(artificial)$  the present or immediately coming night^)
wn#then
  (^ $(artificial)$  that time; that moment; "we will arrive before then"^)
```

Finally, some ***statements were added to represent common relations from a category***. Figure 2.4.3.2 shows universally quantified statements about wn#person. The next FCG shows a universally quantified statement about wn#flight. The meanings of the sub-annotations "no inheritance" and "explore" are explained in subsection 2.4.4.

```
[any wn#flight (^$(no inheritance)$^),
  from_place: a pm#spatial_object,
  to_place: a pm#spatial_object,
  wn#day_of_the_week: a wn#day_of_the_week,
  via_place: a pm#spatial_object,
  departure_time: a pm#time_measure,
  arrival_time: a pm#time_measure,
  may have for relation_from_situation (^$(explore)$^): a pm#thing,
  agent: an wn#airplane_pilot,
  may have for experiencer: several wn#passenger
]_[pm];
```

Interest to bring more semantic corrections to WordNet? Although the top-level of WordNet was structured and complemented with a few relations, the direct specializations of nearly all WordNet categories remain quite heterogeneous, with few exclusion relations, and without distinction between role types and natural types. This lack of structure may be a problem for certain applications but fixing it might be as difficult as creating a better WordNet from scratch. Another problem is that distinctions in WordNet seem to have often been made not simply on semantic grounds but also on lexical grounds, thus leading to a multiplicity of "artificial" categories or categories that should be connected but are not. A few categories were marked as "artificial" but many more would need to be similarly marked, or connected by specialization relations, to improve knowledge normalization and retrieval. Since WordNet is unlikely to get much more structured and since nowadays DBs, KBs or ontologies are getting bigger (as with DBpedia), better structured (as with OpenGalen compared to other medical terminologies), more collaboratively built (as with Semantic Wikipedia, Freebase and OntoWiki) and more available (as with OpenCyc), it is likely that bigger and much better general ontologies than WordNet will become available in the medium term future. Hence, in the near future, I'll integrate a new version of WordNet but will not do major works on it.
Since WordNet now connects its "verb/adjective categories" (those having verbs or adjectives as names) to its noun categories (via relations of type pm#WordNet_noun_type) and its adverb categories to its verb/adjectives categories, I'll integrate all the categories of WordNet, not just those having nouns as names. Here are the currently foreseen translations of these direct or indirect pm#WordNet_noun_type relations.

- Such relations from verb categories to noun categories will be replaced by identity relations since i) there is a lexical distinction between these categories but apparently not a semantic distinction, ii) the lexical information will be kept by specifying what each of the names are (nouns, verbs, English words) using extended specialization relations, and iii) this will ease knowledge comparison.
- The adjective categories will be represented as (indirect) subtypes of pm#attribute_or_quality_or_measure and their pm#WordNet_noun_type relations to noun categories will be replaced by i) an identity relation if the noun is identical to the adjective (as with the categories for "red" or "male") or, if this is not the case, ii) a supertype relation if the noun category is subtype of pm#attribute_or_quality_or_measure, and iii) a relation of type pm#attribute_or_quality_or_measure otherwise. Due to the absence of specialization relations between WordNet adjective categories (and from some viewpoints their lack of conceptual meaning), the difference between these translations will not lead to inconsistencies and they cannot be seen as over-interpretations (they are just interpretations).
- The adverb categories will also be represented as subtypes of pm#attribute_or_quality_or_measure and their relations to verb categories (and thereby to noun categories) or adjective categories will be replaced by i) a supertype relation if these verb/adjective categories have been categorized as subtypes of pm#attribute_or_quality_or_measure, and ii) by a relation of type pm#attribute_or_quality_or_measure otherwise. Like WordNet adjective categories, the WordNet adverb categories have no specialization relations between them.

4. Towards a *Language Ontology* and a *Knowledge Presentation Ontology*

4.1. Example of Semi-Formal Discussion about RDF+OWL and the Need For More Expressiveness

Table 2.1.1.22.1 shows a small semi-formal discussion about certain drawbacks and usually claimed advantages of using XML-based languages for knowledge sharing. A longer semi-formal discussion about this subject can be found in [Martin, 2007b]. The next tables show a semi-formal discussion that was used as a support for stimulating, organizing and summarizing ideas in the discussion "D7 – Which languages are better than OWL?" on the mailing list of the SUO (Standard Upper Ontology IEEE committee) at the end of 2007 [www-SUO-D7, 2007]. In these next tables, the default author of the relations is `pm', the informal statements are left without author and they use the following abbreviations: "KR" for "Knowledge Representation", "KRL" for "KR language" and "F.O.L." for "first-order logic". The statements in bold characters are (counter-)argued in a subsequent table.

The subjects of these two semi-formal discussions lead to recurrent debates on many knowledge representation related mailing lists. If the participants to these mailing lists were adding their contributions to a normalized semi-formal discussion in a WebKB-2 like server in addition to (or instead of) writing to these mailing lists, a huge amount of redundancies (and hence of reading, writing and searching time) would be saved, a lot of precision would be gained (because of the need to write stand-alone argued ideas and relate them to other existing ideas), and the various ideas would be much more retrievable and comparable.

The next argumentation structures were incrementally designed by including into them each relevant idea of each message for the D7 discussion very soon after this message was published. Thus, these structures are not (yet) normalized but they contain most of the ideas relevant to the D7 discussion and in a much more concise and organized way than on the mailing list. The reformulations that were made for building the next argumentation structures led the participants of the discussion to precise their claims. E.g., they led Gian Piero Zarri (`gpZarri') – the researcher having provided the most messages to the D7 discussion – to finally provide precise explanations about its initial repeated claim that "non-binary relations are necessary for knowledge representation" and acknowledge that strictly speaking his claim was not true and that he actually simply wanted to highlight two requirements for a *general* KRL: i) permitting its users to express meta-statements, and ii) permitting them to specify that associating certain relations to the instances of certain categories is mandatory.
More generally, the alleged "necessity of using non-binary relations for precisely representing certain pieces of information" is often argued for (e.g., still by [Zarri, 2009]) but, it seems, without valid arguments. The probable absence of "necessity to *use* non-binary relations" is compatible with formal proofs that "besides unary and binary relations, there must *exist* at least one ternary relation in order to *generate* all possible relations" [Correia & Pöschel, 2006]. For knowledge sharing purposes, *using* non-binary relations has many drawbacks (Paragraph 2.3.2.2 summarizes them).

Even though they are not yet normalized, argumentation structures such as the ones below cannot be expected to be automatically extracted from informal discussions. However, they may be the result of a semi-automatic re-organization of discussions and then may be refined by further semi-formal discussions.
Research works about knowledge extraction from dialogs or discussion forums (e.g., [Makni et al., 2008]) most often find rhetorical relations between sentences (see Table 3.1.9.5 for a list) or generate an organized terminology.

Table 4.1.1. Semi-formal discussion about RDF+OWL not being a good general KRL

```
"RDF+OWL is not a good_KRL_for_exchanging_knowledge"
  argument: "RDF+OWL is not a good_general_KRL"
            __[pm, objection: "exchanging_ontologies_via_OWL does not require that
                               `OWL is a good_general_KRL'" __[jrSchoening] ];

  "exchanging_ontologies_via_OWL does not require that `OWL is a good_general_KRL'"
    argument: ("exchanging_ontologies_via_OWL is not detrimental to
                knowledge_representation/inferencing_with_a_language_more_expressive_than_OWL"
                  objection: – "exchanging_ontologies_via_OWL may lead to a loss of semantics"
                             – ("some applications require (or provide better results with)
                                 precisions that cannot be represented via OWL"
                                   .> "applications requiring precision need expressiveness")
              )__[jrSchoening];

  "RDF+OWL is not a good_general_KRL"
    argument:
      ("RDF+OWL is not expressive enough for general_KR"
          argument: "applications requiring precision need the expressiveness",
          argument: – "RDF+OWL does not include F.O.L., general meta-statements nor sets"
                    – "F.O.L., general meta-statements and sets are necessary to represent
                       certain natural language sentences or certain other information",
          argument: ("a general_knowledge_interchange_format should be as expressive
                      as possible"
                        argument: "the more expressive the KRL, the better for
                                   knowledge re-use" //see Table 4.1.4
                    ),
          argument: "most practical_applications cannot be carried on correctly making use
                     of a language as limited as OWL"__[gpZarri]  //see Table 4.1.4
          argument: – "the model of RDF+OWL does not permit to define
                       numerical quantifiers"
                    – "a perfect_general_KRL should permit to define numerical quantifiers"
          argument: – "the model of RDF+OWL does not include numerical quantifiers"
                    – "a perfect_general_KRL should include numerical quantifiers"
      ),
    .> "in 2007, no W3C_language is a good_general_KRL"
```

Table 4.1.2. Semi-formal discussion about W3C languages not being good general KRLs

```
"in 2007, no W3C_language is a good_general_KRL"
  opposition: "each W3C_language is useful for some purposes" __[jfSowa],
  argument:
    – ("in 2007, no W3C_language provides a way to specify in a definition that
        some relations are mandatory"
          objection: "OWL permits to specify that a relation is mandatory via the use
                      of cardinality restrictions or by specifying it as functional"
      )__[gpZarri]
    – "a good_general_KRL should provide a way to specify in a definition that
       some relations are mandatory"__[gpZarri],
  argument:
    – ("in 2007, no W3C_language provides a way to specify that some relations are
        contemporaneous"
          objection: – "RDF permits to use meta-statements"
                     – "meta-statements permit to specify that some relations are
                        contemporaneous"
      )__[gpZarri]
    – ("a good_general_KRL should provide a way to specify that some relations are
        contemporaneous"
         .> "a good_general_KRL should permit to use meta-statements"
      )__[gpZarri];

"the W3C should adopt a language like XCL"
  argument: – "an XML-based language is necessary for the
                transition from current W3C languages to
                more general KRLs for the future Semantic Web" __[jfSowa],
            – "XCL is an XML-based language for Common Logic" __[jfSowa],
            – "Common Logic supports URIs" __[jfSowa],
            – "the expressiveness of Common Logic is a minimal
                requirement for the Semantic Web" __[jfSowa];
```

Table 4.1.3. Semi-formal discussion about Common Logics not being a perfect general KRL

```
"in 2007, Common Logics is not a perfect_general_KRL"
  argument: – "in 2007, Common Logics does not permit to define numerical quantifiers"
            – "a perfect_general_KRL should permit to define numerical quantifiers",
  argument: – "in 2007, Common Logics does not include numerical quantifiers"
            – "a perfect_general_KRL should include numerical quantifiers";
  //note: KIF does not include numerical quantifiers but permits to define them

  "a perfect_general_KRL should permit to define numerical quantifiers"
    argument: ("a user should be able to represent numerical quantifiers"
                 argument: – "numerical quantifiers are needed to represent
                              natural language sentences or certain other information"
                           – "applications requiring precision need the expressiveness"
              );

  "a perfect_general_KRL should include numerical quantifiers"
    .< ("a perfect_general_KRL should include as many knowledge representation
         constructs (such as numerical quantifiers) as possible";
           argument:
             ("a user should be able to use KR constructs (such as numerical)
               quantifiers without having to define them"
                argument:
                  – "very few inference engines are able to exploit the logic definition
                     of complex KR constructs (such as numerical quantifiers) for
                     inferencing purposes"
                  – "it is complex to implement an exploitation of the logic definition
                     of complex KR constructs (such as numerical quantifiers) for
                     inferencing purposes"
                  – "it is easy to calculate a generalization relation between expressions
                     using complex KR constructs (e.g., quantification expressions such
                     as 'at least 2', 'at least 3' and '4'"
                  – "generalization relations are useful for all kinds of inferencing,
                     including logical inferencing"
                  – ("not all expressions using complex KR constructs can be defined in
                      an ontology"
                       example: "there are an infinite number of possible
                                 numerical quantification expressions"
                    )
                  – "even if all expressions using complex KR constructs could be defined
                     in ontologies, different users would use KR construct categories from
                     different ontologies and hence, when these categories from different
                     ontologies are not connected by identity/equivalence relations,
                     knowledge comparison/retrieval/sharing is reduced";
```

Table 4.1.4. More argumentation structures about the need for expressive KRLs

```
"the more expressive the KRL, the better for knowledge re-use"
  argument: "the less expressive the KRL, the more a user may have to
             represent knowledge in an imprecise and ad-hoc way",
  objection: "the more expressive the knowledge, the more difficult
              it will be to use for efficient reasoning"
             __[fg, objection: "applications requiring precision need the expressiveness",
                    objection: "applications requiring efficiency and not a great precision
                                can ignore some certain elements of the statements and, for
                                example, use simple graph-matching techniques"
             );


"most practical_applications cannot be carried on correctly making use of a language as
 limited as OWL"
  .> ("most practical_applications requires a language more expressive than OWL"
        argument: – "a practical_application requires the use_of_rules"__[gpZarri]
                  – "the use_of_rules is hard with a DL"__[gpZarri],
        argument: "any notation for OWL can only be ugly and verbose"__[gpZarri],
        argument: "OWL is difficult for an average programmer to deal with"__[gpZarri],
        argument: ("OWL has been a flop from a concrete/commercial viewpoint until 2007"
                     source: "IEEE Intelligent Systems issue of September/October of 2007"
                             __[gpZarri]
                  )__[gpZarri],
        argument: ` "DL-Safe-SWRL is decidable" has for consequence "SWRL variables can
                     only be bound to known individuals in a KB" '__[gpZarri]
     ),
  argument: "most programmers do not care about computational_tractability"__[gpZarri],
  argument: ("RDF is probably more useful than OWL from an 'applicability' viewpoint"
               argument: ("several commercials_products use RDF and not OWL"
                             example: "Oracle_11g_RDF_database uses RDF, not OWL"__[gpZarri],
                             example: "GroupMe! and not OWL"__[gpZarri]
                          )__[gpZarri],
            )__[gpZarri];
```

4.2. Comparison of Three Main Notations of WebKB With Other Knowledge Representation Languages

Goals of making such a comparison via the panorama of knowledge representation features proposed by the next subsections. Subsection 1.2.3 includes the claim that FL, FCG and FE

- have a same underlying formal model which extends the CG model and the ISO Common Logic model (CL) for expressiveness and collaboration purposes but which is not more expressive than the model of KIF) and
- improve over other existing notations (e.g., Lisp-like notations, frame-like/graph-based notations and, from a restricted viewpoint, formal controlled natural languages) on at least two of the following criteria: expressiveness, normalization, intuitiveness and concision.

The next subsections illustrate this claim by comparing FL, FCG and FE with other notations:

- CGLF since this is the notation from which were derived FCG, FL and FE in order to further improve on its expressiveness, normalization, intuitiveness and concision, that is, on the characteristics that generally made it appealing to those who adopted it,
- CGIF since it is the official syntax for CG and one standard syntax for the CL model,
- KIF since it is a prefixed notations that is well defined and more expressive than CLIF (the KIF like standard notation for CL), e.g., CLIF does not support contexts since the CL model does not support them (IKL [Hayes & Menzel, 2006] is an extension of CLIF that supports contexts but is not as expressive as KIF),
- RDF+OWL-Full/XML (RDOX) since this is the de-facto standard KRL, and
- RDF+OWL-Full/Notation3 (RON3) since it is often used by the Semantic Web community as a more concise alternative to RDOX (in the future, the functional-style syntax of OWL 2 may become a more common alternative).

The comparison is made on a ***panorama of KR features*** commonly required for representing natural language sentences (e.g., contexts, definitions, collections, ... but no fuzzy logics related features). In each example, an English (E) sentence is given and translated in the other notations. Each sentence illustrates one or several KR features. This panorama does not list all the features of the compared languages (e.g., it does not list all those presented in Subsection 2.1.1) but permits to highlight various inadequacies of CGLF, CGIF, RDOX and RON3. In these illustrations of these four notations, ad-hoc representations are often proposed. Some ad-hoc terms are used. To ease knowledge entering, matching and sharing with these notations, some of these ad-hoc terms could be included as keywords to these notations. These particular ad-hoc terms are highlighted using italic characters. When more than one representation (composed of one or several statements) are proposed in the same notation, these representations are equivalent. When the translation in a notation is obvious (given previously given translations), it is not given. The authors of the terms and statements are most often left implicit. The KIF translations provide a logical interpretation. When the RDOX translation uses an OWL or RDF relation type and no equivalent relation type or short construct exists in KIF, this relation type is also used in the KIF translation.

Knowledge providers may see this panorama as a ***guide to knowledge representation***, because

- it may help making representations that can be represented in various notations, and
- it complements the documentations of notations (the above cited ones or others) since those documentations often omit to explain how to represent certain complex cases or to state that some of these cases cannot be represented (this is especially true for RON3: coming up with the RON3 statements presented below was often difficult and many parts in these statements may be ad-hoc even though they are not marked as such); this lack of details in these documentation makes notations difficult to compare.

Language developers may see this panorama as a list of cases to take into account for their notations or inferences engines and for comparing them to other ones.

Finally, this panorama shows how the above cited notations can be ***syntactically translated*** into each other – "syntactically" because no deep translation involving conversions between ontologies (e.g., as in [Corcho, 2005]) is involved. Regarding such translations, the originality of this work lies in the expressiveness of the listed features. Hence, it complements more theoretical works such as [Yao & Etzkorna, 2006] and [Baget et al., 2009] that focus on

syntactic translations basic RDF statements (i.e., those only requiring the RDFS model) and simple CG statements; this part is illustrated by Subsection 4.2.1. Another example of complementary work is [Hayes, 2005] which proposed translations of OWL relations (i.e., having types defined in OWL) into CLIF.
To conclude, even though FL, FCG and FE are unlikely to be widely adopted, this panorama can provide some ***insights in knowledge representation and the development or improvements of notations that are more expressive, normalized, intuitive or concise***.

For readability reasons and for supporting various kinds of knowledge entering or views on the knowledge, ***various formal or semi-formal notations should be proposed by knowledge servers.***

- FL is interesting for displaying a lot of information in a very concise and structured way.
- FCG is more interesting for short statements (e.g., query graphs), especially if they are complex.
- FE is neither structured nor concise and hence is not adequate for building or browsing a reasonably complex KB but it can be shown to anyone (any beginner in KR). Hence, it can for example be used for showing the various interpretations that a NLU parser makes of a sentence expressed in a natural or controlled language and then let the user select the correct interpretation or precise the sentence.

Towards a scalable comparison table about expressiveness features for KRLs. The next table summarizes some pieces of information from the next subsections. In the near future, this table will be extended to include classic distinction in logics such as those referred to by the letters in the names of description logics such as SHOIN and SROIQ(D) to which respectively correspond OWL DL and OWL 2. In this table, 'F*' refers to FL, FCG and FE while OWLL, OWLD and OWLD respectively refer to RDF+OWL-Lite, RDF+OWL-DL and RDF+OWL-Full. Here is the list of marks used:

- '+': the language/notation has this feature,
- '-': the language/notation does not have this feature,
- '+-': the language/notation has this feature only with some predefined concepts/relations,
- '?': unknown,
- '+.': the notation has the feature and some short syntactic sugar for this feature,
- '+=': the language allows the definition of a relation/function/concept for using the corresponding feature in an easy way.

Table 4.2.20.1. Comparison of some notations according to some expressiveness features

	KIF	CL	CGIF	CGLF	N3	F*	OWLL	OWLD	OWLF
conjunction	+.	+.	+.	+.	+.	+.	+	+	+
conjunction_of_types	+=	+=	+=	+.	–	+	–	+	+
conjunction_of_statements	+.	+.	+.	+.	+.	+.	+	+	+
inclusive_disjunction	+.	+.	+	+	+	+	–	–	–
inclusive_disjunction_of_types	+=	+=	–	–	+	+	–	+	+
inclusive_disjunction_of_statements	+.	+.	+	+	+	+	–	–	–
exclusive_disjunction	+=	+=	+=	+=	+=	+=	–	–	–
exclusive_disjunction_of_types	+=	+=	+	+	+	+	–	–	+
formula_conjunction_of_statements	+=	+=	+=	+=	+	+=	–	–	–
negation	+.	+.	+.	+.	+	+.	–	+	+
exclusion_of_types	+=	+=	+=	–	+	+	–	+	+
complement_of_type	+=	+=	+=	+.	+	+	–	+	+
negation_of_statement	+.	+.	+.	+.	+	+.	–	–	–
relation	+	+	+	+	+	+	+	+	+
relation_on_type_to_define_this_type	–	–	–	–	+	+	+	+	+
relation_on_statement	+	–	+	+	+	+	–	–	–
n-ary_relation	+	+	+	–	–	+	–	–	–
computed_relation (actor)	+=	–	–	+	–	+	–	–	–
collection	+	+	+	+	+	+	+	+	+
set	+	+	+	+	+	+	–	–	–
collection_of_types	+	+	+	+	+	+	–	–	–
collection_of_statements	+	–	+	+	+	+	–	–	–
AND_collection	–	+	+	+	+	+	–	–	–
OR_collection	+	+	+	+	–	+	–	–	–
XOR_collection	+=	+=	+=	+=	–	+	–	–	–
distributive_collection	+	+	+	+	+	+	+	+	+
collective_collection	–	–	+	+	–	+	–	–	–
quantification	+	+	+	+	+	+	+	+	+
existential_quantification	+	+	+	+	+	+	+	+	+
universal_quantification	+	+	+	+	+	+	–	–	–
numerical_quantification	+=	–	–	–	–	+	–	–	–
quantification_with_positive_integer	+=	–	–	–	–	+	–	–	–
quantification_with_percentages	+=	–	–	–	–	+	–	–	–
type_definition	+	+	+	+	+	+	+	+	+
lambda_abstraction	+	+	+	+-	+	+	+-	+-	+-
category_identifier	+	+	+	+	+	+	+	+	+
URI_as_category_identifier	+=	+	+	+	+	+	+	+	+
data_type	+	+	+	+	+	+	+	+	+
string	+	+	+	+	+	+	+	+	+
time_duration	–	–	–	–	–	+	–	–	–
date	–	–	–	–	–	+	–	–	–
informal_annotation	–	–	+	+	–	+	–	–	–
informal_annotation_on_type	–	–	+	+	–	+	–	–	–
informal_annotation_on_statement	–	–	+	+	–	+	–	–	–
comment (discarded at parsing)	+	+	+	+	+	+	+	+	+
line_comment	+	?	?	?	–	+	–	–	–
multi-line_comment	–	?	?	?	+	+	+	+	+

4.2.1. Existential Quantification, Conjunction, Difference

```
E:     Tom owns a dog that is not Snoopy.
FL:    Tom owner of: (dog != Snoopy __[?x<->.]) __[.->?x];
FL:    Tom owner of: (dog ?x != Snoopy);     //the two relations share the same individual ?x
FL:    Tom owner of: (a dog != Snoopy);      //the two relations share the same individual
FL:    Tom owner of: (a dog ?x != Snoopy);   //idem
FL:    [Tom owner of: (a dog ?x != Snoopy)]; //idem
FL:    Tom owner of: a (dog != Snoopy);      //idem but with a type restriction/lambda-abstraction
     //Tom owner of: (dog != Snoopy __[any->?]);      //<=> "Tom owns a dog and any dog != Snoopy"
     //Tom owner of: a (. dog != Snoopy __[any->?]);  //<=> "Tom owns a dog and any dog != Snoopy"
FE:    Tom is owner of a dog different_from Snoopy.
FE:    Tom is owner of a dog != Snoopy.
FE:    Tom is owner of a `dog != Snoopy'.
FCG:   [Tom, owner of: (a dog != Snoopy)]; //'!=' is not in Sowa's CGLF grammar but he uses it
FCG:   [Tom, owner of: a (dog != Snoopy)];
CGLF:  [T:Tom]<-(owner)<-[dog: *x != Snoopy]; //'T': uppermost concept type (pm#thing)
CGIF:  [dog:*x] (owner ?x Tom) (different_from ?x Snoopy)
KIF:   (exists ((?x dog)) (and (owner ?x Tom) (/= ?x Snoopy)))
RDOX:  <Dog><owner><owl:Thing rdf:about="#Tom"/></owner>
          <owl:differentFrom rdf:resource="#Snoopy"/></Dog>
RON3:  a :Dog;  :owner :Tom;  owl:differentFrom :Snoopy.
```

4.2.2. Simple Contextualizations Or Meta-statements

The W3C Specification of January 23rd 2003 for the RDF/XML syntax [www-RDF/XML 2004] removed the `rdf#bagID' and `rdf#aboutEach' keywords from the RDF grammar. These keywords permitted *simple meta-statements* not just on a "triple" (one relation) but *on a statement composed of many relations*. Since then, RDF/XML users are expected to express meta-statements by reifying a statement (i.e., giving it a name via the rdf#ID property/relation and then referring to it via rdf#about) and it seems that according to the RDF documentation [www-RDF/XML-reification 2004] this reification can only be performed if the statement is a triple.
It should also be noted that (current) RDF reification does not permit to express any notion of nesting, scope or truth restricting context. To permit this, and then also permit quantifiers, disjunction and negation (more general ones than OWL permits), various propositions of extensions to this model have been proposed, e.g., [Conen et al., 2001] [Mcdermott & Dou, 2002] [Patel-Schneider, 2005]. The authors of such works expect that in the future the RDF+OWL model will be extended to be much more expressive. In the mean time, in this chapter, rdf#bagID and rdf#aboutEach are used and they are assumed to permit the expression of nesting, scope and truth restricting contexts.

The W3C Rule Interchange Format [www-RIF 2009] and its Basic Logic Dialect permit to represent "definite Horn rules" but, even when combined with RDF+OWL, is less expressive than CL. Neither RIF nor SPARQL (the W3C query language for RDF+OWL graphs) are discussed here.

```
E:     Tom believes Mary likes him (now) in 2003, and that before she did not.
FL:    Tom  believer of: [?p [Mary agent of: (a liking  object: Tom)]
                            time: 2003]  [!?p  before: 2003];
FE:    Tom is believer of ` ?p `Mary is liking Tom' at time 2003'
       and is believer of `!?p is before 2003'.
FCG:   [Tom, believer of: [?p [Mary, agent of: (a liking, object: Tom)], time: 2003],
             believer of: [!?p, before: 2003] ];
CGLF:  [proposition: *p [T:Mary]<-(agent)<-[liking]->(object)->[T:Tom] ]
       [T:Tom]- { (believer)<-[ [situation: ?p]->(time)->[time:"2003"],
                  (believer)<-[ [situation:~?p]->(before)->[time:"2003"] ] }
CGIF:  [proposition *p: (agent [liking *l] Mary)  (object ?l Tom) ]
       (believer [situation: (time [situation: ?p] "2003")] Tom)
       (believer [situation: (before [situation: ~[?p]] "20032")] Tom)
KIF:   (exists (?p)
         (and (= ?p '(exists ((?x liking)) (and (agent *l Mary) (object ?l Tom))))
              (believer ^(time ,?p 2003) Tom)
              (believer ^(before (not ,?p) 2003) Tom)))
RDOX:  <rdf:Description rdf:bagID="p">
         <Liking><agent><rdf:Description rdf:about="#Mary"/></agent>
                 <object rdf:resource="#Tom"/></Liking></rdf:Description>
       <rdf:Description rdf:bagID="now"  rdf:aboutEach="#p"  time ="2003"/>
       <owl:Class rdf:bagID="class_of_not_p">
         <owl:complementOf rdf:parseType="Collection">
           <owl:Class><owl:oneOf rdf:parseType="Collection">
              <owl:Thing rdf:about="#p"/></owl:oneOf>
           </owl:Class></owl:complementOf></owl:Class>
       <class_of_not_p rdf:bagID="not_p"/>
       <rdf:Description rdf:bagID="past"  rdf:aboutEach="#not_p"  before="2003"/>
       <rdf:Description rdf:aboutEach="#now"><believer rdf:resource="#Tom"/>
       </rdf:Description>
       <rdf:Description rdf:aboutEach="#past"><believer rdf:resource="#Tom"/>
       </rdf:Description>
RON3:  { :p {[a :Liking;  :agent :Mary; :object :Tom]}  :time "2003"} :believer :Tom.
       :not_p negation :p.  {:not_p :before "2003"} :believer :Tom.
```

The above FE and FCG statements are shorter and clearer than the other translations.

- Compared to the CGLF and CGIF statements, this is due to the minimization of syntactic sugar and to the fact that i) intermediary nodes of type pm#proposition and pm#situation do not have to be used, and ii) the scope of a variable is restricted to the same context (in FL, FCG and FE, the scope of a variable is until the end of the statement where it has been declared).
- The predicate oriented notation of CGIF and KIF, and hence their use of additional variables, also make them less readable and longer.
- The RON3 statements have some similarities with the FCG statement but have a more complex and less systematic syntactic sugar (it is different whether or not a node has one, two or more relations) and the direction of relations cannot be simply reversed as with the keyword 'of' in FCG. This is why there are three RON3 statements and only one FCG statement.

More generally, FCG and FL look like other frame-like notations, e.g., Frame-Logics and the JSON notation which is now often used to store, handle and print RDF graphs with Javascript. However, the syntactic choices made for these notations does not permit them to be extended in a simple and homogeneous way to include all the constructs (quantifiers, contexts, collections, ...) that FL, FCG and FE have.

In the above example, to represent the negation of a statement in OWL, an intermediary class is created for this statement and then an instance of the complement of this intermediary class is referred to. Although syntactically correct, it is doubtful that an inference engine will be able to exploit such an indirection any time soon.

4.2.3. Identities, Names and Authorship

```
E:     According to pm, pm#nothing is identical to owl#nothing,
       and according to Joe, owl#nothing identical to kif#bottom.
FL:    pm#nothing   =  owl#nothing _[pm]  kif#bottom _[Joe];
FE:    `pm#nothing = owl#nothing'_[pm];  `pm#nothing = kif#bottom'_[Joe];
FCG:   [pm#nothing, = owl#nothing]_[pm];  [pm#nothing, = kif#bottom]_[Joe];
CGLF:  [ [pm#nothing]->(equal)->[owl#nothing] ]->(pm#author)->[T: pm];
       [ [pm#nothing]->(equal)->[kif#bottom] ]->(pm#author)->[T: Joe];
CGIF:  (equal pm#nothing owl#nothing)
       (pm#author [situation: (equal pm#nothing owl#nothing)] pm)
       (pm#author [situation: (equal pm#nothing kif#bottom)] Joe)
KIF:   (dc#Creator '(= pm#nothing  owl#nothing) pm)
       (dc#Creator '(= pm#nothing  bottom) Joe)
RDOX:  <owl:Class rdf:about="&pm;Nothing">
         <owl:equivalentClass rdf:resource="&kif;Bottom" dc:Creator="Joe"/>
         <owl:equivalentClass rdf:resource="&owl;Nothing" dc:Creator="spamOnly@phmartin.info"/>
       </owl:Class>
RON3:  {pm:Nothing owl:equivalentClass kif:Bottom}  dc:Creator "Joe".
       {pm:Nothing owl:equivalentClass owl:Nothing} dc:Creator "spamOnly@phmartin.info".
       //owl:equivalentClass is for OWL-Lite; owl:same_as can be used with OWL-Full
```

The above RDOX translation uses dc#Creator (a supertype of pm#author) the way 'xml:lang' is used in XML, that is, as if it was a special XML attribute of relation. Using a meta-statement (as in the previous section) would be more correct but would reduce readability. Similarly, the 'xml#lang' relation is used in the next example.

```
E:     There is a type named "dog" and "domestic_dog" by wn, and
       "chien" by pm; "dog" is a key name for wn; "chien" is a French name.
FL:    wn#dog__domestic_dog  kind: pm#type,
         name: ("chien" language: wn#French __[.->?]) __[pm];
CGIF:  [Type: wn#dog *x] (pm#author ?x http://wordnet.princeton.edu)
       (pm#author [situation: (pm#name ?x "domestic_dog")] http://wordnet.princeton.edu)
       (pm#author [situation: (pm#name ?x "chien")] pm)
       (pm#author [situation: (pm#language "chien" [wn#French])] pm)
KIF:   (pm#type wn#dog)  (dc#Creator '(rdfs#label wn#dog "dog") http://wordnet.princeton.edu)
       (dc#Creator '(rdfs#label wn#dog "domestic_dog") http://wordnet.princeton.edu)
       (dc#Creator '(rdfs#label wn#dog "chien") pm)  (dc#Creator '(xml#lang "chien" fr) pm)
RDOX:  <owl:Class rdf:about="&wn;dog">
         <rdfs:label dc:Creator="http://wordnet.princeton.edu">dog</rdfs:label>
         <rdfs:label dc:Creator="http://wordnet.princeton.edu">domestic_dog</rdfs:label>
         <rdfs:label xml:lang="fr" dc:Creator="spamOnly@phmartin.info">chien</rdfs:label>
       </owl:Class>
RON3:  {wn:Dog rdfs:label "dog", "domestic_dog"} dc:Creator "http://wordnet.princeton.edu".
       {wn:Dog rdfs:label "chien"} dc:Creator "spamOnly@phmartin.info".
       {"chien" xml:lang "fr"} dc:Creator "spamOnly@phmartin.info".
```

4.2.4. Relation Type Declaration

```
E:     Relations of type pm#husband are both functional and inverse functional,
       and relate an object of type pm#woman to an object of class pm#man.
FL:    pm#husband .(pm#woman [0..1], pm#man [0..1]);
FL:    pm#husband .(pm#woman [0..1] -> pm#man);  //idem in FE and FCG
CGLF and CGIF:  //no standard -> use of the same OWL relations as in KIF, RDOX and RON3
KIF:   (owl#functional_property pm#husband)  (owl#inverse_functional_property pm#husband)
       (rdfs#domain pm#husband pm#woman)  (rdfs#range pm#husband pm#man)
RDOX:  <owl:FunctionalProperty rdf:about="&pm;husband">
         <rdf:type    rdf:resource="&owl;InverseFunctionalProperty" />
         <rdfs:domain rdf:resource="&pm;Woman"/>
         <rdfs:range  rdf:resource="&pm;Man"/> </owl:FunctionalProperty>
RON3:  pm:husband  a owl:FunctionalProperty, owl:InverseFunctionalProperty;
                   rdfs:domain pm:Woman;   rdfs:range pm:Man.

E:     Relations of type pm#parent are from pm#animal to pm#animal. The domain cardinality
       is 0..* and the range cardinality 2..2 (in our real world).
FL:    pm#parent .(pm#animal, pm#animal [2]);  //idem in FE and FCG
FL:    pm#parent .(pm#animal, pm#animal);  pm#animal pm#parent: 2 pm#animal __[0..*<-any];
CGLF and CGIF:  //no standard form -> use of the same OWL relations as in RDOX and RON3
KIF:   (defrelation pm#parent (?x ?y) :=> (and (pm#animal ?x) (pm#animal ?y)))
       (forall ((?a pm#animal)) (existsN ('?p pm#animal (pm#parent '?a ?p))))
       ;;'existsN' is defined in Subsection 4.2.6
RDOX:  <owl:ObjectProperty rdf:about="&pm;parent">
         <rdfs:domain rdf:resource="&pm;Animal"/>
         <rdfs:range  rdf:resource="&pm;Animal"/> </owl:ObjectProperty>
       <owl:Class rdf:about="&pm;Animal"/>
         <rdfs:subClassOf><owl:Restriction>
           <owl:onProperty rdf:resource="&pm;parent"/>
           <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">2
           </owl:cardinality> </owl:Restriction> </rdfs:subClassOf> </owl:Class>
RON3:  pm:parent  a owl:ObjectProperty;  rdfs:domain pm:Animal;  rdfs:range pm:Animal.
       pm:Animal  rdfs:subClassOf
                    [a owl:Restriction;  owl:onProperty pm:Parent;  owl:cardinality 2].
```

FL, FCG, FE and KIF accept N-ary relations. To translate such a relation into RDOX and RON3, the W3C approved way [Hayes & Welty, 2006] is to convert it into a concept type and introduce a relation for each argument.

```
FL:    between_3 (pm#spatial_object,pm#spatial_object,pm#spatial_object);  //idem in FCG
RDOX:  <owl:Class rdf:ID="Between_3"/>
       <rdf:Property rdf:ID="between_3_arg1">
         <rdfs:domain rdf:resource="&pm;SpatialObject"/> </rdf:Property>
       <rdf:Property rdf:ID="between_3_arg2">
         <rdfs:domain rdf:resource="&pm;SpatialObject"/> </rdf:Property>
       <rdf:Property rdf:ID="between_3_arg3">
         <rdfs:domain rdf:resource="&pm;SpatialObject"/> </rdf:Property>
RON3:  :Between_3  a owl:Class;
       :between_3_arg1  a owl:ObjectProperty;  rdfs:domain pm:Between_3;
                                               rdfs:range pm:SpatialObject.
       :between_3_arg2  a owl:ObjectProperty;  rdfs:domain pm:Between_3;
                                               rdfs:range pm:SpatialObject.
       :between_3_arg3  a owl:ObjectProperty;  rdfs:domain pm:Between_3;
                                               rdfs:range pm:SpatialObject.
```

4.2.5. Universal Quantification, Definitions and Lambda-Abstractions

Paragraph 2.1.1.5 and Subsection 2.2.5 explain the difference between a definition of a type and a statement using a universal quantifier for this type, and how this difference is exploited in the shared KB edition protocols of WebKB-2. In FL, FCG and FE, 'every' refers to the universal quantifier while 'any' permits to make a definition using the form of a universally quantified statement. Other forms are also available.

Since RDF+OWL does not make such a distinction, the RDOX example below uses a definition but adds a relation of type pm#quantifier. This is of course completely ad-hoc (since RDF+OWL inference engines will not interpret pm#quantifier in a special way) and is not even consistent with the fact that a rdfs#subClassOf relation is used but i) this information may be understood by people or exploited by certain applications, and ii) other quantifiers, e.g., numerical quantifiers such as '83%', can be specified instead of 'every'. Curiously, Notation 3 has a universal quantifier; hence it is used here. Unlike with other notations, the RDOX and RON3 examples have to declare the relation type `headPart'.

In CGLF and CGIF, since the order of the concept nodes in a statement has no importance, the precedence of the quantifier must be specified via the embedding of contexts. To reduce the need for this cumbersome operation, Sowa added the rule: "the existential quantifier has less precedence than the universal quantifier". For CGLF and CGIF statements, it seems that this rule may be generalized by "a more specialized quantifier has more precedence".

In FL, FCG and FE, the order of quantifiers is determined by their order of appearance in the entered statement. In the current structure of KBs in WebKB-2, this order is kept when isolated statements are stored. In the future KB structure or with relations entered via FL, the quantifiers are associated to each relation and, when variables are used, there is a generated unique identifier for each variable.

```
E:     Animals happen to have exactly one head.
FE:    animal  part:  head __[every->1];
FE:    Every animal has for part 1 head.
FCG:   [every animal, part: 1 head];
CGLF:  [animal: @forall]->(part)->[head: @1]  //'@1' is common but not standard
CGIF:  (part [animal: @forall] [head: @1])
KIF:   (forall ((?a animal)) (exists1 '?h (and (head ?h) (part ?a '?h))))
       ;;'exists1' is defined in Subsection 4.2.6
RDOX:  <rdf:Property rdf:ID="headPart">
         <rdfs:subPropertyOf rdf:resource="#part"/>
         <rdfs:range rdf:resource="#Head"/> </rdf:Property>
       <owl:Class rdf:about="#Animal" pm:quantifier="every">
         <rdfs:subClassOf><owl:Restriction>
           <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
           </owl:cardinality> <owl:onProperty rdf:resource="#headPart"/>
                        </owl:Restriction></rdfs:subClassOf></owl:Class>
RON3:  :headPart a rdf:Property;  rdfs:subPropertyOf :part;  rdfs:range :Head.
       :StuffWithOneHead
          rdfs:subClassOf [a owl:Restriction;  owl:onProperty :headPart;  owl:cardinality 1].
       @forAll :a . { {:a rdf:type :animal; rdf:type :StuffWithOneHead; :part :Head; }
                      :modality :PhysicalPossibility }.
```

A owl#restriction concept node can be seen as a lambda abstraction, that is, an on-the-fly definition of necessary and sufficient conditions for an unnamed type. The next example illustrates a definition of necessary conditions for a person and shows how a lambda-abstraction is delimited/introduced: via parenthesis in FL and FCG, via quotes in FE, via the use of the 'lambda' keyword in CGLF and CGIF, and via a owl#restriction concept node in RDOX and RON3. This example illustrates four different forms for representing a definition of necessary conditions: a quantified statement form (see 'any'), a definition construct (see ':=>'), an implication between two statements within a definition

(see '=>') and a subtype relation. The following CGLF and CGIF may or may not be ad-hoc: indeed, the current CG standard is quite incoherent and restrictive about lambda-abstractions and type definitions.

```
E:     By definition, a person necessarily has for parent a male person.
FL:    person  parent: (person  kind: male __[?p->.]) __[any->?p];
FL:    person  parent: (a person  kind: male);
FL:    person  parent: a (person  kind: male);  //here, a lambda-abstraction is used
FE:    Any person has for parent a `person that is male';
FE:    Any person has for parent a male person;  //indeed, `male' is not a subtype of
                      //  pm#attribute_or_quality_or_measure (see Paragraph 2.3.1.2)
FCG:   [any person, parent: a male person];
FCG:   [type person (*x) :=> [*x, parent: a (person, kind: male)] ];
CGLF:  type person (*x)
         [ [T: *x]   [[person: *y] => [(lambda(*p) [person: ?p]->(kind)->[T:male]) : *z ]] ];
CGIF:  (rdfs#subClassOf person (lambda(*x) ((person ?x) (kind ?x male))))
KIF:   (defrelation person (?p)
          :=> (exists ((?p2 #person)) (and (pm#parent ?p ?p2) (kind ?p2 male))))
RDOX:  <owl:Class rdf:about="&wn;Person">
         <rdfs:subClassOf><owl:Restriction>
                           <owl:onProperty rdf:resource="&pm;parent"/>
                           <owl:allValuesFrom>
                             <owl:Class rdf:ID="malePerson">
                               <owl:intersectionOf rdf:parseType="Collection">
                                 <owl:Class rdf:about="&wn;Person"/>
                                 <owl:Class rdf:about="&wn;Male"/>
                               </owl:intersectionOf></owl:Class></owl:allValuesFrom>
                         </owl:Restriction></rdfs:subClassOf></owl:Class>
RON3:  wn:Person  a owl:Class;  rdfs:subClassOf
         [a owl:Restriction; owl:onProperty pm:parent;
                             owl:allValuesFrom [a owl:Class owl:intersectionOf (wn:male wn:Person)].
```

Similarly, a definition of sufficient conditions can use "<=" in FL, ":<=" in FCG and KIF, and an inverse of a rdfs:subClassOf relation in RDOX and RON3. A definition of necessary and sufficient conditions can use "<=>" in FL, ":<=>" in FCG, ":=" in KIF and, in RDOX and RON3, a relation of a type such as owl#equivalent_class, owl#equivalent_property, owl#intersection_of, owl#union_of and owl#complement_of. The next example illustrates this.

```
E:     A pm#initiator is a pm#causal_entity and wn#initiator that is pm#agent of a process.
FL:    pm#initiator  :<=> ( (pm#causal_entity  wn#initiator)  pm#agent of<=  process);
FCG:   type pm#initiator (*x) :<=> [pm#causal_entity wn#initiator *x, pm#agent of: a process];
CGLF:  type pm#initiator (*x)  [pm#causal_entity & wn#initiator: *x]<-(pm#agent)<-[process];
CGIF:  (= person (lambda(*x) ([pm#causal_entity:*x][wn#initiator:*x](pm#agent *x [process]))))
KIF:   (defrelation initiator (?x) :=  (exists ((?p process))
          (and (pm#causal_entity ?x) (wn#initiator ?x) (pm#agent ?p ?x))))
RDOX:  <owl:ObjectProperty rdf:ID="agentOf"><owl:inverse_of rdf:resource="agent"/>
       </owl:ObjectProperty>
       <owl:Class rdf:about="&pm;initiator">
         <owl:intersectionOf rdf:parseType="Collection">
             <owl:Class rdf:about="&pm;CausalEntity"/>
             <owl:Class rdf:about="&wn;Initiator"/>
             <owl:Restriction><owl:onProperty rdf:resource="#agentOf"/>
               <owl:allValuesFrom rdf:resource="#Process"/></owl:Restriction>
         </owl:intersectionOf></owl:Class>
RON3:  :agentOf  a rdf:Property;  owl:inverse_of pm:agent.
       :initiator  a owl:Class;
          owl:intersectionOf (pm:CausalEntity  wn:Initiator
                [a owl:Restriction; owl:onProperty :agentOf; owl:allValuesFrom pm:Process])
```

A type can be defined by intension or by extension. Here is an example.

```
E:     The type Da_Ponte_opera_of_Mozart has only three instances:
       Nozze_di_Figaro, Don_Giovanni and Cosi_fan_tutte.
FL:    Da_Ponte_opera_of_Mozart
        instance: {Nozze_di_Figaro,Don_Giovanni,Cosi_fan_tutte} __[. -> .complete];
        //the use of the keyword `complete' has been explained in Paragraph 2.1.1.11
FCG:   [a Da_Ponte_opera_of_Mozart, = OR{Le_Nozze_di_Figaro,Don_Giovanni,Cosi_fan_tutte}]
       //by default, the interpretation of a collection is distributive
CGLF and CGIF:  //no standard form -> use of keywords/forms similar to those in FCG, e.g.,
       (=  [Da_Ponte_opera_of_Mozart] [OR{Nozze_di_Figaro,Don_Giovanni,Cosi_fan_tutte}]
KIF:   (<=> (Da_Ponte_opera_of_Mozart ?x)
            (or (= ?x Nozze_di_Figaro) (?x Don_Giovanni) (= ?x Cosi_fan_tutte)))
RDOX:  <owl:Class rdf:ID="Da_Ponte_opera_of_Mozart">
         <owl:oneOf rdf:parseType="Collection">
           <owl:Thing rdf:ID="#Nozze_di_Figaro"/>
           <owl:Thing rdf:ID="#Don_Giovanni"/>
           <owl:Thing rdf:ID="#Cosi_fan_tutte"/>
         </owl:oneOf></owl:Class>
RON3:  :Da_Ponte_opera_of_Mozart  a owl:Class;
          owl:oneOf (:Nozze_di_Figaro, :Don_Giovanni, :Cosi_fan_tutte).
```

4.2.6. Relation Cardinalities (a Restricted Kind of Numerical Quantification?)

In the next example, compared to the other notations, the RDOX and RON3 translations require 8 additional intermediary objects: 2 concept nodes of type owl#restriction, the declarations of the types `armPart' and `armPartOf', and the use of the relations of type rdfs#subClassOf, rdfs#range and owl:inverseOf. This illustrates how cumbersome or low-level the RDF+OWL model is. This cumbersomeness does not ease knowledge inferencing and makes knowledge entering (in any notation or graphical interface following the RDF+OWL model) more difficult.

```
E:     Every human body has at most 2 arms. Every arm belongs to at most 1 body.
FL:    pm#human_body part: wn#arm __[every->0..2, 0..1<-every];
FE:    Every pm#human_body has for at most 2 wn#arm. Any arm is part of 0..1 wn#body];
FCG:   [every pm#human_body, part: 0..2 wn#arm]; [every arm, part of: at most 1 wn#body];
CGLF:   [pm#human_body: @forall]->(part)->[wn#arm: @0..2]
        [wn#arm: @forall]<-(part)<-[pm#human_body: @0..1]  //same approach in CGIF
KIF:   (forall ((?b pm#human_body)) (atMostN 2 '?a wn#arm (pm#part ?b '?a)))
       (forall ((?a wn#arm)) (atMostN 1 '?b pm#human_body (pm#part '?b ?a)))
RDOX:  <rdf:Property rdf:ID="armPart"><rdfs:subPropertyOf rdf:resource="&pm;part"/>
         <owl:inverseOf rdf:ID="armPartOf"/>
         <rdfs:range rdf:resource="&wn;Arm"/> </rdf:Property>
       <owl:Class rdf:about="&pm;HumanBody" pm:quantifier="every"><rdfs:subClassOf>
         <owl:Restriction><owl:onProperty rdf:resource="#armPart"/>
           <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">2
           </owl:maxCardinality></owl:Restriction> </rdfs:subClassOf></owl:Class>
       <owl:Class rdf:about="&wn;Arm" pm:quantifier="every"><rdfs:subClassOf>
         <owl:Restriction><owl:onProperty rdf:resource="#armPartOf"/>
           <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1
           </owl:maxCardinality></owl:Restriction> </rdfs:subClassOf></owl:Class>
RON3:  :armPart  a rdf:Property;            rdfs:subPropertyOf pm:part;
                 owl:inverseOf :armPartOf;  rdfs:range wn:Arm.
       pm:HumanBody  pm:quantifier "every";
          rdfs:subClassOf [a owl:Restriction;  owl:onProperty armPart;  owl:cardinality 2].
       wn:Arm  pm:quantifier "every";
          rdfs:subClassOf [a owl:Restriction;  owl:onProperty armPartOf;  owl:cardinality 1].
```

Below is a KIF definition for `atMostN' followed by a definition of similar relations, some of which are used above or below. Unlike OWL relation types for cardinalities, these types can also be used in a concept node that is source of a relation. Thus, they can be seen as numerical quantifiers.

```
(defrelation atMostN (?num ?var ?type ?predicate) :=
  (exists ((?s set)(?n)) (and (size ?s ?n) (=< ?n ?num)
    (truth ^(forall (,?var) (=> (member ,?var ,?s)
                                (and (,?type ,?var) ,?predicate)))))))

(defrelation exactlyN (?num ?var ?type ?predicate) :=
  (exists ((?s set)) (and (size ?s ?num)
    (truth ^(forall (,?var) (=> (member ,?var ,?s)
                                (and (,?type ,?var) ,?predicate)))))))

(defrelation exists1 (?var ?type ?predicate) :=
  (truth ^(exists (,?var)
    (and (,?type ,?var) (,?predicate ,?var)
         (forall(?y) (=> (,?predicate ?y) (= ,?var ?y)))))))

(defrelation exists0or1 (?var ?predicate) :=
  (truth ^(or (not (exists (,?var) (,?predicate ,?var)))
              (exists1 (,?var) (,?predicate ,?var)))))

(defrelation existsN (?op ?num ?var ?type ?predicate) :=
  (exists ((?s set)(?n)) (and (size ?s ?n) (?op ?n ?num)
    (truth ^(forall (,?var) (=> (member ,?var ,?s)
                                (and (,?type ,?var) ,?predicate)))))))
```

4.2.7. Qualifiers and Numerical Quantification via Percentages

The next example illustrates i) numerical quantification via "percentages on the number of instances of a concept type" and ii) the qualifiers `can' (in the sense of "physical possibility") and `good'.

In FL, FCG and FE, concept qualifiers (e.g., `good', `bad', `important') and relation qualifiers (e.g., `can', `more'/`less', `main', `1st', `2nd', ..., `last') are predefined keywords since they are common in natural language but hard to define.
Below, the proposed translations for 'can' use a meta-statement: a modality (a kind of attribute) on the described situation. Other kinds of qualifiers would require different (and more complex) kinds of representations, for example using lambda-abstractions which in KIF would require using variables for relation types. In any case, the representations would be ad-hoc (like those of the example below). Thus, for knowledge normalization, sharing and (lightweight) exploitation purposes, it is interesting that the used KRL includes these qualifiers and percentage-based quantifiers as keywords. When this is not possible, in the same way that RDFS or OWL are associated to RDF, the used KRL should have an associated ontology that includes types declaring or defining the quantifiers and qualifiers cited above and below (and their associated types of concepts or relations, e.g., measure, modality and physical_possibility).

The Lisp-based language of the Knowledge Machine [www-KM 2006] has intuitive keywords similar to those of FL, FCG and FE for quantifiers (e.g., `some', `a', `all', `at_most') but has no percentage-based numerical quantifiers.
In FE and FCG, the keyword `most' may be used and is equivalent to `at least 60%' (hence, it can be translated to other notations in this form).

```
E:     At least 85% of healthy birds are able to fly.
FL:    `bird experiencer of a good health'  can be agent of:  flight __[85%..100%->?];
FE:    At least 85% of `bird experiencer of a good health' can be agent of a flight.
FCG:   [at least 85% of (bird experiencer of: a good health), can be agent of: a flight]
CGLF:   [ [(lambda(*b) [bird:*b]<-(experiencer)<-[health]->(measure)->[good]): *z]: @>85%]
          <-(agent)<-[flight] ]->(modality)->[physical_possibility]  //same approach in CGIF
KIF:   (modality '(forAtLeastNpercent 85 '?x bird (exists ((?f flight)) (agent ?f '?x)))
                 physical_possibility)
RDOX:  <owl:Class rdf:ID="GoodHealth">
         <owl:intersectionOf rdf:parseType="Collection">
             <owl:Class rdf:about="Health"/>
             <owl:Restriction><owl:onProperty rdf:resource="#measure"/>
               <owl:allValuesFrom rdf:resource="#Good"/></owl:Restriction>
         </owl:intersectionOf></owl:Class>
       <owl:Class rdf:ID="HealthyBird">
         <owl:intersectionOf rdf:parseType="Collection">
             <owl:Class rdf:about="Bird"/>
             <owl:Restriction><owl:onProperty rdf:resource="#experiencerOf"/>
               <owl:allValuesFrom rdf:resource="#GoodHealth"/></owl:Restriction>
         </owl:intersectionOf></owl:Class>
       <owl:Class rdf:about="#HealthyBird" rdf:bagID="b" quantifier="at least 85%">
         <agentOf><Flight/></agentOf></owl:Class>
       <rdf:Description rdf:aboutEach="#b">
         <qualifier rdf#resource="#PhysicalPossibility"/> </rdf:Description>
RON3:  :GoodHealth owl:intersectionOf (:Health
          [a owl:Restriction;  owl:onProperty :measure;  owl:hasValue :Good]).
       :HealthyBird  owl:intersectionOf (:Bird
          [a owl:Restriction;  owl:onProperty :experiencerOf;  owl:allValuesFrom :GoodHealth].
       :85percentOfBird  rdfs:subClassOf :Bird;  quantifier "at least 85%".
       @forAll :b. { {:b rdf:type 85percentOfBird;  :agentOf :Flight }
                     :modality :PhysicalPossibility }.
```

Here is a KIF definition of `forAtLeastNpercent' (and its associate functions):

```
(defrelation forAtLeastNpercent (?n ?var ?type ?predicate) :=
  (exists ((?s set))
    (and (truth ^(forall (,?var) (=> (member ,?var ,?s) (,?type ,?var)))
         (>= (numMembersSuchThat ,?s ,?predicate) (/ (* (size ,?s) ?n) 100)))))

  (define-function numMembersSuchThat (?set ?p) :-> ?num :=
    (if (and (set ?set) (predicate ?p)) (numElemsSuchThat (listOf ?set) ?p)))

  (define-function numElemsSuchThat (?list ?p) :-> ?num :=
    (cond ((null ?list) 0)
          ((list ?list) (if ?p (1+ (numElemsSuchThat (rest ?list) ?p))))))
```

4.2.8. Simple Negations (Exclusions, Complements, Inverses, ...) and (X)OR-Collections

Two forms of negation have been presented above: one involving a `different_from' relation (`differentFrom' in OWL, `/=' in KIF), and one involving the negation of a sentence ('not' in KIF). This last form is more difficult to exploit by inference engines and leaves room for ambiguity. For example, "Tom does not own a blue car" may mean that "Tom owns a car that is not blue" or that "Tom does not own a car". Thus, it is better to use the first form or break sentences into smaller blocks connected by coreference variables in order to reduce or avoid ambiguities. The next example illustrates a variant of the first form: negation on types.

```
E:     Tom owns something that is not a car.
FL:    Tom owner of: (pm#thing != car __[?x->?]) __[.->?x].
FE:    Tom is owner of a !car.
FCG:   [Tom, owner of: a !car]
CGLF:  [T:Tom]<-(owner)<-[~car:*]
CGIF:  (owner [~car] Tom)
KIF:   (exists (?type ?x) (and (owner ?x Tom) (holds ?type ?x) (/= ?type car)))
RDOX:  <owl:Thing><owner><owl:Thing rdf:about="#Tom"/></owner>
                  <rdf:type><owl:Class><owl:differentFrom rdf:resource="#car"/>
                            </owl:Class> </rdf:type> </owl:Thing>
RON3:  a [a owl:Class;  owl:differentFrom :Car];  :owner :Tom.
```

OWL makes no "unique name assumption": objects are not assumed to be different when not related by identity relations, and hence unless related by `owl#differentFrom' relations. For ontology matching purposes, this choice has the (slight) advantage of permitting an inference engine to set identity relations between independently defined categories without such settings being some kind of "belief revision". However, this choice has strong drawbacks for knowledge modeling and inferencing purposes since i) almost all categories (even independently defined ones) are actually different, ii) it is practically impossible to manually set `owl#differentFrom' relations between all categories, and iii) the absence of such relations does not permit to make inferences that are important for knowledge checking, problem solving and ontology matching purposes. Thus, within WebKB-2 and, more generally, in a "collaboratively-built&evaluated global well-organized secure Semantic Web", this choice makes no sense. This is why in WebKB-2, objects are assumed to be different when not related by identity relations: this convention eases knowledge entering, storage and inferencing. However, a `different-from' may also be explicitly set between objects. Some details on the algorithmic complexity associated to identification constraints can be found in [Nguyen & Thanh, 2007].

Exclusion between objects (and hence, some forms of negation between types, individuals or statements) may also be represented via XOR-collections or OR-collections.
RDF proposes an `alt' collection to store alternatives but unfortunately does not specify if this 'or' is inclusive or exclusive. Although specializing `alt' by `or_bag' and `xor_set' seems a good idea (even if RDF engines are unlikely to take advantage of this distinction), the current RDF/XML grammar only permits to define members to collections of type `Bag', `Alt' and `Seq' (via rdf#li relations): it does *not* permit to define members to specializations of these types! Furthermore, it is unclear how collections on types are interpreted in RDF (distributive, collective or cumulative interpretation?). Hence, it is better to use relations such as owl#union_of, owl#one_of, owl#disjoint_with, owl#complement_of or owl#differentFrom.
Below is an example of OR-collection between individuals for colors. (The types `red', `yellow' and `orange' are subtypes of `color', and have many subtypes, e.g., `crimson, dark_red' and `chrome_red'. Instances of these types represent the actual occurrences of colors that physical objects have.)
There is no usual way to represent OR collections in CGLF; hence, 'OR{...}' is used, as in FCG, because this method is more generic than the method used in the CGIF translation.

```
E:     Tom's car is red, yellow or orange.
FL:    Tom  owner of: (a car  color: OR{a red, a yellow, an orange})
FE:    Tom is owner of a car that has for color OR{a red, a yellow, an orange}.
FCG:   [Tom, owner of: (a car, color: OR{a red, a yellow, an orange})]
CGLF:  [Tom]<-(owner)<-[car]->(chrc)->[color]->(kind)->[TYPE:OR{red,yellow,orange}]
CGIF:  [car:*x] (owner *x Tom) (color *x [red|yellow|orange:])
KIF:   (exists ((?x car) ?c)
         (and (owner ?x Tom) (color ?x ?c) (or (red ?c)(yellow ?c)(orange ?c))))
RDOX:  <Car><owner><owl:Thing rdf:about="#Tom"/></owner>
         <color><owl:Thing> <rdf:type><owl:Class><owl:unionOf rdf:parseType="Collection">
                                                  <owl:Class rdf:about="#Red"/>
                                                  <owl:Class rdf:about="Yellow"/>
                                                  <owl:Class rdf:about="Orange"/></owl:unionOf>
                            </rdf:type></owl:Thing></color></Car>
RON3:  a :Car;  :owner :Tom;   :color a [owl:unionOf (:Red :Yellow :Orange)].
```

Whenever possible, using types instead of collections is better since this leads to more concise representations that are more likely to be handled by inference engine. For example, in this last example, `warm_color' may have been an adequate type. For representing exclusions, it is preferable to use subtype partitions whenever possible. Below is an example of closed subtype partition. For an open subtype partition, in the notations other than FL, owl#disjoint_with should be used instead of owl#union_of. To express complements and inverses, the relation types owl#complement_of and owl#inverse_of (alias pm#complement_type and pm#inverse) can be used in all notations.

```
E:     According to OWL Lite (but not OWL DL/Full), owl#datatype_property and
       owl#object_property form a complete subclass partition of rdf#property.
FL:    rdf#property > {( owl#datatype_property __[owl_lite]
                         owl#object_property   __[owl_lite] )}  __[owl_lite];
FCG:   [rdf#property, owl#union_of: {owl#datatype_property,owl#object_property}(exclusive)
       ](owl_lite);  //same approach in CGLF and CGIF
KIF:   (dc#Creator '(owl#union_of rdf#property
                          (listof owl#datatype_property owl#object_property)
                    ) owl_lite)
RDOX:  <owl:Class rdf:about="&owl;Property">
          <owl:unionOf rdf:parseType="Collection" dc:Creator="owl_lite" >
             <owl:Class rdf:about="&owl;ObjectProperty"/>
             <owl:Class rdf:about="&owl;DatatypeProperty"/>  </owl:unionOf> </owl:Class>
       <owl:Class rdf:about="&owl;ObjectProperty">
          <owl:disjointWith rdf:resource="&owl;DatatypeProperty" dc:Creator="owl_lite"/>
       </owl:Class>
RON3:  {owl:Property owl:unionOf owl:ObjectProperty,owl:DatatypeProperty} dc:Creator "owl_lite".
       {owl:ObjectProperty owl:disjointWith owl:DatatypeProperty} dc:Creator "owl_lite".
```

4.2.9. Function Calls, Actors and Ordered Collections

Functional relation nodes need no special syntactic sugar for being distinguished from other relation nodes since their specificity can be specified via the relation type declaration. For example, KIF permits to define and re-use LISP functions. However, syntactical differences in the function node are sometimes used for distinguishing various kinds of functions, especially the declarative ones from those that have side-effects or can be "executed" by an external program. In CGLF and CGIF, these last functions are called "actors" and their nodes are delimited by angle brackets while other relation nodes are delimited by parenthesis.
Lists (ordered collections) are also supposed to have angle brackets according to the CG standard even though the CGIF grammar only specify curly brackets for all lists.
The next example illustrates a function ('length') using a list as parameter. In the CGLF and CGIF translations, the syntactic sugar for "actors" are used for representing the fact that this particular 'length' function is executed by an external program but, alternatively, a declarative definition could also be given to this 'length' function. The given translations in RDOX and RON3 are, at best, ad-hoc.

```
E:     The length of the list "Tom, Joe, Jack" plus 1 is inferior to 5.
FCG:   [(length _(LIST{Tom,Joe,Jack}) + 1)  <  5]  //idem in FE and FL; see Paragraph 2.1.1.6
CGLF:  [number:5]<-(superior)<-number:*x]<-<plus>- <-1- [number]<-<length><-[T: <Tom,Joe,Jack>],
                                                   <-2- [number:1]
CGIF:  <length [T: <Tom,Joe,Jack>] *1>  <plus ?1 1 [number]>
KIF:   (superior (+ (length (listof Tom Joe Jack)) 1)  5)
RDOX:  <Number><plus rdf:parseType="Collection">
                  <Number><length rdf:parseType="Collection"><Tom><Joe><Jack></length></Number>
                  "1"^^xsd:integer</plus>
               <superior>"5"^^xsd:integer</superior></Number>
RON3:  [:plus [:length (:Tom :Joe :Jack)] "1"^^xsd:integer] :superior "5"^^xsd:integer;
```

The CG standard does not specify how to define functional relations, just how to use them. The next example is adapted from [Sowa, 1993]. No translation of the definition body in RDOX and RON3 could be found.

```
E:     The length of a list is equal to 0 if the list is empty, and otherwise is
       recursively equal to 1 + "the length of the list without its first element"
FL:    //The next FL translation is similar to the FCG one. This first one uses a concept type
       //  from which the function type (and hence the next FL translation) can be derived.
       Length .(list *l -> natural *r)
         := [{ [*r = 0]_[<= [*l = nil]]   [*r = 1 + length_(rest_(*l))]_[<= [*l != nil]]
             }_[kind: XOR-set]];
FL:    length .(list *l -> natural *r)
         := [if [*l = nil] then [*r = 0] else [*r = 1 + length_(rest_(*l))] ]]
FCG:   [function length .(list *l -> natural *r)
         := [if [*l = nil] then [*r = 0] else [*r = 1 + length_(rest_(*l))] ]]
CGLF:  [function length (list *l, natural *n)
         [IF: [?l]->(EQ)->[list: nil] [THEN: [?n]->(EQ)->[number:0] ]
           [ELSE: [?l]-><rest>->[list]-><length>->[natural]-><plus1>->[?n] ] ]
CGIF:  [function length (list *l, natural *n)
         [IF: (EQ ?l nil) [THEN: (EQ ?n [number:0])]
           [ELSE: (rest ?l [list:*l2])(length ?l2 [natural:?n2])(plus1 ?n2 ?n)]
         ]]
KIF:   (deffunction length (?l)
         := (if (= ?l nil) 0  (if (list ?l) (1+ (length (rest ?l))))))
```

KIF also has built-in operators (e.g., `listOf' and 'setOf') to assemble or decompose lists and sets, as in the following statement. The variable '@items' refers to the elements of the list stored in the variable '?items'. This approach can also be used in FL, FCG and FE but is not used in the other notations.

```
(deffunction first (?l)  :=  (if (= (listof ?x @items) ?l)  ?x))
```

4.2.10. Higher-order Statements

First-order statements only permit universal quantification over individuals. Higher-order statements also permit universal quantification over types. A first-order statement is sufficient to specify the transitivity of one particular relation, e.g., "the part of a part is also a part", but the definition of pm#transitive_binary_relation_type (owl#transitive_property) requires a second-order statement. As opposed to the previous examples, no "lightweight" extension to OWL would be sufficient for representing owl#transitive_property. Hence, below it is represented using a "big possible extension to RDF" that [Berners-Lee, 1999] describes: a `forall' construct with attributes such as `pname' to quantify over classes. The CGLF translation is adapted from a representation given [Sowa, 1993]: '&' is used instead of 'ρ' as a prefix for a second-order variable since KRLs should only require easy-to-enter characters. An alternative is to keep the usual variable prefix '?' or '*' since there is no ambiguity on what the variable refers to. This is the solution adopted in FE and FCG. Free variables, i.e., variables that are implicitly universally quantified, are used in the FCG, CGLF, CGIF and KIF translations. In FCG and FE, they are prefixed by '^'.

```
E:     A pm#transitive_binary__relation_type rt is such that
         if x is connected to y by a relation of type rt, and
            y is connected to z by a relation of type rt,
         then x is connected to z by a relation of type rt.
FE:    `pm#transitive_binary__relation_type *rt has for definition
         `if `^x has for *rt ^y that has for *rt ^z'
          then `^x has for *rt ^z' '.  //x,y,z are "free variables", i.e., variables that are
           //implicitly universally quantified (FCG, CGLF, CGIF and KIF also accept them)
FCG:   [type pm#transitive_binary__relation_type .(*rt) :=
         [ [^x, *rt: (^y, *rt: ^z)] => [^x, *rt: ^z] ] ]
CGLF:  [type pm#transitive_binary__relation_type (*rt) :=   //adapted from [Sowa, 1993]
         [IF: [T: *x]->(&rt)->[T: *y]->(&rt)->[T: *z]  [THEN: [?x]->(&rt)->[?z]] ] ]
```

```
CGIF:  [IF: [pm#transitive_binary__relation_type *rt])
         [THEN: [IF: (holds ?rt [T:*x] [T:*y]) (holds ?rt ?y [T:*z])
                  [THEN: (holds ?rt ?x ?y)] ] ] ]  //'holds' is used as in KIF
KIF:   (defrelation transitive_binary__relation_type (?rt) :=
         (forall (?x ?y ?z) (=> (and (?rt ?x ?y) (?rt ?y ?z)) (?rt ?x ?z))))
       //to be accepted in all versions of KIF, the last line should rather be written:
       //(forall (?x ?y ?z) (=> (and (holds ?rt ?x ?y)(holds ?rt ?y ?z))(holds ?rt ?x ?z))))
RDF+:  <forall var="r" v2="x" v3="y" v4="z"> <!--var,v2,v3,v4: new attributes-->
         <if><&own;TransitiveProperty rdf:about="#rt">
         <then><if><rdf:Description about="#x">
                     <rdf:property pname="#rt"><!-- pname: new attribute -->
                       <rdf:Description about="#y">
                         <rdf:property pname="#rt"><rdf:Description about="#z"/>
                         </rdf:property></rdf:Description>
                     </rdf:property></rdf:Description>
               <then><rdf:Description about="#x">
                       <rdf:property pname="#rt"><rdf:Description about="#z"/>
                       </rdf:property></rdf:Description>
               </then></if></then></if></forall>
RON3:  @forAll :rt .{ :rt a owl:TransitiveProperty. } log:implies
       { @forAll :x, :y, :z.  { :x :rt :y.  :y :rt :z. } log:implies { :x :rt :z. }.
       }.  <!-- here, :rt is not a type but a variable for a type -->
```

4.2.11. Relations from Collections, Collection Interpretations and Quantifier Precedence

This subsection shows how various interpretations of the English sentence *"9 judges have approved 50 laws"* (and some variations of it) can be interpreted and represented. This study of how relations between members of two simple collections can be represented illustrates the importance of specifying how a collection must be interpreted and shows how to handle complex cases of quantifier precedence (between numerical, existential and universal quantifiers).

The sentence *"9 judges have approved 50 laws"* is ambiguous. The 9 judges may have individually or collectively approved 50 laws (the same 50 or not); "collectively" may itself have two meanings: a participation to a "unique" approval act or the approval of "most" of the laws (or a combination of both as illustrated in the last example of this section). This study shows that all this leads to *at least* 32 different logical interpretations. In this subsection, "judges approving together/collectively" means that "there exists a (unique) approval and each of the judges is agent of that approval". Similarly, "collectively approved laws" means that "there exists an approval and each of the laws are object of that approval". This interpretation of "collectiveness" was used in [Sowa, 1992] and, in CG terminology, it implies that the approval must be represented by a "concept node", not by a "relation node" (this was however not made explicit by Sowa). In [Sowa, 1992], any collection in a concept node of a CG can be specified as having a *distributive interpretation* (each member of the collection individually participates to the relations associated to the node), a *collective interpretation* (the members collectively participate to the relations associated to the node), a *default interpretation* (an unspecified mix of collective and distributive interpretation) or a *cumulative interpretation* (the relations are about the collection itself). The current CG standard does not specify what the various interpretations of a collection can be, not even within the CGIF grammar; it mentions the keyword 'Col' as a "collective designator" but not the keywords 'Dist' and 'Cum' used in [Sowa, 1992].

RDF/XML introduced some notion of distributive interpretation via the keyword rdf#aboutEach (which was removed from RDF/XML in January 2003) but it is not clear if this keyword referred to the above cited "default interpretation", to the direct distributive interpretation or to the general distributive interpretation (the distinction between the last two will be explained below). Without rdf#aboutEach, the relations are about the collection itself (cumulative interpretation). However, in their examples, the RDF authors also represent the collective interpretation via direct relations to the bag (e.g., see Section 4.1 of [www-RDF/XML 2004]). For the OWL translations below, the concept type pm#set and the relation types pm#interpretation .(?, rdfs#Literal) and pm#size .(pm#collection -> kif#natural) are used. The type `set' is declared as a subtype of `rdf#bag' and `rdf#about_each' is used on sets.

The first example keeps the ambiguity of the above cited sentence, i.e., both collections have the default interpretation. The 's' at the end of `judges' and `laws' in the FE and FCG representations is assumed to be automatically removed (WebKB-2 does this when a numerical quantifier is used before a type name). In this subsection, only the first example has a FE translation. However, all the examples have Predicate Logic (PL) translations. These PL translations use the symbol "∀" for the universal quantifier, "∃" for the existential quantifier, "∈" for membership to a set, and "^" for "and".
In CGLF and CGIF, the order of the quantifiers at a same level in a context is specified via a simple convention: the existential quantifier marked by the keyword `@certain' has precedence over the universal/numerical quantifiers which have precedence over the over existential quantifiers. In the general case, remaining ambiguities have to be solved by the CGLF/CGIF user via the addition of contexts for specifying the scope of variables. In FE and FCG, the order of the quantifiers is specified by their order of appearance in the entered statement.

```
E:     9 judges have (each/together) approved 50 laws.
FE:    9 judges are agent of an approval with object 50 laws.
FCG:   [9 judges, agent of: (an approval, object: 50 laws)]
CGLF:  [judge:{*}@9 @certain]<-(agent)<-[approval]->(object)->[law:{*}@50]
CGIF:  (agent [approval:*a] [judge: @9 @certain]) (object ?a [law: {}@50])
KIF:   (forAllN 9 '?j judge (forAllN 50 '?l law
         (exists ((?a approval)) (and (agent ?a '?j) (object ?a '?l)))))
PL:    ∃js set(js)^size(js,9) ^ ∀j∈js  ∃ls set(ls)^size(ls,50) ^ ∀l∈ls
         ∃a approval(a) ^ agent(a,j) ^ object(a,l)
RDOX:  <rdf:Property rdf:ID="objectLaw">
         <rdfs:subPropertyOf rdf:resource="#object"/>
         <rdfs:range rdf:resource="#Law"/> </rdf:Property>
       <owl:Class rdf:ID="ApprovalOf50laws">
         <owl:intersectionOf rdf:parseType="Collection">
            <Approval/> <owl:Restriction>
                          <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">50
                          </owl:cardinality>
                          <owl:onProperty rdf:resource="#objectLaw"/></owl:Restriction>
         </owl:intersectionOf></owl:Class>
       <Set rdf:ID="Judges"><size>9</size></Set>
       <rdf:Description rdf:aboutEach="#Judges" interpretation="separately">
         <rdf:type rdf:resource="#Judge"/></rdf:Description>
       <rdf:Description rdf:aboutEach="#Judges" interpretation="default">
         <agentOf><ApprovalOf50laws/></agentOf></rdf:Description>
RDOX:  <!-- This second version is shorter but highly tentative and hence won't be used again.
            It mostly relies on the special meaning of the relation pm#interpretation.
            RDF does not specify the meaning of embedded uses of rdf#aboutEach. -->
       <Set rdf:ID="Laws"/>
       <rdf:Description rdf:aboutEach="#Laws" interpretation="separately">
          <rdf:type rdf:resource="#Law"/></rdf:Description>
       <Set rdf:ID="Judges"><size>9</size></Set>
       <rdf:Description rdf:aboutEach="#Judges" interpretation="separately">
          <rdf:type rdf:resource="#Judge"/></rdf:Description>
       <rdf:Description rdf:aboutEach="#Judges" interpretation="default">
          <agentOf><Approval><object>
            <rdf:Description rdf:aboutEach="#Laws" interpretation="default"> <size>50</size>
           </rdf:Description></object></Approval></agentOf></rdf:Description>
       <!-- OWL may permit to define Judges as a set without introducing
            the class Set, in the following way:
            <owl:AllDifferent><owl:distinctMembers><rdf:Bag rdf:ID="Judges"/>
                              </owl:distinctMembers></owl:AllDifferent> -->
```

```
RON3:  :objectLaw a rdf:Property;  rdfs:subPropertyOf :object;  rdfs:range :Law.
       :ApprovalOf50laws a owl:Class;  owl:intersectionOf :Approval,
                            [a owl:Restriction; owl:onProperty :objectLaw; owl:cardinality 50].
       :Judges a Set;  size 9.    @forAll :j. { :j  a :Judge;  is pm:member of :Judges }
                                          => { [a :ApprovalOf50laws] :agent :j }.
```

Construct introduced in the previous KIF translation:

```
  (defrelation forAllN (?num ?var ?type ?predicate) :=
    (exists ((?s set)) (and (size ?s ?num)
      (truth ^(forall (,?var) (=> (member ,?var ,?s)
                                  (and (,?type ,?var) ,?predicate)))))))
  //forAllN is nearly identical to the previously defined atMostN: only the
  //constraint on the size of the set differs
```

The next example shows a simple inversion of the quantifier order.

```
E:     50 laws have been approved by 9 judges (each/together).
FCG:   [50 laws, object of: (an approval, agent: 9 judges)]
CGLF:  [law:{*}@50 @certain]<-(agent)<-[approval]->(object)->[judge:{*}@9]
CGIF:  (agent [approval:*a] [judge: @9]) (object ?a [law: {}@50 @certain])
KIF:   (forAllN 50 '?l law (forAllN 9 '?j judge
         (exists ((?a approval)) (and (agent ?a '?j) (object ?a '?l)))))
PL:    ∃ls set(ls) ^ size(ls,50) ^ ∀l∈ls  ∃js set(js)^size(js,9) ^ ∀j∈js
         ∃a approval(a) ^ agent(a,j) ^ object(a,l)
RDOX:  <rdf:Property rdf:ID="agentJudge">
         <rdfs:subPropertyOf rdf:resource="#agent"/>
         <rdfs:range rdf:resource="#Judge"/> </rdf:Property>
       <owl:Class rdf:ID="ApprovalBy9Judges">
         <owl:intersectionOf rdf:parseType="Collection">
           <Approval/>
           <owl:Restriction>
             <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">9 </owl:cardinality>
             <owl:onProperty rdf:resource="#agentJudge"/></owl:Restriction>
         </owl:intersectionOf></owl:Class>
       <Set rdf:ID="Laws"><size>50</size></Set>
       <rdf:Description rdf:aboutEach="#Laws" interpretation="separately">
         <rdf:type rdf:resource="#Law"/></rdf:Description>
       <rdf:Description rdf:aboutEach="#Laws" interpretation="default">
         <objectOf><ApprovalBy9Judges/></objectOf></rdf:Description>
RON3:  :agentJudge a rdf:Property;  rdfs:subPropertyOf :object;  rdfs:range :Judge.
       :ApprovalBy9Judges a owl:Class;
                         owl:intersectionOf :Approval,
                            [a owl:Restriction; owl:onProperty :agentJudge; owl:cardinality 9].
       :Laws a Set;  size 50.
       @forAll :l. {:l a :Law;  is pm:member of :Laws} => {[a :ApprovalBy9judges] :object :l}.
```

In FE and FCG, the ***collective interpretation*** is specified via the keywords `together', `group of', `set of', `bag of', `list of', `sequence of' or `alternatives' (the first three are synonyms; in this subsection `set' is used). In CGLF and CGIF, the keyword `Col' is used, and the collection is assumed to be a set.

If we take the two previous examples and gradually introduce the collective interpretation for the collections, that is, the sharing of the approvals by the judges or laws (e.g., in the KIF translations, by gradually moving "(exists ((?a approval)" to the left), we obtain five different logical interpretations (instead of six because when both collections are collectively interpreted, the inversion of quantifier order does not change the meaning). Below are three of these combinations (the two other ones are: "A group of 50 laws has been approved by 9 judges" and "A group of 9 judges has approved 50 laws").

```
E:     9 judges have (each/together) approved a group of 50 laws.
FCG:   [9 judges, agent of: (an approval, object:  50 laws together)]
CGLF:  [judge:{*}@9 @certain]<-(agent)<-[approval]->(object)->[law:Col{*}@50]
CGIF:  (agent [approval:*a] [judge:@9]) (object ?a [law:@Col{}@50 @certain])
KIF:   (forAllN 9 '?j judge (exists ((?a approval) (?ls set))
         (forAllIn ?ls 50 '?l law (and (agent ?a '?j) (object ?a '?l)))))
PL:    ∃js set(js)^size(js,9) ^ ∀j ∈js  ∃a approval(a) ^
         ∃ls set(ls)^size(ls,50) ^ ∀l∈ls  agent(a,j) ^ object(a,l)
RDOX:  <owl:Class rdf:ID="ApprovalOf50laws">
         <owl:intersectionOf rdf:parseType="Collection" interpretation="collective">
           <Approval/> <owl:Restriction>
                         <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">50
                         </owl:cardinality>
                       <owl:onProperty rdf:resource="#objectLaw"/></owl:Restriction>
         </owl:intersectionOf></owl:Class>
       <Set rdf:ID="Judges"><size>9</size></Set>
       <rdf:Description rdf:aboutEach="#Judges" interpretation="separately">
         <rdf:type rdf:resource="#Judge"/></rdf:Description>
       <rdf:Description rdf:aboutEach="#Judges" interpretation="default">
         <agentOf><ApprovalOf50laws/></agentOf></rdf:Description>
RON3:  :Judges a Set; size 9.
       @forAll :j.  { :j a :Judge; is pm:member of :Judges }
          => { @exist :a.  {:a a :approval.    :Laws at set; size 50}
                 => { @forAll :l.  { :l a :Law; is pm:member of :Laws}
                        => {:a :agent :j; :object :l}
                    } }
```

Here is a definition for the "quantifier" `forAllIn' used in the KIF translations above and below.

```
  (defrelation forAllIn (?s ?num ?var ?type ?predicate) :=
    (and (size ?s ?num)
         (truth ^(forall (,?var) (=> (member ,?var ,?s)
                                     (and (,?type ,?var) ,?predicate))))))
```

```
E:     50 laws have been approved by a group of 9 judges.
FCG:   [50 laws, object of: (an approval, agent: 9 judges together)]
CGLF:  [judge:Col{*}@9]<-(agent)<-[approval]->(object)->[law:{*}@50 @certain]
CGIF:  (agent [approval:*a] [judge:@Col{}@9]) (object ?a [law:@50{} @certain])
KIF:   (forAllN 50 '?l law (exists ((?a approval) (?js set))
         (forAllIn ?js 9 '?j judge (and (agent ?a '?j) (object ?a '?l)))))
PL:    ∃ls set(ls)^size(ls,50) ^ ∀l ∈ls  ∃a approval(a) ^
         ∃js set(js)^size(js,9) ^ ∀j∈js  agent(a,j) ^ object(a,l)
RDOX:  <Set rdf:ID="Judges"/>
       <rdf:Description rdf:aboutEach="#Judges" interpretation="separately">
         <rdf:type rdf:resource="#Judge"/></rdf:Description>
       <Set rdf:ID="Laws"><size>50</size></Set>
       <rdf:Description rdf:aboutEach="#Laws" interpretation="separately">
          <rdf:type rdf:resource="#Law"/></rdf:Description>
       <rdf:Description rdf:aboutEach="#Laws" interpretation="default">
         <objectOf><Approval><agent>
           <rdf:Description rdf:aboutEach="#Judges" interpretation="collective">
              <size>9</size> </rdf:Description></agent></Approval></objectOf></rdf:Description>
RON3:  :Laws a Set; size 50.
       @forAll :l.  {:l a :Law; is pm:member of :Laws}
         => {@exist :a.  @exist :Judges. {:a a :approval}
               => {@forAll :j. {:j a :Judge; is pm:member of :Judges.  :Judges a Set; size 9.}
                           => {:a :agent :j; :object :l}
                  } }
```

```
E:     A group of 9 judges has approved a group of 50 laws.
FCG:   [9 judges together, agent of:(an approval,object: 50 laws together)]
FCG:   [a set of 50 laws, object of:(an approval,agent:a set of 9 judges)]
FCG:   [an approval, agent: a set of 9 judges, object: a set of 50 laws]
CGLF:  [judge: Col{*}@9]<-(agent)<-[approval]->(object)->[law: Col{*}@50]
CGIF:  (agent [approval: *a] [judge: @Col{}@9])  (object ?a [law: @Col{}@50 @certain])
KIF:   (exists ((?a approval) (?js set) (?ls set))
         (forAllIn ?js 9 '?j judge (forAllIn ?ls 50 '?l law
           (and (agent ?a '?j) (object ?a '?l)))))
PL:    ∃a approval(a) ^ ∃js set(js)^size(js,9) ^
         ∃ls set(ls)^size(ls,50) ^  ∀j∈js ∀l∈ls agent(a,j) ^ object(a,l)
RDOX:  <Set rdf:ID="Judges"><size>9</size></Set>
       <rdf:Description rdf:aboutEach="#Judges" interpretation="separately">
         <rdf:type rdf:resource="#Judge"/></rdf:Description>
       <Set rdf:ID="Laws"><size>50</size></Set>
       <rdf:Description rdf:aboutEach="#Laws" interpretation="separately">
          <rdf:type rdf:resource="#Law"/></rdf:Description>
       <rdf:Description rdf:aboutEach="#Judges" interpretation="collective">
         <agentOf><Approval><object>
           <rdf:Description rdf:aboutEach="#Laws" interpretation="collective"/>
                         </object></Approval></agentOf></rdf:Description>
RON3:  :Judges a Set; size 9.  :Laws a Set; size 50.
       @exist :a. {:a a :approval}
         => { @forAll :j.  @forAll :l.  { :j a :Judge; is pm:member of :Judges}
              => {:a :agent :j; :object :l.   :l a :Law; is pm:member of :Laws}  }
```

In [Sowa, 1992], the *distributive interpretation* (i.e., where each member of the collection individually participates to the relations associated to the collection) specifies that each member is connected to different objects, directly (i.e., via one relation) or indirectly (i.e., via two or more relations within the statement), unless otherwise specified. For better precision, FE and FCG distinguishes the directly distributive interpretation (which can be specified via the keyword "each") from Sowa's general distributive interpretation (which can be specified via the keyword "separately"). If the directly distributive interpretation is introduced into the previous seven combinations, nine different logical interpretations are obtained. A few more are obtained when the general distributive interpretation is introduced; below are are two of them. In the KIF (resp. PL) translations, replacing the first `exists1For' (resp. ∃!!) by `exists' (resp. ∃) leads to the direct distributive interpretation. In CGLF, the keyword `Dist' is used. The CG standard does not address this issue but allows `@Dist' in CGIF.

Here is a KIF definition of `exists1For' ("∃!!" in PL). This quantifier permits to specify that the judges are agent of different approvals of different laws (first example) or groups of laws (second example).

```
(defrelation exists1For (?var1 ?var2 ?type ?predicate) :=
    (truth ^(exists (,?var2)
       (and (,?type ,?var2)  (,?predicate ,?var1 ,?var2)
         (forall (?x) (=> (,?predicate ,?var1 ?x) (= ,?var2 ?x)))
         (forall (?y) (=> (,?predicate ?y ,?var2) (= ,?var1 ?y)))))))
```

```
E:     9 judges have separately approved 50 laws (that's 450 approvals of law)
FCG:   [separately 9 judges, agent of: (an approval, object: 50 laws)]
CGLF:  [judge: Dist{*}@9]<-(agent)<-[approval]->(object)->[law:{*}@50]
CGIF:  (agent [approval:*a] [judge: @Dist{}@9]) (object ?a [law:{}@50])
KIF:   (forAllN 9 '?j judge (exists1For '?j '?ls set (forAllIn '?ls 50 '?l law
         (exists1For '?j '?a approval (and (agent '?a '?j) (object '?a '?l))))))
PL:    ∃js set(js)^size(js,9) ^ ∀j∈js   ∃!!ls set(ls)^size(ls,50) ^
         ∀l∈ls  ∃!!a approval(a) ^ agent(a,j) ^ object(a,l)
RDOX:  <Set rdf:ID="Laws"/>
       <rdf:Description rdf:aboutEach="#Laws" interpretation="separately">
          <rdf:type rdf:resource="#Law"/></rdf:Description>
       <Set rdf:ID="Judges"><size>9</size></Set>
       <rdf:Description rdf:aboutEach="#Judges" interpretation="separately">
         <rdf:type rdf:resource="#Judge"/>
         <agentOf><Approval><object>
           <rdf:Description rdf:aboutEach="#Laws" interpretation="default">
              <size>50</size>
           </rdf:Description></object></Approval></agentOf></rdf:Description>
RON3:  :Judges a Set; size 9.
       @forAll :j.  { :j a :Judge; is pm:member of :Judges }
           => { :Laws a Set; size 50.
                @forAll :l.  { :l a :Law; is pm:member of :Laws}
                  => { @exist :a. {:a a :approval} => {:a :agent :j; :object :l}
                       {:a :agent ?j; :object :l} => {?j = :j}
                       {:a :agent :j; :object ?l} => {?l = :l}
                     } }
```

```
E:     9 judges have separately approved a group of 50 laws (that's 450 approvals of law)
FCG:   [separately 9 judges, agent of: (an approval, object: a set of 50 laws)]
CGLF:  [judge:Dist{*}@9]<-(agent)<-[approval]->(object)->[law:Col{*}@50]
CGIF:  (agent [approval:*a] [judge:@Dist{}@9]) (object ?a [law:@Col{}@50])
KIF:   (forAllN 9 '?j judge (exists1For '?j '?ls set (exists1For '?j '?a approval
         (forAllIn '?ls 50 '?l law (and (agent '?a '?j) (object '?a '?l))))))
PL:    ∃js set(js)^size(js,9) ^ ∀j∈js   ∃!!ls set(ls)^size(ls,50) ^
         ∃!!a approval(a)  ∀l∈ls  agent(a,j)^ object(a,l)
RDOX:  <Set rdf:ID="Laws"/>
       <rdf:Description rdf:aboutEach="#Laws" interpretation="separately">
          <rdf:type rdf:resource="#Law"/></rdf:Description>
       <Set rdf:ID="Judges"><size>9</size></Set>
       <rdf:Description rdf:aboutEach="#Judges" interpretation="separately">
         <rdf:type rdf:resource="#Judge"/>
         <agentOf><Approval><object>
           <rdf:Description rdf:aboutEach="#Laws" interpretation="collective">
              <size>50</size>
           </rdf:Description></object></Approval></agentOf></rdf:Description>
RON3:  :Judges a Set; size 9.
       @forAll :j.  { :j a :Judge; is pm:member of :Judges }
           => { :Laws a Set; size 50.  @exist :a.
                @forAll :l.  { :l a :Law; is pm:member of :Laws}
                  => {  {:a a :approval; :agent :j; :object :l}
                        {:a :agent ?j; :object :l} => {?j = :j}
                        {:a :agent :j; :object ?l} => {?l = :l}
                     }
              }
```

Finally, the quantifier "most" can also be introduced as an interpretation of collectiveness in each of the previous (7+9=16) combinations (hence, 16 logical interpretations again). Here is one of them.

```
E:     A group of 50 laws has been approved by most in a group of 9 judges.
FCG:   [a group of 9 judges, agent of:
         (an approval, object: most in a group of 50 laws)]
FCG:   [most in a group of 50 laws, object of:
         (an approval, agent: a group of 9 judges)]
CGLF:  [judge:Col{*}@9]<-(agent)<-[approval]->(object)->[law:Col{*}@50 @most]
CGIF:  (agent [approval:*a] [judge:@Col{}@9])(object ?a [law:@Col{}@50 @most])
KIF:   (exists ((?a approval) (?js set) (?ls set))
         (forAllIn ?js 9 '?j judge (forMostIn ?ls 50 '?l law
           (and (agent ?a '?j) (object ?a '?l)))))
PL:    ∃a approval(a) ^  ∃js set(js)^size(js,9) ^  ∃ls set(ls)^size(ls,50) ^
         ∀j∈js agent(a,j) ^  ∃mostOfls set(mostOfls)
           (∀l∈ls  (object(a,l) => l∈mostOfls))   ^  size(mostOfls) >= 2
                                                 // >=2 since size(ls)/2=1.5
RDOX:  <Set rdf:ID="Judges"/>
       <rdf:Description rdf:aboutEach="#Judges" interpretation="separately">
         <rdf:type rdf:resource="#Judge"/></rdf:Description>
       <Set rdf:ID="Laws"><size>50</size></Set>
       <rdf:Description rdf:aboutEach="#Laws" interpretation="separately">
          <rdf:type rdf:resource="#Law"/></rdf:Description>
       <rdf:Description rdf:aboutEach="#Laws" interpretation="collective">
         <objectOf><Approval><agent>
           <rdf:Description rdf:aboutEach="#Judges" interpretation="collective"
                                                    quantifier="most">
              <size>9</size>
           </rdf:Description></agent></Approval></objectOf></rdf:Description>
RON3:  @exist :a. {:a a :approval}  :Judges a Set; size 9.  :Laws a Set; size 50.
       @forAll :j.  { :j a :Judge; is pm:member of :Judges }
           => { {:a :agent :j;}.  @forSome :mostOfls. :mostOfls a :Set; >= 2.
                @forAll :l.  { :l a :Law; is pm:member of :Laws}
                  =>{ {:a :object :l} => {:l is pm:member of :mostOfls} }
              }
```

Here is a KIF definition of `forMostIn' (the definition of `numMembersSuchThat' was given earlier).

```
(defrelation forMostIn (?set ?num ?var ?type ?predicate) :=
  (and (size ?set ?num)
       (truth ^(forall (,?var) (=> (member ,?var ,?set) (,?type ,?var))))
       (>= (numMembersSuchThat ,?set ,?predicate) (* (size ,?set) 0.6))))
```

4.2.12. Quantitative Valuation (Measures, Intervals, Temporal Entities, ...)

The representation of measures – e.g., "a weight of 923.5 kg" – is a "content ontology" issue (such as how to represent relations from states, processes or statements) but also a "language ontology" issue (such as quantification and contextualization) since for knowledge sharing purposes a language should provide simpler ways to represent such basic kinds of information than via the classic use of i) a ternary relation type `measure_3' or ii) a concept type such as wn#measure and two relation types such as wn#unit and wn#value. The value should be represented as if it was a numerical quantifier for the unit. One way would be to consider it as such and adapt the KIF definition of numerical quantifiers such as atMostN to accept any number, not just a non-negative integer. To avoid such a stretch of the notion of numerical quantification, it seems preferable to consider "923.5 kg" as a shortcut for any of the two above cited classic representation of measures. The next example illustrates this. Three equivalent FCG translations are given because WebKB-2 allows any subtype of pm#attribute_or_quality_or_measure (e.g., wn#weight and wn#color) to be

used in a relation node. An advantage of the approach is that the FCGs [a car, wn#weight: 923.5 wn#kg], [a car, wn#color: a wn#red] and [a car, wn#color: 400E+12 to 484E+12 wn#Hertz] are similarly structured and hence can be easily compared (the first FCG is correct since wn#kg is subtype of wn#weight and pm#unit_of_measure, the second is correct since wn#red is a subtype of wn#color, and the third is correct because in WebKB-2 it is equivalent to [a car, wn#color: (a color, pm#measure: 400 to 484 wn#Hertz] since wn#Hertz is a subtype of pm#unit_of_measure which is subtype of pm#attribute_or_quality_or_measure); wn#red is a measure equivalent to 400-484 teraHz for the frequency of light; more precisely, it is a type of measures since it has subtypes, e.g., wn#dark_red.

```
E:     There is a car that weights 923.5 kg.
FE:    There is a car that has for wn#weight 923.5 kg.
FCG:   [a car, weight: 923.5 kg]
FCG:   [a car, pm#attribute_or_quality_or_measure: 923.5 kg]
FCG:   [a car, attribute: (a weight, pm#measure: (a wn#measure, unit: a kg, value: 923.5))]
CGLF and CGIF:  //no standard -> same approach as in FCG
KIF:   (exists ((?c car)(?w weight)(?m wn#measure))
         (and (attribute ?c ?w) (pm#measure ?w ?m) (unit ?m kg) (value ?m 923.5)))
RDOX:  <Car><attribute><Weight><&pm;measure><&wn;Measure><unit rdf:resource="#kg"/>
                                                    <value>923.5</value></&wn;Measure>
                              </&pm;measure></Weight></weight></Car>
RON3:  a :Car; :attribute [a :Weight; pm:measure [a wn:Measure; :unit :kg; :number "923.5"] ].
```

Similarly, intervals have different representations depending on their meaning: number of objects, measure or collection of 2 numbers. In this last case, the use of a collection is sufficient. Here is an example for the first two cases.

```
E:     2 to 3 persons are running for 45.5 minutes to an hour.
FE:    2 to 3 persons is agent of a run with duration 45.5 to 60 minutes.
FCG:   [2 to 3 persons, agent of: (a run, duration: 45.5 to 60 minutes)]
CGLF:  [person: {*}@2-3]<-(agent)<-[run]->(duration)->[wn#time_period]-
           { ->(unit)->[minute], ->(value)->[number]- { ->(inferior)->[number: 45.5],
                                                        <-(inferior)<-[number: 60] } }
CGIF:  //same approach as in CGLF
KIF:   (forAllInBetween 2 3 '?p person
         (exists ((?r run) (?t wn#time_period) (?n number))
           (and (agent ?r '?p) (duration ?r ?t) (unit ?t ?minute) (value ?t ?n)
                (>= ?n 45.5) (=< ?n60))))
RDOX:  <Set rdf:ID="persons"/>
       <Set rdf:ID="Persons"><size><Number><atLeast>2</atLeast>
                                    <atMost>3</atMost><Number></size></Set>
       <rdf:Description rdf:aboutEach="#Persons" interpretation="default">
         <rdf:type rdf:resource="#Person"/>
         <agentOf><Run><duration><&wn;TimePeriod>
               <unit rdf:resource="#minute"/>
               <value><Number><inferior>"45.5"^^xsd:real</inferior>
                             <superior>"60"^^xsd:real</superior><Number></value>
         </agentOf></Run></duration></TimePeriod></rdf:Description>
RON3:  :Persons a Set; :size [a :Number; atLeast 2; atMost 3].
       @forAll :p. {:p a :Person; is :member of :Persons}
          => { [a :Run;  agent :p;  :duration
                  [a TimePeriod;  unit :minute;  :value
                    [a :Number;  :inferior "45.5"^^xsd:real;  :superior "60"^^xsd:real] ] ] }
```

Here is a KIF definition of `forAllInBetween'.

```
(defrelation forAllInBetween (?s ?n1 ?n2 ?var ?type ?predicate) :=
   (exists (?n) (and (size ?s ?n) (>= ?n ?n1) (=< ?n ?n2)
      (truth ^(forall (,?var) (=> (member ,?var ,?s) (and (,?type ,?var) ,?predicate))))))
```

FL, FCG and FE recognize various expressions for dates (e.g., "21/01/2001 18:38:20 GMT", "Jan 21 2001" and "2001/01/21") and can compare them. Since they are useful for knowledge representation and sharing, and are easy to parse, other notations should have them too.

4.2.13. Use of Concept Types in Relations; Generation of Relation Types From Concept Types

The way and interest of allowing certain kinds of concept types to be used in relation nodes have been described in Paragraph 2.1.1.14, Subsection 2.3.2 ("Structural or Semantic Normalization") and Subsection 3.1.2 ("Minimizing re-categorization – Examples with DOLCE"). Below are examples of how this can be made possible with KIF. Exceptionally, i) the KIF relation type `subrelation_of' is used instead of `supertype' between two relation types, and ii) the identifiers of concept types begin with an uppercase character in order to readily distinguish them from relation types.

```
FL:     Father .(Animal,Male)  <  Parent;  //with (already asserted):  Parent .(Animal,Animal);
KIF:    (supertype Father Parent)
        (relational_concept_signature Father Animal Male "generatedRelTypeFor_")
KIF:    (supertype Father Parent)
        (binary_relation_type generatedRelTypeFor_Father)
        (corresponding_relation_type Father generatedRelTypeFor_Father)
        (domain generatedRelTypeFor_Father Animal)  (range generatedRelTypeFor_Father Male)
        (=> (generatedRelTypeFor_Father ?a ?m) (Father ?m))  ;;?a and ?m are free variables
        (=> (Father ?m)
            (exists (?a ?m) (and (Animal ?a) (Male ?m) (generatedRelTypeFor_Father ?a ?m))))
        (subrelation_of generatedRelTypeFor_Father generatedRelTypeFor_parent)
        ;;since "Parent .(Animal,Animal);" is already asserted, generatedRelTypeFor_parent is
        ;;  assumed to exist
```

Below is a definition of `relational_concept_signature' aimed to permit the above two KIF translations to be equivalent. The definitions of the two subfunctions re-use the KIF functions `subrelation_of', `domain', `range', `name' (which returns an identifier of the given object) and `denotation' (which returns the object from the given identifier). The other terms are from `pm'. These definitions were inspired by one of the examples in Section 10.3 of Standford's KIF manual [www-KIF 1998]. These definitions may not be entirely correct in some versions of KIF, e.g., some versions of KIF do not accept variables in the place of predicates. They might also not actually achieve the intended effect; however, they give an idea of the intended approach.

```
(defrelation relational_concept_signature
     (?relationalConceptType ?sourceType ?destType ?generatedRelTypePrefix) :=
  (exists ((?relType binary_relation_type) (?supRT binary_relation_type))
    (and (supertype ?relationalConceptType thing_that_can_be_seen_as_a_relation)
         (corresponding_relation_type ?relationalConceptType ?relType ?generatedRelTypePrefix)
         (domain ?relType 1 ?sourceType)  (range ?relType ?destType)
         ;;to ease the comparison with the previous example, the free variables ?a and ?m
         ;;  are used below; however, more appropriate names could be ?source and ?dest
         (=> (?relType ?a ?m) (?relationalConceptType ?m))
         (=> (?relationalConceptType ?m)
             (exists (?a ?m) (and (?sourceType ?a) (?destType ?m) (?relType ?a ?m))))
         (=> (exist ((?supCT concept_type))
               (and (supertype ?relationalConceptType ?supCT)
                    (corresponding_relation_type ?supCT ?supRT)))
             (subrelation_of ?relType ?supRT))
    )))
```

```
(defrelation corresponding_relation_type
     (?relationalConceptType ?relType ?generatedRelTypePrefix) :=
  (and (concept_type ?relationalConceptType)
       (binary_relation_type ?relType)
       (=> (not (exists ((?rt binary_relation_type))
                  (manually_set_corresponding_relation_type ?relationalConceptType ?rt)))
           (and (=> (/= ?generatedRelTypePrefix "")
                    (= ?relType (denotation (concat ?generatedRelTypePrefix
                                                    (name ?relationalConceptType)))))
                (=> (and (= ?generatedRelTypePrefix "")
                         (uppercase (car (name ?relationalConceptType))))
                    (= ?rt (denotation (cons (lowercase (car (name ?relationalConceptType)))
                                             (cdr (name ?relationalConceptType))))))
                (=> (and (= ?generatedRelTypePrefix "")
                         (not (uppercase (car (name ?relationalConceptType)))))
                    (?relType (denotation (concat "generatedRelTypePrefix_"  ;;default prefix
                                                  (name ?relationalConceptType)))))
           ))
       (=> (exists ((?rt binary_relation_type))
             (manually_set_corresponding_relation_type ?relationalConceptType ?rt))
           (= ?relType ?rt)))) ;;to be safe, even if this may be redundant with the
                               ;;  fact that manually_set_corresponding_relation_type is
                               ;;  a subtype of corresponding_relation_type
```

Since the relation generatedRelTypeFor_Father is actually meant to be functional, here are the actually looked-for equivalences.

```
FL:     Father .(Animal -> Male)  <  Parent;  //with:  Parent .(Animal -> Animal);
                                              //and therefore:  Father < Male;
KIF:    (supertype Father Parent)
        (functional_concept_signature Father Animal Male "generatedRelTypeFor_")
KIF:    (supertype Father Parent)
        (supertype Father Male)  (unary_function_type generatedRelTypeFor_Father)
        (corresponding_relation_type Father generatedRelTypeFor_Father)
        (domain generatedRelTypeFor_Father Animal)  (range generatedRelTypeFor_Father Male)
        (=> (= ?m (generatedRelTypeFor_Father ?a)) (Father ?m))
        (=> (Father ?m) (exists (?a ?m)
                          (and (Animal ?a) (Male ?m) (= ?m (generatedRelTypeFor_Father ?a)))))
        (subrelation_of generatedRelTypeFor_Father parent)
```

Here is a definition of `functional_concept_signature' aimed to permit the above two KIF translations to be equivalent.

```
(defrelation functional_concept_signature
     (?functionalConceptType ?sourceType ?destType ?generatedRelTypePrefix) :=
  (exists ((?fctType unary_function_type) (?supRT binary_relation_type))
    (and (supertype ?functionalConceptType ?destType)
         (supertype ?functionalConceptType thing_that_can_be_seen_as_a_function)
         (corresponding_relation_type ?functionalConceptType ?fctType ?generatedRelTypePrefix)
         (domain ?fctType ?sourceType)  (range ?fctType ?destType)
         (=> (= ?m (?fctType ?a)) (?functionalConceptType ?m))
         (=> (?functionalConceptType ?m)
             (exists (?a ?m) (and (?sourceType ?a) (?destType ?m) (= ?m (?fctType ?a)))))
         (=> (exist ((?supCT concept_type))
               (and (supertype ?functionalConceptType ?supCT)
                    (corresponding_relation_type ?supCT ?supRT)))
             (subrelation_of ?fctType ?supRT))
    )))
```

4.3. Towards a Shared LR(1) Grammar For Parsing FL, FCG, FE, CGLF, CGIF and KIF

Versions of the grammars used by WebKB-2 for FL, FCG and FE are accessible at http://www.webkb.org/doc/languages/. This section presents a grammar that permits to i) parse FL, FCG, FE, CGLF, most of CGIF, and the basic syntax of KIF, and ii) use some of their components in various kinds of commands. (This grammar will soon be extended to parse all of CGIF and KIF). Such a grammar has three kinds of interests:

- it offers another viewpoint for comparing these notations;
- it permits to mix them and hence benefit from their respective advantages;
- it is a step towards the parsing of many (families of) notations and the dynamic modification of the notation by each user (Section 4.5 provides another step); having a single parser instead of several also eases the development of a KBMS;
- it is not restricted to declarative assertions: it can also be used for querying and procedural programming.

However, this shared grammar cannot anymore enforce the following of one of its sources: FE, FCG, FE, CGLF, CGIF or KIF. If necessary, this may be implemented by a simple check in the action associated to each rule of this shared grammar: if the rule does not belong to the notation selected by the user, an error message is issued. This grammar is meant to be usable with the Yacc (or Bison) parser generator, and hence is meant to be LALR(1). It might have to be refined for actually be LALR(1): its implementation will ascertain this point and, if necessary, permit to refine this grammar.

This grammar is meant to be completed by lexical parsing rules usable by a lexical parser such as Lex or Flex. (Every language parser in WebKB-2 is implemented with Bison and Flex; the above cited Web page also leads to versions of Flex grammars used by WebKB-2). Table 3.4.1 shows some lexical rules. Many other lexical rules are not shown, e.g.,

- those skipping white spaces: '\n', '\t', '\r', ' ', HTML tags, HTML comment delimiters, inline/multiline comments such as those in C++ ('//...', '/* ... */') and Pascal ('(* ...*)'), ...
- those parsing annotation strings ("(^ ...^)") and other kinds of strings,
- those expanding lexical variables (those beginning by '$') – including within double quoted strings or strings delimited by '$(' and ')$' – before their re-parsing (this is why only 'lexical_var_name' appears in the structure grammar, not "$"lexical_var_name), and
- those required to consider as a token each string in the structural grammar shown by the tables 3.4.2 to 3.4.7.

In the tables of this section, the Extended BNF notation is used. Reminder:

- '?' means 0 or 1 times, '*' means 0 to N times, '+' means 1 to N times,
- '|' introduces an alternative and has the lowest precedence.

Additional conventions are also used:

- the form 'Rule_name_1 = Rule_body = Rule_name_2' is sometimes used as an abbreviation for 'Rule_name_1 = Rule_body' and 'Rule_name_2 = Rule_body',
- spaces between tokens indicate that white spaces *must* be used between the tokens (this permits "terms" to be composed of nearly any character that is not a white space or a punctuation sign; this is needed since most words or names should directly be usable as terms without having to introduce escape characters,
- '0-9' is used to refer to any digit and '[...]' is used to refer to a set of alternative possible characters,
- tokens beginning by an uppercase initial have at least one sub-element that cannot be directly acquired by the lexical parser (i.e., their acquisition has to involve the structural parser), and
- C++ comments are used for making some comments.

Finally, in Table 4.3.1, the order of the rules is important, e.g., the description of the rules for 'number', 'user_ID' and 'term' partially overlap but the rule for 'number' has precedence over the rule for user_ID' which has precedence over the rule for 'term'.

Table 4.3.1. Some of the lexical rules

```
non-predefined_token = numerical_value | non-numerical_value

 numerical_value = number | date | time
  number         = ("+"|"-")?natural("."[0-9]*)?
   natural       = [0-9]+
  date           = [0-9][0-9]?"/"[0-9][0-9]?"/"[0-9][0-9][0-9][0-9]
  time           = [0-9][0-9]?":"[0-9][0-9](":"[0-9][0-9])?

 non-numerical_value = user_ID | term | variable | annotation //(^...^)
  user_ID        = [a-z0-9]term_letter  //if this user_ID is not registered, the lexical parser
                                        // must mark this token as 'term', not 'user_ID'
  term           = url | string | term_letter1(term_letter)*
   url           = [A-Za-z]+"://"[A-Za-z0-9_\-/.~#%$@?&+=]+ //more characters could be accepted
   term_letter1  = "#"?[a-z0-9_.]                           //more characters could be accepted
   term_letter   = [A-Za-z0-9_\-/.~#%$@?&|\'] | "\\"(.|\n)  //more characters could be accepted
  variable       = ("*"|"?"|"@"|"^")(term|number)  //not "$"term since expanded by another rule
```

Table 4.3.2. Commands

```
Commands     = Command  command_end?
             | Command (command_sep Commands)*
 command_end = ";" | "?" | "."      //normally, "." or "?" in FE, and ";" in FCG or FL
 command_sep = command_end | "|"    //"sep": separator
              //when a pipe ("|") is used after a command, its results become additional
              //  arguments of the next command (even from a parsing viewpoint);
              //  thus, for example, the next two commands are equivalent:
              //    print "def" "ghi" | print "abc";    print "abc" "def" "ghi";
 Command     = Assertion | Search | Comparison | Generation | Removal | Update
             | Print | Warn | Control

 Assertion   = graph
 Search      = query_operator   Node
  Node       = user_ID  |  graph //and hence also:  term, string, value, relation, ...
 Comparison  = query_operator   Node+
               "compare" term term+ "on" "(" Links ")" "," "maxdepth:" number  //Table 2.4.5.1
 Generation  = gener_operator   Node+
 Removal     = removal_operator Node
 Update      = update_operator  Node Node  //the second node replaces the first
 Print       = print_operator   Node+
 Warn        = warn_operator    user_ID  Node+
 Control     = file_load  |  Ctrl_structure  |  parsing_directive

  query_operator      = is_specialization | is_generalization | is_comparable
   is_specialization  = "spec" | "ext-spec" | "specGtypes" | "ext-specGtypes" | "?"
                      | "spec/noExcl"                 //"ext-specGtypes" and "?" are aliases
                      | "ext-spec/noExcl" | "specGtypes/noExcl" | "ext-specGtypes/noExcl"
   is_generalization  = "gen"  | "ext-gen" | "gen/noExcl" | "ext-gen/noExcl"
   is_comparable      = "comp" | "ext-comp" | "comp/noExcl"
                      | "ext-comp/noExcl" | "??" | "Is there"  //these last three are aliases
  generation_operator = "maxjoin"
  removal_operator    = "del"
  update_operator     = "mod"
  print_operator      = "print" | "echo"  //"print" (Term|Number)*
  warn_operator       = "email"           // (user_ID | LinkKind Term)
  file_load           =  file_load_operator URL
   file_load_operator = "load" | "run " | "display" | "include"
  Ctrl_structure      = lexical_var_name "=" (Node | Expression)
                      | "if" "(" Expression ")" Command_block ("else" Command_block)?
                      | "while" "(" Expression ")" Command_block
                      | "for" lexical_var_name "=" Expression "to" Expression  Command_block
   Command_block      = "{" Commands "}"
   Expression         = "(" Expression ")"  |  "-" Expression
                      | Expression binary_operator Expression
                      | term (":="|"=") Expression   | "set" term  Expression
                      | value  | lexical_var_name  | Node
    lexical_var_name  = term
    binary_operator   = "+" | "-" | "*" | "/" | "%" | "^" | "||" | "&&"
                      | "<" | ">" | "=<" | ">=" | "==" | "!=" | "=>" | "<="
  parsing_directive   = //  Section 4.5.5 shows how to build parsing/presentation directives;
                        //the next ones are certain abbreviations currently used in WebKB-2:
                        ("load"|"run"|"display")"mode"  |  "no"? "storage"
                      | ("use"|"no") "names"  |  ("unprefixed"|"prefixed") "variables"
                      | "default creators:" user_ID*  |  "no"? "trace"
```

In the next tables of this section, 'Relation' refers to a statement represented by a stand-alone and complete relation (i.e., a relation node with all the concept nodes it relates) while 'Link' refers to a relation less one of its concept nodes (the one that "has" this link, from the viewpoint of the parser). In FL, FCG an FE, a link is binary. In CGLF it may not be. A concept node (e.g., 'Top_concept' and 'Delimited_concept') includes its links. In the token names of this grammar, the substring "F_" refers to FL, FCG and FE while the substring "CG_"refers to CGLF and CGIF.

Table 4.3.3. Top-level components of a graph

```
Graph                = Top_concept //not always sufficiently delimited to be used in a link
                     | Relation    //stand-alone and complete relation node
 Top_concept         = Concept_graph_core  //to allow a non-delimited concept/graph
                     | Pre_context?  Concept_graph  Post_context?
  Concept_graph_core = F_pre_links?  Concept_core  F_post_links?
  Concept_graph      = negation  "[" "[" Concept_graph_core "]"  CGLF_post_links "]"
                     | negation  "[" Concept_graph_core "]"
                     |           "[" Concept_graph_core "]"  CGLF_post_links?
                     | negation? "`" Concept_graph_core "'"
                     | "[if" Graph  "then" Graph  ("else" Graph)? "]"
                     | Term_definition  //Table 4.3.5
                     | string  //to allow the use of an informal string as a statement
   negation          = "~" | "!" | "not"   //only "~" in CGLF

  Pre_context        = "[_"  Context_links "]"   //pre/post contexts are only for FL/FCG/FE
  Post_context       = "_["  Context_links "]"   //  see Paragraph 2.1.1.4
   Context_links     = Context_link  (link_separator Context_link)*
    link_separator   = "," with? | "and"? with          //only "," in CGLF
     with            = "with"|"that"|"has for"|"have for"|"for"|"is"|"are"
    Context_link     = user_ID | date /*creation_date*/ | Cardinality | Links
   Link_context_links = Link_on_link (link_separator Link_on_link)  //used only in next table
    Link_on_link     = Context_link | "1st" | "2nd" | "3rd" | (number|variable)"th" | "main"

Relation             = KIF+CGIF_relation  |  CGLF_relation  |  F_relation
 KIF+CGIF_relation   = "(" relation_type  (Top_concept | KIF+CGIF_relation) ")"
                     | "<" relation_type  (Top_concept | KIF+CGIF_relation) ">"
                       //this permits KIF-like relations and CGIF relations/actors, plus
                       //  the use of FL/FCG/FE/CGIF/CGLF concepts
  relation_type      = (predefined_rel_type | term) annotation*
                       //the above term may actually be a concept type (Paragraph 2.1.1.14)
 CGLF_relation       = "(" relation_type ")" CGLF_relation_args
  CGLF_relation_args = "<-" "[" Concept_core "]" "," "->" "[" Concept_core "]"
                     | "->" "[" Concept_core "]" "," "<-" "[" Concept_core "]"
                     | "-"natural"->" "[" Concept_core "]"
                        ("," "-"natural"->" "[" Concept_core "]")*
 F_relation          = "(:"  Relation_elem+ ")"   //see Paragraph 2.1.1.6
  Relation_element   = Delimited_rel_core  //only 1 such elem in this Relation
                     | Delimited_concept   //only 1 for a unary relation, 3 for a ternary ...
   Delimited_concept = Concept_min_core  //"minimal" -> still LR(1) even without delimitations
                     |  "(" "."?  Concept_graph_core ")"
                     | Pre_context?  Concept_graph  Post_context?
     //In a F_relation or a concept, "(" doesn't introduce a relation (unlike at the
     //  top-level or in a KIF+CGIF relation) but isolates concept node (like "[")
     //  without turning it into a statement (unlike "["). These two usages are rather
     //  easy to distinguish when looking at a statement, and there are unlikely to be mixed.
     //  Paragraph 2.1.1.20 illustrates the purpose of the above "(" ".".
```

Table 4.3.4. Links

```
CGLF_post_links          =  CG_link  |  "-" "{" CG_link ("," CG_link)+ "}"
 CG_link                 = "->" D_link_core ("->[" Concept_core "]")?
                         | "<-" D_link_core ("<-[" Concept_core "]")?
                         | "<-"natural"-" D_link_core ")-"natural"->[" Concept_core "]"
                         | "<-"natural"-" D_link_core "-" "{" "-"natural"->[" Concept_core "]"
                                                ("," "-"natural"->" "[" Concept_core "]")* "}"
  D_link_core            = "(" Link_core ")"  |  "<" Link_core ">"  //delimited Link_core

F_pre_links              = "(_" Links ")"     //see Paragraph 2.1.1.6
F_post_links             = "_(" Links ")"     //see Paragraph 2.1.1.6
                         | link1_begin_in_F  Links
 link1_begin_in_F        = link_separator     //see link_separator in previous table
    //This separator is needed for this grammar to be LALR(1) and hence be used with Yacc.
    //For F_link1_begin, FCG uses "," and FE uses 'with' (see previous table).
    //Since FE does not have a 'F_link1_begin', when a parser based on this grammar is
    //  used on FE statements, its lexical sub-parser (e.g., the Lex parser) must first
    //  send an 'F_link1_begin' to the structural sub-parser (e.g., the Yacc parser)
    //  whenever it recognizes the beginning of a relation node, e.g., when it encounters
    //  i) a predefined_rel_type, ii) a Pre_context after a Concept_core, or
    //  iii) a term followed by an Link_core_end.
    //  This is the kind of things that the current FE parser of WebKB-2 does.
    //  The current FL parser of WebKB-2 does not need to do that
    //  because its FL grammar does not use a Concept_core as complete as the one in FCG.
 Links                   = Link (link_separator Link)*
  Link                   = Link_with_1_dest
                         | Link_with_destinations //-> links of same type from the same source
   Link_with_1_dest      = Link_core  Destination_concept
   Link_with_destinations= Link_core  Destination_concepts
    Link_core            = "not"? predefined_rel_type
                         | "not"? Link_regular_expr?  Link_min_core  Link_core_end
     predefined_rel_type = "=>" | "<=>" | "<=" | "=" | "!=" | "/=" | "<" | "=<" | ">" | ">="
                         | ".<" | ".>" | "^" | ":" | "~" | "!" | "/" | "-" | "_" | "&"
                         | "is" ("a"|"an")  //an alias for "kind:"
     Link_core_end       = "of"? ("_[" Link_context_links "]")?  (":"|"<="|"=>"|":<="|":=>"|":=")
      //regular expressions may be used in queries and assertions (e.g., see Table 4.5.4.2)
     Link_regular_expr   = Link_regExpr_core+  ("|" Link_regExpr_core+)*
      Link_regExpr_core  = "(" Link_with_1_dest ")" count  |  "(" Link_regExpr ")" count
       count             = "?" | "*" | number?"+" | number

     Link_min_core       = Pre_context? link_modality? link_rank*
                            (relation_type ("?"|"+")? | variable) annotation*
                         | "?"  //only in queries, to display relation types instead of graphs
      link_modality      = ("may" | "may" | "able" "to") ("have" "for" | "be" "the"?)?
      link_rank          = "1st" | "2nd" | "3rd" | (number|variable)"th" | variable
                         | "(" variable "!=" ("last"|number|Set_of_numbers_or_variables) ")"
                         //by default, variables in links are existentially quantified

    Destination_concepts = Dest_concept/partition Post_linkDest_context?
                            (dest_c/p_separator Dest_concept/partition Post_linkDest_context?)*
     Dest_concept/partition   = Delimited_concept
                              | Pre_context? "{"  Delimited_concept+  "}" Post_context?
                              | Pre_context? "{(" Delimited_concept+ ")}" Post_context?
                              | ("-"  Concept_graph Post_context?)+  //-> no use of "{" and "}"
       Post_linkDest_context  = "__[" Link_context_links "]"   //see Paragraph 2.1.1.4
     dest_c/p_separator       = "and"?  //omitted in FL, "and" in FE, not used in FCG
```

Table 4.3.5. Term definitions, signatures and cardinalities

```
 //The next lines permit CGLF(-like) definitions and KIF-like definitions
 //Links and context links can be associated to definitions (this may or may not make sense)
Term_definition = ("type"|"relation")? term ".(" Signature? ")" (":="|":<=>"|":=>"|":<=") Graph
                | ("type"|"relation")  term "("  Signature? ")" (":="|":<=>"|":=>"|":<=") Graph
                | "function"?          term ".(" Signature? ")" ":->" term? variable ":=" Graph
                | "function"           term "("  Signature? ")" ":->" term? variable ":=" Graph
 Signature      = Sig_arg  ("," Sig_arg)*  ("," term "+" |  term? ("?"|"*"))?  ("->" Sig_arg)?
  Sig_arg       = ((term variable?)|"?"|Collection_in_signature)?   // */?/+ -> variable arity
                  ("."|"@")?  ("[" half_numerical_cardinality "]")?
  half_numerical_cardinality  = number(".."(number|"*"))?

Cardinality    = (half_cardin ("<-"|"->"|"<->"|"<="|"=>"|"<=>") half_cardin) Post_context?
  half_cardin  = number".."("*"|number) | "any" | "every" | "*" | "?" | "@"
               | "." ("cuml"|"coll"|"complete")?  //collection interpretation (Table 2.1.1.11)
```

Table 4.3.6. Concept core

```
Concept_core     = CG_concept_core  |  F_concept_core

 CG_concept_core = Type (":" CG_referent)?
  CG_referent    = term_or_var?  (Collection? quantifier?  | Concept_graph)?
   term_or_var   = term  |  variable ("!=" (term|variable)?  |  "~"variable

 F_concept_core  = Term_expr   (quantifier Restrictors)? (Collection | Concept_graph+)?
                 |              quantifier Restrictors   (Collection | Concept_graph+)?
                 | group_of    quantifier? Restrictors    Collection?
                 | group_of    quantifier? Restrictors?   Collection
                 | value_range       |  ("!" variable)?  (Collection | Concept_graph+)?

 //Concept_min_core refers to the parts of F_concept_core that can be used without embedding
 //  them within square brackets or parenthesis, while still having an LR(1) grammar
Concept_min_core = Term_expr (quantifier Restrictor)?  |  quantifier Restrictor
                 | value_range  |  "!" variable?

 Term_expr       = Term_fct_expr  |  term ".(" ( Signature | Links /*Process_signature*/ ) ")"
  Term_fct_expr  = numerical_value  |  variable  |  "(" "-" Term_fct_expr ")"
                 | "(" Term_fct_expr ("+"|"-"|"*"|"/"|"mod")  Term_fct_expr ")"
                 | term                          //non-quantified term, e.g. wn#London
                 | term "_(" Params? ")"         //fct call or relation; n-ary arguments
                 |      "(_" Params? ")" term  //idem but with arguments before the term
   Params        = Delimited_concept ("."? Delimited_concept)+

 Restrictor      = (rank? | rank "occurrence of") qualifier? ("?"|variable|(Type  variable?))
 Restrictors     = (rank? | rank "occurrence of") qualifier? ("?"|variable|(Types variable?))
         //above "?": only in queries, to display concept categories/nodes instead of graphs
  rank           = "1st" | "2nd" | "3rd" | number"th"
  qualifier      = "good" | "bad" | "important" | "small" | "big" | "great"
  Types          = Type ("&"? Type)+  //in CGLF and CGIF, "&" is used for a type conjunction
   Type          = "(lambda" "(" Signature ")" Graph ")" //for CGLF and CGIF only
                 | ("!"|"~")? term annotation*  |  "(" "."? Type (Links | "|" Type)? ")"
                    //above line: type negation, lambda-abstraction and type disjunction;
                    //  Subsection 4.2.1 illustrates the purpose of the above "(."
```

Table 4.3.7. Collections, quantifiers and value ranges

```
Set_of_numbers_or_variables = "{" (number|variable) ("," (number|variable|"last"))* "}"

Collection_in_signature = "{" Signature_elements "}"          //optional arguments at the end
 Signature_elements     = signature_elem  ("," signature_elem)*  ("," signature_elem "?"?)?
  signature_elem        = (term variable?  |  variable)

group_of            = "together"  |  exist_quantif ("group of" | "bag of" | "set of" |
                                                    "sequence of" | "alternatives")

Collection          = F_pre_links?  Collection_core  F_post_links?
 Collection_core    = ("BAG"|"SET"|"LIST"|"SEQ"|"XOR"|"OR"|"AND")? (Open_coll | Closed_coll)
  Open_coll         = "{"  Elements?  "}" coll_size?
  Closed_coll       = "{(" Elements  ")}" coll_size?
   Elements         = Element ((","?|"or"|"and")  Element)*
    Element         = Delimited_concept
                    | term ("?"|"+"|"*")  //in signatures -> optional arguments
                    | "?" | "*"  //one vs. 'any number' of things (pm#thing / given Restrictors)
    coll_size       = "@" ("<"|">"|"=<"|">=")? number

quantifier          = existential_quantif | universal_quantif | numerical_quantif "of"? "the"?
 existential_quantif= "some"  |  "the"  |  "there is"? ("a"|"an")
                    | "@certain"  //for CGLF and CGIF
 universal_quantif  = "any"  |  "every" |  "@forall"
 numerical_quantif  = "about"?  number "%"?   |  "at" ("least"|"most") number "%"?
                    | "between" number "%"? "and" number "%"?   |   "no" //same as "0"
                    | "from"?   number "%"? "to"  number "%"?   |   number".."("*"|number)
                    | "most" "of"? "the"?  |  "mostly"  |  "several"  |  "a"? "few"
                    | "dozens"|"hundreds"|"thousands"|"millions"|"billions"|"trillions"
                    | "@"number(".."("*"|number))?"%"?  //for CGLF and CGIF
                    | "@"("<"|">"|"=<"|">=")number"%"?  //for CGLF and CGIF

value_range         = "about" (numerical_value|variable)
                    | "at" ("least"|"most") (numerical_value|variable)
                    | "between" (numerical_value|variable) "and" (numerical_value|variable)
                    | "from"? (numerical_value|variable) "to" (numerical_value|variable)
```

4.4. Summary of the Future Data Model of WebKB-2

There are many ways of storing and indexing knowledge representations or, more generally, graphs of concept nodes and relation nodes. The choice of data structures depends on the expressiveness of the stored knowledge representations and on the expected kind of queries. However, at a first general level, the approaches are either concept node centric, relation node centric, or a mix of both.

- In the *concept node centric approach*, each *main object* stores one concept node and has for sub-objects the relation nodes from/to this concept node. The main objects are indexed according to their concept types or identifiers but the relation nodes are not indexed. Among the KBMSs that follow this approach and use a classic DBMS (relational DBMS, OODBMS or deductive DBMS; "classic" in the sense of "exploiting a data model with few tables and that cannot be dynamically modified by the end-users"), typically for ensuring persistency and manage large amounts of knowledge, two kinds may be distinguished: i) the "KBMSs related to an application" that use *one table* (or "main/indexed class" in an OODBMS) *for each predefined type of concept* (then, the relations are represented as predefined attributes of this table), e.g., as in OntoDB [Dehainsala et al., 2007], and ii) the more generic KBMSs that use *one or two tables* for representing *all the concept terms and concept nodes that the end-users provide*.
- In the *relation centric approach*, the main objects are i) types/individuals, and ii) objects storing relation nodes and including concept nodes/terms as sub-objects. These objects storing relation nodes are indexed according to relation types and possibly also according to the types/individuals they relate. The concept nodes within them are not indexed. Thus, for information retrieval to be efficient, the relation types should not be basic and domain-independent. This unfortunately goes against the knowledge normalization rules and best practices listed in Section 2.3. Among the KBMSs that follow this approach and use a classic DBMS, two kinds may be distinguished: i) the "KBMSs related to an application" that use *one table for each predefined type of relation*, and ii) the more generic KBMSs that use *one table for representing all the terms that the end-users provide* (with, for each term, attributes for storing its direct supertypes and subtypes) and *one or two tables for storing all the relation nodes that the end-users provide*. Most general KBMSs using a DBMS follow this last approach, e.g., Powerloom [www-PowerLoom 2009], RDF-based DBMSs, and Formal Concept Analysis based KBMSs such as ToscanaJ [www-ToscanaJ 2009]. They use relational DBMSs and many of them translate knowledge queries into SQL queries (e.g., as described in [Groh & Eklund, 1999]); given the limitations of RDBMSs and SQL, this approach is only possible or efficient when the data model – especially the concept node structure – is simple, e.g., without explicit quantifiers. Most RDF-focused KBMSs (often called "RDF database systems") do not take OWL into account and many of them do not even take supertype relations into account.
- In the *mixed approach*, the concept nodes and relation nodes are stored in different kinds of objects that refer to each other.

```
pm#graph_ADT_storing_knowledge-representations  //declared in Table 2.1.3.1
  >  {concept_node_centric_graph_ADT_storing_knowledge-representations
      relation_node_centric_graph_ADT_storing_knowledge-representations};
```

WebKB-2 is written in C++ on top of the free-to-use object relational DBMS FastDB/Gigabase [www-Knizhnik-tools 2009]. FastDB loads/maps the whole DB in main memory before managing it and hence is extremely efficient despite this initial loading/mapping of about one or two seconds. In case the DB grows larger than 4Gb (on a 32 bit system), FastDB can be replaced with its disk-based version called GigaBASE which, like other OODMSs, is at least 10 times slower than FastDB.

The data model of WebKB-2 uses a concept node centric approach. It currently has four tables (or "main/indexed classes"; the name prefix "Db" highlights this): DbUser, DbCategory, DbCategName and DbNode. Subsection 2.4.2 introduced the "structural change" that will be made in WebKB-2 to simplify its structures, ease programming, speed up the comparison of graphs and ease their display in FL without making more difficult their display in other formats. Thus, in the future, the tables will be: DbUser and DbTerm. These tables and the sub-structures they re-use are presented below using the core of their C++ declarations. They minimize the number of operations or database

accesses required for WebKB-2 to perform its core operations: graph-matching of expressive knowledge, the display of knowledge in various formats, and knowledge-based collaboration management. Here are some additional more precise points.

- DbNode disappears due to the above cited structural change.
- DbCategory and DbCategName are merged into DbTerm in order to i) organize informal terms the same ways formal terms are organized, and ii) re-use the same management procedures (comparison, display, navigation, ...). This management is extended to handle the `extended_specialization' relations, not simply the (formal) specialization relations. Several kinds of objects stored as string fields of DbCategory are now stored as instances of DbTerm and hence can more efficiently be retrieved.
- FastDB/Gigabase more efficiently manages a fewer number of tables. However, DbUser is still distinguished from DbTerm for code readability purposes and to permit static type checking.
- FastDB/Gigabase does not permit to declare inheritance links between classes specifying database tables. Hence, fields are used for storing term "kinds", e.g., the boolean 'isFormal' to distinguish formal terms from informal terms, and the term reference 'kind' to distinguish between an individual, a concept type, a relation type (a functional one, a transitive one, etc.) and a type of higher order. For code readability purposes, the formal/informal distinction is still made; to that end, type aliases are used; however, this does not permit static type checking. There is no such downside for the term reference 'kind' and other fields 'kind' used in this data model since, most often, no more specialized classes could have been used even if inheritance had been possible. Furthermore, C++ does not offer any easy way to test the belonging of an object to a class. On the other hand, a field can easily be tested and can be hash-coded.
- In this model, relations from/to a term are called "links". Since during parsing the kinds of links that a particular term will have are unknown, to save memory/database space all links could be stored in a same array sorted according to the link kinds. Alternatively, to make the access to links of particular kinds more efficient, special fields may be used for storing these links. The code should be independent from such a choice. For illustrating this and for readability purposes, in the data model presented below, many fields are used for storing special kinds of links; two fields named otherOutLinks and otherInLinks are used for storing the other links. Separating outLinks (links from the term to other terms) to inLinks (links from other terms) speed up graph comparison.

```
//preliminary declarations:
class DbUser;  class DbTerm;  //the two tables (main/indexed classes)
typedef dbReference<DbUser> dbrUser;  //dbrUser is a database reference to a user
typedef dbReference<DbTerm> dbrTerm;
class Link;  //a sub-structure of DbTerm
class NodeQuantificationAndInterpretation;  //a sub-structure of Link
//dbArray, dbDateTime and real8 are types declared by FastDB/Gigabase



class DbUser
{ const char *name; //login name in WebKB-2, e.g. "pm", "sowa", "cyc", "suo"
  const char *password;  //always stored encrypted
  dbrTerm uri;  //term with keyName an email/URL address
  dbArray<dbrTerm> createdTerms;  //link 1-N; inverse relationship: creator (see class Term)
};
```

```
class DbTerm   //formal/informal named/anonymous (and quantified or not) term
{ dbrUser creator;  //inverse relationship: createdTerms (see class User); indexed by the DBMS
  dbDateTime creationDate;  //most often automatically added by WebKB
  dbArray<dbrTerm> sourceModules;  //interpreted by the "creator" if not created by him/her/it
  bool isFormal;
  dbArray<const char *> keyNames;  //to speed knowledge retrieval/comparison, each of the
    //  keyNames must be hashed or indexed; FastDB/Gigabase permits the code of WebKB-2 to
    //  specify this via the C++ macro "TYPE_DESCRIPTOR(( KEY(keyNames,HASHED|INDEXED) ))"
    //if isFormal=true, this field stores a name that is unique when prefixed/postfixed by the
    //  namespace of the creator;
    //if isFormal=false, this field stores the name of this informal term;
    //in WebKB-2, this name begins by "*" for a variable (e.g., a coreference) and
    //  by "?" for a generated term (e.g., for a term referring to a collection or
    //  lambda-abstraction; such generated terms are not displayed when query results are
    //  displayed, e.g., the referred collection or lambda-abstraction is displayed)
  dbArray<dbrTerm> instances;
  dbArray<dbrTerm> types;  //most specific types (generated term if lambda abstraction)
  dbrTerm kind;  //general type from a predefined list such as the following one
    //  (to speed comparisons, the references to such types need to be hash-coded):
    //  individual (with special hash-codes for common datatypes such as string, number,
    //  date, duration, ... and also for collections such as set, bag, list, OR-set, ...),
    //  concept type, (functional, transitive, symmetric, ..., or injective) relation type,
    //  type of higher order, ..., yet undefined category (reference: null)
    //from a collection, links are used for specifying its members and size (conversely,
    //  from a term in a collection, links are used for specifying this collection and may
    //  be used for specifying its rank in the collection
  dbArray<Link> subtypes;  //the direct subtypes and, first in the array, the terms (with
    //  generated identifiers) representing closed subtype partitions (followed by open
    //  subtype partition unless only those involving direct subtypes are displayed; indeed,
    //  such partitions can quickly be re-generated based exclusions between these subtypes)
  dbArray<Link> supertypes;  //inverse relationship: subtypes
  dbArray<Link> otherExtendedSpecializations; //inverse relationship: informalGeneralizations
  dbArray<Link> otherExtendedGeneralizations; //e.g., category names if this term is formal
  dbArray<Link> exclusions;  //the type of each link refers to the exact type of exclusion
  dbArray<Link> equivalentTerms;  //identity links; to speed graph comparison and avoid
    //  duplications, all the other links may be moved to one of the equivalent terms
  dbArray<Link> exclusions,  equivalentTerms;
  dbArray<Link> parameters_or_definitions;  //relation signature, non-binary relation/function
    // call, or definition of this term by necessary and/or sufficient conditions
  dbArray<Link> otherOutLinks;  //other links to other terms
    //sorting in this array: first otherTransitiveOutLinks (e.g., member), then
    //  otherNonContextualizingOutLinks, outLinksToOuterContext (links to terms in
    //  meta-statements on this term) and, finally, outLinksToInnerContext
  dbArray<Link> otherInLinks;  //inverse relationship: otherOutLinks
  bool isNegated;
};
```

```
class Link   //direct relation to a term or collection of terms
{ dbrUser creator;  dbDateTime creationDate;  dbArray<dbrTerm> sourceModules;
  dbArray<dbrTerm> types;  //most specific types (one generated term in case a
    //  lambda abstraction is used for the type of this link)
  dbrTerm kind; //general type from a predefined list such as the following one
    //  (to speed comparisons, the references to such types need to be hash-coded):
    //  relation signature parameter, relation/function call,
    //  definition of NSC, definition of NC, definition of SC, can have, may have, ...
  dbArray<Link> outLinks;  //links to outer context
  dbArray<Link> inLinks;   //inverse relationship: outLinks
  bool isNegated;
  dbrTerm destTerm;  //null if the destination is a number/date/period/...;
    //  the destTerm may be a collection (with a generated or manually given identifier)
  NodeQuantificationAndInterpretation sourceNodeQI, destNodeQI;
};

class NodeQuantificationAndInterpretation
{ dbrTerm kind;  //null if no quantifier; hash-code for an item in a predefined list such as:
    //  existential quantifier, universal quantifier,
    //  numerical quantifier (e.g., with a percentage, w.r.t. a collection, ...), ...
  dbrTerm interval;  //null if no interval (exact number/date/duration/...); hash-code for
    //  about/near, around/(very) approximatively, at least, at most, between/from-to, ...
  real8 value;  //e.g., 60 for the representation of "60% of wn#cat" or "60 to 75 wn#cat";
    //  hash-coded for qualitative values, e.g., 60 means "most" if the above "kind" refers
    //  to "numerically quantified with qualitative range" (similarly, with other values,
    //  60 can mean "mostly" and -60 can mean "millions")
  real8 value2;  //e.g., 75 for the representation of "65% to 75% of wn#cat"
  dbrTerm qualifier;  //null if no qualifier; hash-code for an item in a predefined list such as:
    //  certain, bad, big, good, great, important, small, ...
  dbrTerm collectionInterpretation;  //null if not a collection; hash-code for
    //  (at least partially) distributive, fully distributive, collective, cumulative,
    //  complete (w.r.t. this link; default: incomplete)
};
```

4.5. A General Ontology for Notations and Knowledge Presentation

The general introduction has presented the interest of building of a core ontology for components of models and notations of KRLs: supporting the building of

- a generic parser for one or several families of notations, depending on the values/instances of certain categories in this language ontology (e.g., pm#relation_node_end_delimiter), and
- a generic KRL that can be personalized by people to suit their own preferences or to make this KRL look like an existing notation for import/export purposes.

Then, Paragraph 2.1.1.6 and Paragraph 2.1.1.19 gave more precisions. Subsection 4.5.2 completes the technical explanation and shows the top-level of the presentation ontology. The subsections 4.5.3 and 4.5.4 refine this top-level and Subsection 4.5.5 shows how arbitrary complex parsing/presentation directives can be built by exploiting this ontology.

Common Logic, the current KR interlingua standard [ISO/IEC 24707, 2007] gives a common meta-model or abstract syntax for first-order logic KRLs (Section 6.1 of this standard), describes its semantics, and gives three standard notations that follow this model: CLIF (a predicate-logic based notation that directly matches this model), XCL (an XML based notation that also directly matches it) and CGIF (a graph-based notation that indirectly matches it).

Table 4.5.1.1 shows a representation in FL of this meta-model (in the standard, it is described informally and a UML representation is then given for describing certain relationships between its elements; only the informal description has been translated in Table 4.5.1.1., not the UML representation which is over-specified on some points and incomplete on other points). This meta-model is clearly (and purposely) "minimal" and hence, for the purposes of this section, needs to be extended. It is however taken as a starting point since it is a standard. The other tables of the next subsection extend this meta-model using a graph-oriented viewpoint. This extension is similar to Conceptual Graphs related meta-models [Sowa, 1984, 2000] [Gerbé et. al, 2001, 2007] but is represented using a KRL (more precisely, FL), is a bit more detailed or expressive, and includes distinctions related to "commands" (i.e., to assertions, knowledge queries and control commands) rather than to assertions only. This extension is used in the (re-)presentation ontology described by the remaining subsections.

The CL meta-model and this extension constitute the "meta-model of FS" (and hence of FCG, FE, FL and all other KRLs that WebKB parses or will parse). A good part of this meta-model of FS was re-used in the MOF2 meta-model [www-MOF2 2006] named "KRM" (for "Knowledge Representation Meta-model") [Martin, 2003d]. The MOF HUTN/textual format of KRM was named KRF. The Meta Object Facility (MOF) is a standard of the Object Management Group (OMG) that has been used for modeling most technologies standardized by the OMG, including UML and various UML profiles. KRM was at the core of the DSTC proposal to the OMG [Raymond, Martin & Colomb, 2003] in answer to its Ontology Definition Meta-model RFP [www-ODM-RFP 2003]. This RFP asked for propositions of extensions to UML 2.0 for allowing UML users to create/import/export ontologies (especially OWL ontologies) and hence represent and share various kinds of knowledge. The four proposals received by the OMG have been merged into the final specification of the Ontology Definition Meta-model [Colomb et al., 2005].

Each year, at the ICCS conference, a "CG tool workshop" occurs. One of its main goals is to compare current CG tools [www-CG-tools 2009]. Subsection 2.4.5 introduces a way to support, formalize and explore such a comparison. The other main goal of this workshop is to find solutions for CG tools to be more inter-operable, for example via

- *the use and extension of CGIF*; Chapter 4 lists certain extensions that are needed for knowledge sharing purposes and shows how they can be made;
- *the use of a common way to call (and hence compose) services proposed by various CG tools*; [Sowa, 2004] advocated the use of a common framework such as the Flexible Modular Framework (FMF); a more lightweight and complementary approach is to use a HTTP GET/POST based command call RESTful approach and a language of commands as propose in WebKB and introduced in the paragraphs 2.4.1.3 to 2.4.1.5;
- the design and use of a common API; on the CG mailing list the author of this document advocated the re-use of the API of Open Knowledge Base Connectivity [www-OKBC, 1998] since it would also permit network-based calls to other KBMSs or knowledge servers, e.g., Loom or Ontolingua, and enable the KB to be graphically browsed and edited by the Generic Knowledge Base Editor [GKBE, 1998];
- the design of a common meta-model; from a certain viewpoint, this quest ended when CGIF became a standard notation for the CL model; however, many CG researchers prefer adopting and extending to a CG-based meta-model for describing their own tools or notations; being an ontology that extends the CL model, is CG-based and that can be collaboratively extended via WebKB-2, the "meta-model of FS" can be used as a starting point for each CG researcher to represent elements that its tools or notations require and to relate them to other elements.

The Peirce project [Ellis et al., 1992] aimed to build a "freely available, state-of-the-art, industrial strength CG workbench and involved over 40 researchers from over 8 countries". The project was not fully successful, including with respect to the "freely available" part. The Griwes project [Baget et al., 2008] involves three French research teams and aims to "build a generic graph-based knowledge representation platform to support efficient and generic ways to store, query and present knowledge representations of all kinds, and for proposing commands or a programming interface to perform such operations". Griwes began by building a CG language/model that extends the one used by the CG workbenches created by these French teams: Corese [Corby et al., 2004] and CoGITaNT [Genest & Salvat, 1998]. The re-use of features from WebKB-2 – its collaboration protocols, its ontologies (including a very particular one: the "meta-model of FS"), its API abstracting over its data model, and its languages/notations (including its command-based interface) – would make Griwes more generic.

4.5.1. Ontology (or Meta-model) of FS and hence of Most Kinds of Knowledge Representations

Table 4.5.1.1. Representation in FL of the meta-model of Common Logic

```
[_ parsing][pm#new_term  pm#default_creator: cl]; //new unprefixed terms are from Common Logic
                                          //the default creator for existing terms is still pm

Common_Logic_element
  > (Common_Logic_model  direct_part=> 0..* phrase)  //top of the CL model partOf hierarchy
    phrase  term  sequence_marker  connective  comment;

    phrase
      > {( (module  direct_part:
                { 1    (. module_identifier_for_the_local_universe_of_discourse
                             < identifier)  //"(.": Paragraph 2.1.1.5 and Subsection 4.2.1
                  0..1 (. exclusion_set___names_excluded_from_local_universe_of_discourse
                             direct_part: 1..* name)
                  1    (. body_text  direct_part: 1..* phrase)
                }__[any -> ? complete] )
           (sentence
             > {( (quantified_sentence  direct_part:
                      { 0..* (. quantifier
                                   > {universal_quantifier  existential_quantifier} )
                        0..1 (. binding_sequence direct_part: 1..* name_and_sequence_marker)
                        1    (. body_of_quantified_sentence  < sentence)
                      }__[any -> ? complete] )
                  (boolean_sentence  direct_part:
                      { 1    (. connective  > {conjunction  disjunction  implication
                                              biconditional  negation} )
                        1..* (. component_of_boolean_sentence  < sentence)
                      }__[any -> ? complete] )
                  (atom > (equation  direct_part: 2 term)
                          (atomic_sentence  direct_part: 2 (. predicate < term))
                          argument_sequence)
                  (sentence_with_attached_comment  direct_part: 1 comment)
                  irregular_sentence
                )} )
           (importation  direct_part: 1 (. identifier_of_the_imported_piece_of_Common_Logic
                                           < identifier) )
           (text_with_attached_comment  direct_part: 1 comment___annotation)
       )};

    term
      > {( (name  > identifier,  exclusion: (sequence_marquer  = pm#coreference_variable) )
           (functional_term
              direct_part: { 1 (. operator < term)
                             1 (. argument_sequence
                                  <  (term_sequence
                                        direct_part: 1..* (. term_or_sequence_marquer
                                                             > {(term  sequence_marquer)} ) ) )
                           }__[any -> ? complete]
           (term_with_attached_comment  direct_part: 1 comment)
       )};
```

The following tables give a graph-oriented viewpoint and extension of the above meta-model.

Table 4.5.1.2. The top-level of the meta-model of FS (and hence of WebKB)

```
[_ parsing][pm#new_term  pm#default_creator: pm];  //new unprefixed terms are from pm again

FS_model_element   //FS_model is the top of the FS_model_element partOf hierarchy
  > (FS_model  = pm#AND-set_of_statements_in_FS,  //e.g., in FCG, FE, FL, RDF+OWL, CLIF, ...
               member=> 0..* (. command
                                 > cl#phrase  {(assertion  command_that_is_not_an_assertion)})
               >  Common_Logic_model /*because of the above line */ )
    Common_Logic_element
    command   query_operator   (term_or_KR_node >  {(term  KR_node)})
    type_reference_or_lambda_abstraction  concept_referent  set_of_link_nodes
    (concept_referent_element  direct_part of: concept_referent __[any->?,?<-any],
       > concept_node_literal  concept_node_designator  concept_node_quantifier
         concept_node_qualifier  concept_node_collection  embedded_graph
    );

    assertion  //set as supertype of sentence to be safe but there is no real difference
      > (cl#sentence  direct_part: 0..* KR_node);
        //FS uses a graph-oriented decomposition of a sentence while CL uses a
        //  predicate-logic oriented decomposition of a sentence

    command_that_is_not_an_assertion
      > {control_command
         (query  direct_part: {(   1 query_operator
                                0..1 (. set_of_terms_or_KR_nodes_as_parameters
                                          member=>  1..* term_or_KR_node) )} )  };

    KR_node  //something that is a statement; in this model, any KR_node except a KR_graph
             // can have links; in CGs, only a concept_node can have links/relations
      > {( (concept_node   /* = concept_node_core + in/out link nodes */
             > (first_concept_node  1st member of=> a set_of_concept_nodes)) //see below
           concept_node_core  //"concept node" in CG terminology; no context_link on this
           (relation_node  /* = relation_node_core + set_of_connected_KR_nodes */
             > (first_relation_node  1st member of=> a set_of_relation_nodes)) //see below
           (KR_graph
             direct_part: {(   1 (. set_of_concept_nodes  member=> 1..* concept_node)
                            0..1 (. set_of_relation_nodes member=> 1..* relation_node) )},
             > {( belief
                  (definition
                     > {non-anonymous_definition  lambda_abstraction}
                       {definition_of_NSC_condition  definition_of_NS_condition_only
                        definition_of_NC_condition_only} )  )} )
        )},
      direct_part: 0..* (. type_reference_or_lambda_abstraction
                           > {(concept_type/lambda  rel_type/lambda)} ),
      source_or_creator: a (. user_ID  < term_or_KR_node);  //source_or_creator is a shortcut
          // for the source/creator links in the context links that each KR_node may have;
          // this shortcut is not represented formally here; however, links are shortcuts too
          // and are represented formally below
      //annotations are represented via relations
```

Section 4.3 distinguished a Link from a (whole) Relation. Similarly, here, a link_node – the abstract structure that represents a Link – is a relation_node less one concept_node (the one that is focused on at some stage of the parsing or exporting of a graph; in this model, this may not be only a "concept node": see KR_node below). The definition of

set_of_link_nodes permits to derive links from relations (or conversely). A relation_node has for part a set_of_connected_KR_nodes and a concept_node has for part a set_of_links (if a set_of_relations was used, there would be a cycle of part relations). Contexts (on a concept_graph or a relation_node_core) are also represented via a set_of_links. However, the relationship with embedded graphs is not explicitly defined here. Since this model has both links and relations, it supports (re-)presentations via concept node centric approaches as well as relation node centric approaches.

Table 4.5.1.3. Concept nodes and links

```
concept_node  direct_part:   //pm#direct_part is defined in Table 2.1.1.17
  {(   1 concept_node_core
    0..1 (. set_of_concept_links  < set_of_link_nodes)
    0..1 (. set_of_context_links_on_a_concept_graph  < set_of_link_nodes,
           member=> 1..* (. context_link_node  < link_node) __[1<=any] )
         ) //creators, creation dates and negations are represented via contexts
  )} __[any ?c -> ?];

    set_of_link_nodes
      //direct_part of=> a concept ?c, //these first 5 lines seem redundant with those after
      //member: //all link nodes derivable from relations connecting ?c to other nodes
      //        (link_node  direct_part: (a link_node_core  direct_part of:
      //           (a relation_node  direct_part: (a set_of_connected_KR_nodes
      //                                             member: ?c))))__[?sl=>1..*, ?sl<=any]
      member=>
        1..* (. link_node  //either an out-link_node or an in-link_node:
               > {( (out-link_node                 //a concept_node is a KR_node
                     member of: (a set_of_link_nodes  direct_part of: a KR_node ?ko),
                     direct_part: (a (. link_node_core  < relation_node_core)
                                     direct_part of: (a relation_node ?ro
                                           direct_part: (a set_of_connected_KR_nodes ?sko
                                                           1st member: ?ko))), //-> out-link
                     direct_part: (a (. set_of_out-link_other_KR_nodes
                                       < (set_of_link_other_KR_node  < FS_model_element))
                                     member: (KR_node  != ?ko,  member of: ?sko)
                                             )__[?ol<->any] ) ),
                    (in-link_node
                     member of: (a set_of_link_nodes direct_part of: a KR_node ?ki),
                     direct_part: (a link_node_core
                                     direct_part of: (a relation_node ?ri
                                           direct_part: (a set_of_connected_KR_nodes ?ski
                                                           ( ^Nth != 1 ) member: ?ki))),
                     direct_part: (a (. set_of_in-link_other_KR_nodes
                                       < set_of_link_other_KR_node)
                                     member: (KR_node != ?ki,  member of: ?ski)
                                             )__[?il<->any] ) )
                )} ) __[1..*<=any],
      subset=>  //this complementary decomposition is also needed for FL:
        1..* (. set_of_link_nodes_with_same_type_and_direction
                 > {( (set_of_out-link_nodes_with_same_type  member=> out-link_node)
                      (set_of_in-link_nodes_with_same_type  member=> in-link_node) )},
                 member: (a link_node  direct_part: (a link_node_core  direct_part:
                             (a set_of_relation_types  member: a rel_type/lambda ?r)),
                 member: (link_node  direct_part: (a link_node_core  direct_part:
                           (a set_of_relation_types  member: ?r))) __[?ln<->any]
             ); //when no same relation type, each subset has only one element
```

Table 4.5.1.4. Concept node cores

```
concept_node_core  direct_part:
  {0..1 (. set_of_concept_type_references_or_lambda_abstractions
            member=> 1..* (. concept_type/lambda  >  type  lambda_abstraction
                          ) //concept/relation/2nd-order type
        ) //no type -> pm#thing
   1 (. concept_referent  direct_part:
            0..1 (. concept_node_literal  > number  string)
            0..* (. concept_node_designator
                      > category_name_or_identifier  coreference_variable)
            0..1 (. concept_node_quantifier
                      > cl#quantifier  numerical_quantifier
                        (quantifier_with_respect_to_a_collection
                           > numerical_quantifier_with_respect_to_a_collection) )
            0..1 concept_node_qualifier
            0..1 (. concept_node_collection
                      direct_part: 1..* concept_node
                                   0..1 (. collection_kind  < concept_type)
                                   0..1 collection_size
                                   0..1 collection_interpretation )
            1..* (. embedded_graph  < graph_node) //these disjoint graphs are
     )                     // generally seen as components of one global graph
  }__[any -> ? complete];

  numerical_quantifier
    > {from-to_quantifier  quantifier_with_one_value}
      {(quantifier_with_percentage  quantifier_without_percentage)}
      {(global_quantifier  numerical_quantifier_with_respect_to_a_collection)};
```

Table 4.5.1.5. Relation nodes

```
relation_node
  > {( (non-binary_relation_node
          direct_part: 0 source_KR_node  0 destination_KR_node)
       (binary_relation_node
          direct_part: 1 source_KR_node  1 destination_KR_node)
    )}
    {( (static_relation_node
          >  (contextualizing_relation_node  > negation  creator  creation_date))
       (dynamic_relation_node
          definition: "relation that represents a process, that can be triggered,
                       that takes concept nodes as input/output parameters, and that
                       computes value or asserts/retracts concept nodes"
          > (KR_actor_node  definition: "relation node that computes a value")
            (KR_demon definition: "relation node that asserts/retracts concept nodes"))
    )},
  direct_part:
        {(1 relation_node_core
          1 (. set_of_connected_KR_nodes  //connected to some relation node
                member=>  1..* (. connected_KR_node  < KR_node) __[1..*<=any],
                1st member=> 1 (. source_KR_node  = first_KR_node)__[1..*<=any],
                last member=> 0..1(. destination_KR_node  = last_KR_node)__[1..*<=any],
                (?Nth != 1st) member=> 0..* non-first_connected_KR_node __[1..*<=any]
            )
            member: 1..* (connected_KR_node  //-> connected to this relation node ?r
                          direct_part: (a set_of_link_nodes  member: ?r))
          0..1 (. set_of_context_links_on_a_relation  < set_of_link_nodes,
                    member=> 1..* (. context_link_node  < link_node) __[1<=any] )
        )} __[any ?r -> ?];


  relation_node_core
    direct_part:
          {( 0..1 (. set_of_relation_types
                    member=> 1..* (. rel_type/lambda
                                  > type  lambda_abstraction
                                    (rfol_for_modality
                                      > rfol_for_modal_possibility rfol_for_physical_ability
                                    ) )
                  )  //in  FL, FE and FCG, certain concept types may be used as relation types
                     //no type -> pm#relation
             0..1 (. relation_variable  < coreference_variable)
             0..1 (. assertion/definition_mark
                      > {mark_of_simple_assertion  mark_of_assertion_of_sufficient_condition
                         mark_of_mandatory_destination  mark_of_full_definition
                         mark_of_definition_of_sufficient_condition
                         mark_of_definition_of_necessary_condition} )
          )};
```

4.5.2. Parsing, Presenting and their Parameters

The goal of this subsection and the following ones has been presented in the general introduction and refined in Paragraph 2.1.1.6 and Paragraph 2.1.1.19. In the tables of these subsections, bold characters are either used for highlighting important variables and comments or for highlighting the identifiers of categories specialized in a subsequent table. These tables present (the top-level of) a ontology about how FS model elements can be (re-)presented. Usually, the presentation of elements is specified procedurally or via functions, and indeed, using KIF functions would sometimes have been much easier and, for the intended purposes, would have been sufficient. However, this ontology has the advantage of presenting a more explicit, declarative and original representation of element presentations. For example, all the presentation elements are not generated by functions but represented by concept types related by specialization relations or part relations (general part relations or member relations). Indeed, there are always many (kinds of) presentations for an element. This also permits to use this ontology not only for generation purposes but also for presentation checking purposes. Thus, like any grammar for a KRL fitting this presentation model (and hence the previous meta-model), the rather generic grammar of Section 4.3 seems *theoretically* derivable from this model. However, actually deriving it from this ontology also seems *difficult*.

Finally, this ontology permits to better illustrate features that are not common in classic KRLs (e.g., the extended specialization and several features of FL) and shows how they are necessary for giving a concise (and hence easier-to-understand) presentation of this ontology.

Table 4.5.2.1. Parsing and presenting of a FS_model_element

```
parsing_of_FS_model_element  < wn#parsing,
  input=> (a presentation_of_FS_model_element ?p
              language: a KRL,
              description_medium of: a FS_model_element ?e )__[?parsing->?p]
  result=> FS_model_element __[?parsing->?e],
  > (parsing_of_FS_model_element_within_FS_code_with_an_FS_parser
      agent=> a FS_parser,  parameter: a KRL ?Input_KRL,
      result=> (a presentation_of_FS_model_element_in_FS ?peInFS  language: ?Input_KRL)
    ); //The last three lines are a fake way of specifying that any FS parser uses a
       //  predefined global variable ?Input_KRL. This variable can be set to a particular
       //  language by the user. This is only a "fake way" since in FL the scope of a
       //  variable ends with the final ";". On the other hand, ideally, an FS parser would
       //  interpret the following parsing directive (assuming that the input KRL is FL):
       //  [_ parsing]_[every presentation_of_FS_model_element  language: some FCG];


presenting_of_FS_model_element  < wn#representation.activity,
  input=>  FS_model_element __[?presenting -> ?e],
  result=> (a presentation_of_FS_model_element
               description_medium of: ?e,
               language: a KRL ?Output_KRL) __[?presenting->?pe],
  > (presenting_of_FS_model_element_in_WebKB-2
      agent=> a WebKB-2,  parameter: a KRL ?Output_KRL,
      result=> (a presentation_of_FS_model_element_in_FS  language: ?Output_KRL)
    ); //The last three lines are a fake way of specifying that WebKB-2 uses a predefined
       //  global variable ?Output_KRL that users of the FS language can change.
       //  On the other hand, ideally, WebKB-2 would interpret this presentation directive:
       //  [_ presentation]_[every presentation_of_FS_model_element  language: some RDF/XML];
```

Table 4.5.2.2. Presentation of a FS model element

```
presentation_of_FS_model_element  < description_medium,
  description_medium of=> a FS_model_element, //hence the next line is left implicit
//> presentation_of_FS_code   //knowledge (re-)presentation(s)/command(s) in FS
//idem for all other kinds of presentations that are description medium of FS model elements
//however the next subtypes need to be declared explicitly
  > (presentation_of_FS_model_element_with_some_language
      //the most specialized presentation_of_FS_model_element that has no exclusion relation
      //  with a presentation in a particular language is a presentation in this language;
      //  the 'ext-comp' operator can be used to find these presentations/specializations
      > (presentation_of_FS_model_element_in_FS
          = KR_in_FS,     //a representation in a notation is a presentation
          language=> FS)  //  and may have parts in various languages (here, FS)
        (KR_in_FL_or_FCG_or_FE                                          //-> such types
          > (KR_in_FL_or_FCG                                            // need not be (and
              > (KR_in_FL  language=> FL,  > KR_in_FL_and_FCG_and_FE)    // are not) used as
                (KR_in_FCG language=> FCG, > KR_in_FL_and_FCG_and_FE))   // supertypes in
            (KR_in_FE  language=> FE,  > KR_in_FL_and_FCG_and_FE))       // tables below when
        (KR_in_KIF_or_CLIF_or_CGIF_or_CGLF                              // the language of
          > (KR_in_KIF_or_CLIF_or_CGIF                                  // the description
              > (KR_in_KIF  language=> KIF) (KR_in_CLIF  language=> CLIF) // is specified
                (KR_in_CGIF  language=> CGIF) )
            (KR_in_CGLF  language=> CGLF)
        (KR_in_RDF/XML  language=> RDF/XML) )
    (presentation_of_a_FS_model_element_with_no_subpart  //and with a known instance/subtype
       part: 0 presentation_of_FS_model_element,         //  in some KRL
       > KR_delimiter //alias 'delimiter_of_FS_model_element', see below
  part: //---- the next 5 lines are intended to state that spaces (e.g., white spaces and
        //     comments) may be used before or after any (re-)presentation of FS model element:
        { 0..* space
          (a presentation_of_FS_model_element  description_medium of: a FS_model_element)
          0..* space
        }_[kind: sequence]  __[any ?p1->every]
        //---- at least one space is necessary between two elements:
        { a presentation_of_FS_model_element    1..* space
          a presentation_of_FS_model_element
        }_[kind: sequence]  __[any ?p2->every]
        //---- delimiters:  //see Table 4.5.2.4
        0..* (. KR_delimiter  = delimiter_of_FS_model_element,
                <  (delimiter < description_medium),
                .> white_space,  //by default, any white_space can be used
                >  (KR_begin_delimiter begin_delimiter of: FS_model_element __[1<-any]
                      .> white_space) //using " " may be useful for automatic generation
                   (KR_end_delimiter  end_delimiter of: FS_model_element __[1<-any]
                      .> white_space) //using " " may be useful for automatic generation
                   (KR_separator  delimiter of: /* 2 */ FS_model_element  /*,  .> " " */)
                delimiter of=> FS_model_element  //thus, delimiters used below are subtypes
             ); // of KR_delimiter and this type is subtype of presentation_of_FS_model_element
```

Thus, given the following definition

```
    end_delimiter_of_line_comment_in_FL   .>  "\n" "@",
      < end_delimiter_of_line_comment  KR_in_FL_and_FCG_and_FE;
```

if ?Input_KRL is FL, an FL parser should take into account that a line comments ends with "\n" or "@". Ontologically, for relating `end_delimiter_of_line_comment_in_FL' to '\n' and '@', both pm#instance or pm#subtype would have been acceptable relation types. Using an extended specialization relation permits not to make a choice

between pm#instance and pm#subtype. In any case, the parser should take into account '.>' relations between delimiter types and values, not just pm#instance and pm#subtype relations. Indeed, '.>' covers string specializations and hence permits to specify that white spaces before or after the delimiter value are also acceptable by the the parser without the value having to specify this via a regular expression (see Paragraph 2.1.1.10). The next table shows some values (more precisely, some extended specializations) of pm#white_space. Within an input file, the user may specify other values for the parsing of the remainder of this input file. The next example specifies that a dot should be considered as white space but not a space character. Section 4.5.5 gives more elaborate examples of directives for parsing or presentation.

```
[_ parsing][white_space .>  "."   0 " "]
```

Table 4.5.2.3. Spaces and comments

```
space  < description_medium,
  > (white_space
       .> " "  "\n"_[language: C]  "\t"_[language: C]
          "\f"_[language: C]  "\o240"_[language: C]  " "  " ")
    comment  HTML_tag;
```

Table 4.5.2.4. Top level categories for strings and delimiters

```
string  //strings are description mediums and the default description mediums of themselves
  > {( (unquoted_string  begin_delimiter: "",  end_delimiter: "")
       (quoted_string
          > (dollar-parenthesis_delimited_string  begin_delimiter: "$(",  end_delimiter: ")$")
            (single_quoted_string  begin_delimiter: "'",  end_delimiter: "'")
            (double_quoted_string  begin_delimiter: '"',  end_delimiter: '"'))
    )}
    (delimiter
     //> KR_delimiter,  //already asserted
       > (begin_delimiter  begin_delimiter of=> description_medium,
            > KR_begin_delimiter  begin_delimiter_of_comment)
         (end_delimiter  end_delimiter of=> description_medium,
            > KR_end_delimiter  end_delimiter_of_comment)
         (separator >  KR_separator)
         (delimiter_of_comment
            > (begin_delimiter_of_comment
                 begin_delimiter of=> comment,
                 > (begin_delimiter_of_line_comment
                      >  (begin_delimiter_of_line_comment_in_FS  .> "//") )
                   (begin_delimiter_of_multiline_comment
                      >  (begin_delimiter_of_multi-line_comment_in_FS
                            .>  ("/*"  begin_delimiter of=> (a comment  end_delimiter: "*/"))
                                ("(*"  begin_delimiter of=> (a comment  end_delimiter: "*)"))
                         ) ) )
              (end_delimiter_of_comment
                 end_delimiter of=> comment,
                 > (end_delimiter_of_line_comment
                      > (end_delimiter_of_line_comment_in_FS  .> "" //no delimiter; if someone
                        ) )                       // adds a value, it will be an optional value
                   (end_delimiter_of_multiline_comment
                      >  (end_delimiter_of_multi-line_comment_in_FS  .> "*/"  "*)" ) )
              ) ) );
```

Table 4.5.2.5. Presentation of terms

```
presentation_of_term  < string,
  > {((presentation_of_informal_term  description_medium of=> informal_term,
        > { (presentation_of_informal_term_as_quoted_string  < quoted_string)
            (presentation_of_informal_term_as_unquoted_string  < unquoted_string) },
      (presentation_of_formal_term  < unquoted_string,
        description_medium=> a formal_term,
        > {( (presentation_of_formal_term_with_user_ID
               > { (presentation_of_formal_term_with_user_ID_as_prefix
                     part: {1 (. presentation_of_user_ID  description_medium of=> user_ID),
                              1 (. separator_between_key_name_and_its_source_ID_prefix_in_term
                                  < KR_separator,   .> "#")
                              1 (. presentation_of_key_name_of_term
                                  description_medium of=> key_name)
                             }_[kind: sequence] __[?p -> ? complete,  ? <= any] )
                   (presentation_of_formal_term_with_user_ID_as_postfix
                     part: {1 presentation_of_user_ID
                              1 (. separator_between_key_name_and_its_source_ID_postfix_in_term
                                  < KR_separator,   .> "\")
                              1 presentation_of_key_name_of_term
                              }__[?p -> ? complete,  ? <= any] )
                 } )
             presentation_of_formal_term_without_user_ID  = presentation_of_key_name_of_term
          )} )
    )};
```

4.5.3. Presentation Ontology of Code, Commands and Graphs

Table 4.5.3.1. Presentation of code

```
presentation_of_FS_code
  <  (KR_presentation  < description_medium),
  description_medium of=> pm#AND-set_of_statements_in_FS __[any ?p->?sfs, ?<-any],
  > { (presentation_of_FS_code_embedded_in_HTML_marks
         begin_delimiter:  $("<script .* language='(FS|FE|FCG|FL)'>")$ _[language: Perl],
         end_delimiter:  "<script>" )
      (presentation_of_FS_code_embedded_in_dollar-parenthesis_delimited_string
        < dollar-parenthesis_delimited_string );  //see Table 4.5.8.2
      (presentation_of_FS_code_without_delimiters  begin_delimiter: "",  end_delimiter: "")
    },
  part: SEQ{ 1 (. FS_code_begin_delimiter  begin_delimiter of: a presentation_of_FS_code)
                //the above two subtypes give examples of code delimiters
                //__[?p->1]  //?p is already mentioned in the embedding set
             (1 presentation_of_FS_command_with_delimiters  //next table
               description_medium of: (a command  (?Nth != last) member of: ?sfs) )
             (1 presentation_of_FS_command_with_optional_delimiters
               description_medium of: (a command  last member of: ?sfs) )
             1 (. FS_code_end_delimiter  end_delimiter of: a presentation_of_FS_code)
           } __[?p -> ? complete];
     // SEQ{...} is an abbreviation for {...}_[kind: sequence]; it is rarely used below
```

Table 4.5.3.2. Presentation of FS commands

```
presentation_of_FS_command_with_optional_delimiters
  description_medium of=> a FS_command,               //two subtypes:
  > {( presentation_of_FS_command                     //  one for a command without delimiter,
       (presentation_of_FS_command_with_delimiters    //  one for a command with delimiters
          description_medium of=> a FS_command ?cwd,
          > (presentation_of_query_with_delimiters
               end_delimiter: 1 (. query_end_delimiter .> "?" ";")),
          part: {1 (. FS_command_begin_delimiter  < KR_begin_delimiter) //default: white_space
                 1 (presentation_of_FS_command  description_medium of: ?cwd),
                 1 (. FS_command_end_delimiter  < KR_end_delimiter,
                       .> ";")  //overrided by "?" for queries and "|" for piped commands
                                //";\n\n" is better to use for automatic presentation purposes
                }_[kind: sequence] __[any ?p -> ? complete] )
    )};

    presentation_of_FS_command
      > (presentation_of_control_command   //pipe/for/if/...;  not described in this document
           description_medium of=> a control_command)
        (presentation_of_KR_graph  description_medium of=> an assertion)
        (presentation_of_query  description_medium of=> a query ?q,
           part: { (1 presentation_of_query_operator //the string for the query operator
                      description_medium of:        //  Table 4.3.2 gives examples
                                           (the query_operator direct_part of: ?q))
                   //if ?q has a set of parameters, it is now presented
                   (1 presentation_of_set_of_terms_or_KR_nodes_as_parameters
                      description_medium of: (?s  direct_part of: ?q)
                   )__[<= [?q  direct_part: a set_of_terms_or_KR_nodes ?s] ]
                 }_[kind: sequence] __[any ?p -> ? complete] );

        presentation_of_set_of_terms_or_KR_nodes_as_parameters
          description_medium of=> set_of_terms_or_KR_nodes ?s,
          := (a sequence  size: (a size  size of: ?s),
                ^Nth member:  //^Nth is a free variable (hence, universally quantified)
                   {1 (. begin_delimiter_of_term_or_KR_node_as_parameter < KR_begin_delimiter)
                    (a presentation_of_term_or_KR_node
                      description_medium of: (the term_or_KR_node  ^Nth member of: ?s)
                   }_[kind: sequence] );

            presentation_of_term_or_KR_node
              > (presentation_of_term description_medium of=> a term) //the string of the term
                (presentation_of_KR_node  description_medium of=> a KR_node,
                   > presentation_of_KR_graph  presentation_of_relation_node
                     presentation_of_concept_node  presentation_of_relation_node_core);
```

Table 4.5.3.3. Presentation of a graph

```
presentation_of_KR_graph  //relation by relation, as in KIF and CLIF, or concept by concept
  > {relation_centric_presentation_of_KR_graph  //see next table
     concept_centric_presentation_of_KR_graph   //as in frames
    }; //incomplete partition since a mixed approach is possible, e.g., as in FCG

    concept_centric_presentation_of_KR_graph  //see tables 4.5.1.2 – 4.5.1.5 for the structure
      description_medium of=> (KR_graph direct_part: (a set_of_concept_nodes
                                1st member: a concept_node ?c1)) __[any ?pg->?g],
      exclusion: KR_in_KIF_or_CLIF_or_CGIF,  //only relation_centric_presentation_of_KR_graph
      part: SEQ{1 presentation_of_concept_graph  description_medium of: ?c1}__[?pg->? complete];

      presentation_of_concept_graph  //concept node seen as a graph, with its links+context
        > {( presentation_of_concept_graph_with_delimiters
             presentation_of_concept_graph_without_delimiters )}
          (presentation_of_concept_graph_with_set_of_context_links
             exclusion: KR_in_CGLF  KR_in_RDF/XML), //only in FL, FCG and FE
             > presentation_of_concept_graph_with_context_links_before   //both can be used in
               presentation_of_concept_graph_with_context_links_after),  // FL, FCG and FE
        description_medium of=> concept_node __[any ?pc->?c],
        part: { {1 (. begin_delimiter_for_prefix_context_links  < KR_begin_delimiter, .> "[_")
                  (1 presentation_of_set_of_link nodes  description_medium of: ?sl)
                   1 (. end_delimiter_for_context_links  < KR_end_delimiter,  .> "]")
                }_[kind: sequence] __[?pc -> 0..1 complete,  //may or may not be "here"
                      <= AND{[?c direct_part of: a set_of_context_links_on_a_concept_graph ?sl]
                             [?p kind: presentation_of_concept_graph_with_context_links_before]} ]
                1 (. begin_delimiter_of_concept_graph  < KR_begin_delimiter,
                    > (begin_delimiter_of_concept_graph_in_FCG  < KR_in_FCG KR_in_FL, .> "[")
                      (begin_delimiter_of_concept_graph_in_FE   < KR_in_FE,  .> "`")
                      (begin_delimiter_of_concept_graph_in_CGLF < KR_in_CGLF,  .> "")
                  )__[<= [?pg  kind: presentation_of_concept_graph_with_delimiters] ]
                (1 presentation_of_concept_node description_medium of: ?c) //with links
                1 (. end_delimiter_of_concept_graph  < KR_end_delimiter,
                    > (end_delimiter_of_concept_graph_in_FCG  < KR_in_FCG KR_in_FL, .> "]")
                      (end_delimiter_of_concept_graph_in_FE   < KR_in_FE,  .> "'")
                      (end_delimiter_of_concept_graph_in_CGLF < KR_in_CGLF,  .> "")
                  )__[<= [?pg  kind: presentation_of_concept_graph_with_delimiters] ]
                {1 (. begin_delimiter_for_postfix_context_links  < KR_begin_delimiter, .> "_[")
                  (1 presentation_of_set_of_link nodes  description_medium of: ?sl)
                  1 end_delimiter_for_context_links
                }_[kind: sequence] __[?pc -> 0..1 complete,  //may or may not be "here"
                       <= AND{[?c direct_part of: ?sl]
                              [?p kind: presentation_of_concept_graph_with_context_links_after]}]
              }_[kind: sequence] __[?pc -> ? complete];

      presentation_of_concept_node  //concept node (with its links except its context links)
        description_medium of=> (concept_node  direct_part: a set_of_concept_links ?s
                                  )__[any ?pc->?c],
        part: { 1 (. begin_delimiter_of_concept_node  < KR_begin_delimiter,  .> "",
                    > (begin_delimiter_of_concept_node_in_CGLF  <  KR_in_CGLF,  .> "["))
                  (1 presentation_of_concept_node_core  description_medium of: ?c)
                  (1 presentation_of_set_of_link_nodes  description_medium of: ?s)
                  1 (. end_delimiter_of_concept_node  < KR_end_delimiter,  .> " ",
                      > (end_delimiter_of_concept_node_in_CGLF  <  KR_in_CGLF,  .> "]"))
                }_[kind: sequence] __[?pc -> ? complete];
```

4.5.4. Presentation Ontology of Relation Nodes

Table 4.5.4.1. Relation centric presentation of a graph or relation node

```
//As in the grammar of Section 4.3, only the basic structure for the relation centric
//  approach is yet developed here; KIF elements are not yet presented.

relation_centric_presentation_of_KR_graph
  exclusion:  KR_in_CGLF  KR_in_FL  KR_in_FE, //only concept_centric_presentation_of_KR_graph
  description_medium of=> (a KR_graph  direct_part: a set_of_relation_nodes ?s),
  := (a sequence  size: (a size  size of: ?s),
        ^Nth member:
            {1 (. begin_delimiter_of_set_of_relation_nodes < KR_begin_delimiter)
                        //white space (the default value for KR_begin_delimiter)
             (1 relation_centric_presentation_of_relation_node
                description_medium of: (the relation_node  ^Nth member of: ?s)
            }_[kind: sequence] );

  relation_centric_presentation_of_relation_node
    description_medium of=> relation_node __[any ?p->?r],
    > (presentation_of_relation_with_predefined_relation_type_position
         exclusion: KR_in_FL_or_FCG_or_FE ,
         > (prefix_presentation_of_relation_node //relation type always first, e.g., as in KIF
              > KR_in_KIF   KR_in_CLIF   KR_in_CGIF)
           (postfix_presentation_of_relation_node //relation type always last; ever been used?
           (predefined_infix_presentation_of_relation_node
              > infix_presentation_of_binary_relation_node) )
      (presentation_of_relation_with_variable_relation_type_position
         exclusion: KR_in_KIF_or_CLIF_or_CGIF_or_CGLF),
    part: { 1 (. begin_delimiter_of_relation_node_with_predefined_relation_type_position
                < KR_begin_delimiter,  .> "("
              )__[<= [?p kind: presentation_of_relation_with_predefined_relation_type_position
                     ]]
            1 (. begin_delimiter_of_relation_node_with_variable_relation_type_position
                < KR_begin_delimiter,  .> "(:"
              )__[<= [?p kind: presentation_of_relation_with_variable_relation_type_position
                     ]]
            (1 presentation_of_rel_type/lambda
               description_medium of:  //the case of many/no relation types is not yet handled
                 (a rel_type/lambda  direct_part of: (the set_of_relation_types
                      direct_part of: (the relation_node_core  direct_part of: ?r))))
            )__[<= [?p  not kind: prefix_presentation_of_relation_node] ]
                   //1st by default if not postfix (see below)
            (1 relation_centric_presentation_of_set_of_connected_KR_nodes  //Table 4.5.4.9
               description_medium of:
                 (the set_of_connected_KR_nodes ?sc  direct_part of: ?1st_rel)
            )__[<= [?sc  direct_part: a relation_node ?1st_rel] ]
            (1 presentation_of_rel_type/lambda  description_medium of:
                 (a rel_type/lambda  direct_part of: (the set_of_relation_types
                      direct_part of: (the relation_node_core  direct_part of: ?r))))
            )__[<= [?p  kind: relation_presentation_postfix_mode] ] //last if postfix
            1 (. end_delimiter_of_relation_node  < KR_end_delimiter,  .> ")")
          }_[kind: sequence] __[?p -> ? complete];
```

Table 4.5.4.2. Presentation of link nodes

```
presentation_of_set_of_link_nodes
  > presentation_of_set_of_link_nodes_with_sharing_of_link_core,  //as in FL
  description_medium of=> (set_of_link_nodes  size: a size ?lsl) __[any ?psl->?sl],
  := (a sequence  size: (a size =< ?lsl),  //"=<" to allow prefix and postfix context links
       1st member:
            { (1 presentation_of_link_node  //defined in a table below
                description_medium of: (a link_node  1st member of: ?sl))
            }_[kind: sequence] __[<= [?psl  not kind:
                                 presentation_of_set_of_link_nodes_with_sharing_of_link_core] ]
            {1 begin_delimiter_of_link_nodes
             (1 presentation_of_set_of_link_nodes_with_same_type_and_direction
                description_medium of: (the set_of_link_nodes_with_same_type_and_direction
                          1st subset of: ?sl) //according to the model (see Table 4.5.1)
            }_[kind: sequence] __[<= [?psl  kind:  //as in FL
                                 presentation_of_set_of_link_nodes_with_sharing_of_link_core]],
       ( ^Nth != 1 ) member:
            {1 (. separator_of_link_nodes  < KR_separator,  .> ",")
             (1 presentation_of_link_node
                description_medium of: (the link_node  ^Nth member of: ?sl))
            }_[kind: sequence] __[<= [?psl  not kind:
                                 presentation_of_set_of_link_nodes_with_sharing_of_link_core] ]
            {1 separator_of_link_nodes
             (1 presentation_of_set_of_link_nodes_with_same_type_and_direction
                description_medium of: (the set_of_link_nodes_with_same_type_and_direction
                                          ^Nth subset of: ?sl))
            }_[kind: sequence] __[<= [?psl  kind:
                                 presentation_of_set_of_link_nodes_with_sharing_of_link_core] ]
     );


presentation_of_set_of_link_nodes_with_same_type_and_direction
  description_medium of=> (a set_of_link_nodes_with_same_type_and_direction ?sl
                             size: ?lsl),
  := (a sequence  size: (a size =< ?lsl),  //"=<" to allow prefix and postfix context links
       1st member:
            {1 (. begin_delimiter_of_link_nodes_with_same_type_and_direction  .> "",
                 < KR_begin_delimiter)
             (1 presentation_of_link_node  description_medium of: ?l1)
             )__[<= [?sl kind: presentation_of_set_of_links_with_same_type_and_direction]]
            }_[kind: sequence],
       ( ^Nth != 1 ) member:  //presentation of the destinations of the other links
            {1 separator_of_link_nodes,  // .> ""  //already asserted
             1 (presentation_of_link_node_with_implicit_link_core
                 description_medium of: (a link_node  ^Nth member of: ?sl))
            }_[kind: sequence];
```

As opposed to a functional specification of the presentation of a graph, this relational specification does not mandate one particular presentation order for the links, i.e., one particular graph traversal order. In any particular concept-oriented presentation ?pg of a graph ?g, each relation ?r is presented via an out-link or an in-link and cannot later appear again in ?g. This is ensured in the next two table by the setting of predecessor_presentation relations between each concept node (more generally, each KR_node) and the relation nodes presented after it, and by stating that there must be a path of predecessor relations between each concept/relation node (except the first) and the first presented concept node.

Table 4.5.4.3. Presentation of a link node

```
predecessor_presentation .(description_medium ?x, description_medium ?y)  < predecessor;

previous_presentation .(description_medium ?x, description_medium ?y)
  := [ [?x predecessor_presentation: ?y] or: [?x previous_presentation: ?y] ];  //or:
//:= [?x  (predecessor_presentation: a description_medium)* predecessor_presentation: ?y];



presentation_of_link_node
  > presentation_of_link_node_with_implicit_link_core,
  description_medium of=>
    (link_node  direct_part: (a link_node_core ?lCore  direct_part: a relation_node ?r)
                           (a set_of_link_other_KR_nodes ?sok),
                member of: (a set_of_link_nodes direct_part of: a KR_node ?k)
    )__[any ?pln->?ln]
    ?r __[any ?pln->?],  //this presentation of ?ln is also a presentation of ?r
  part of:
    (presentation_of_KR_graph
       description_medium of: (a KR_graph ?g  direct_part: (a set_of_concept_nodes
                                                             1st member: a concept_node ?c1)),
       part: (?pc1  description_medium of: ?c1,
             (any (KR_node part of: ?g)      //there is a path of predecessor_presentation
                previous_presentation of: ?pc1)  //  relations between any KR_node and ?c1
             (?pr  description_medium of: ?r ?ln,  not previous_presentation: ?pr)
             (?pk  description_medium of: ?k,  not previous_presentation: ?pk ?pr)
    )__[?pln->?pg],                              //  and this path is unique
  part: { (1 presentation_of_link_node_core  description_medium of: ?lCore
          )__[<= [?pln  kind: presentation_of_link_node_with_implicit_link_core] ]
          (1 presentation_of_link_other_KR_nodes  description_medium of: ?sok)
        }_[kind: sequence] __[?pln -> ? complete,
                                    <=  [?pr  predecessor_presentation: ?pk] ],
  part: "" __[?pln -> ? complete, <= [?pr  not predecessor_presentation: ?pk] ];
```

Table 4.5.4.4. Presentation of the destination(s) of a link

```
presentation_of_link_other_KR_nodes
  > presentation_of_context_after_link_other_KR_nodes,
  description_medium of=>
    (set_of_link_other_KR_nodes ?sok  direct_part of:
        (a link_node ?l  member of: (a set_of_link_nodes  direct_part: a KR_node ?k)),
                         direct_part: (a link_node_core  direct_part of: (a relation_node ?r
                             direct_part: (a set_of_connected_KR_nodes ?sk  ^Nk member: ?k)))
    )__[any ?p->?sok],
  part of: presentation_of_KR_graph __[?p->?pg],
  part: { 1 (. begin_delimiter_of_binary_out-link_arc_2  < KR_begin_delimiter,  .> "",
               >  (begin_delimiter_of_binary_out-link_arc_2_in_CGLF < KR_in_CGLF,  .> "->")
            )__[<= [?l  kind: out-link_node] ]
          1 (. begin_delimiter_of_binary_in-link_arc_2  < KR_begin_delimiter,  .> "",
               >  (begin_delimiter_of_binary_in-link_arc_2_in_CGLF < KR_in_CGLF,  .> "<-")
            )__[<= [?l  kind: in-link_node] ]
          (1 presentation_of_KR_node  //the one != ?k, member of: ?sk, or equivalently:
             description_medium of: (the KR_node member of: ?sok)  // the only one in ?sok
             predecessor_presentation: (the presentation_of_link_node
                                          part of: ?pg,  description_medium of: ?r))
          {1 (. begin_delimiter_for_a_set_of_context_links_on_link_core_after_its_destination
                < KR_begin_delimiter, .> "__[")
           (1 presentation_of_set_of_link nodes  description_medium of: ?sl)
           1 end_delimiter_for_context_links
          }_[kind: sequence] __[?p -> 0..1 complete,  //may or may not be "here"
                   <= AND{[?r direct_part of: a set_of_context_links_on_a_relation ?sl]
                          [?p kind: presentation_of_context_after_link_other_KR_nodes] }]
        }_[kind: sequence] __[?pln -> ? complete, <= [?sok size: (?lsok = 1)] ]

        //now, the case of non-binary links; in CGLF, the "-{" has now already been presented
        { (1 sequence  size: (a size  size of: ?sk), // > 2
            ^Nk member: "", //empty element (for the size to be the size of ?sk)
            ( ^Nth != ^Nk ) member:
               {1 (. begin_delimiter_of_non-binary_link_arc  < KR_begin_delimiter,
                    > (begin_delimiter_of_non-binary_link_arc_in_CGLF < KR_in_CGLF,  .> "-"))
                (1 description_medium  description_medium of: ^Nth)
                1 (. end_delimiter_of_non-binary_link_arc  < KR_end_delimiter,
                    > (begin_delimiter_of_non-binary_link_arc_in_CGLF < KR_in_CGLF,  .> "->"))
                (1 presentation_of_KR_node
                  description_medium of: (the KR_node  ^Nth member of: ?sk),
                  predecessor_presentation: (the presentation_of_link_node
                                               part of: ?pg,  description_medium of: ?r))
               }_[kind: sequence] __[? -> ? complete]
          1 (. end_delimiter_of_non-binary_link_end  < KR_end_delimiter,
              >  (end_delimiter_of_non-binary_link_end_in_CGLF < KR_in_CGLF,  .> "}"))
          {1 begin_delimiter_for_a_set_of_context_links_on_link_core_after_its_destination
           (1 presentation_of_set_of_link nodes  description_medium of: ?sl)
           1 end_delimiter_for_context_links
          }_[kind: sequence] __[?p -> 0..1 complete,  //may or may not be "here"
                   <= AND{[?r direct_part of: ?sl]
                          [?p kind: presentation_of_context_after_link_other_KR_nodes] }]
        }_[kind: sequence] __[?pln -> ? complete, <= [?lsok > 1] ];
```

Table 4.5.4.5. Presentation of a link node core

```
presentation_of_link_node_core
  > (presentation_of_link_core_with_set_of_context_links  //in F*, not in CGLF, CGIF, KIF, ...
      exclusion: KR_in_KIF_or_CLIF_or_CGIF_or_CGLF,
      > presentation_of_link_core_with_context_links_before  //FL, FCG and FE allow
        presentation_of_link_core_with_context_links_after)  //  these two subtypes
    {( presentation_of_binary_link_core
      (presentation_of_non-binary_link_core  exclusion: KR_in_FL_or_FCG_or_FE) )},
  description_medium of=> (link_node_core  direct_part of: a link_node ?l) __[any ?p -> ?lc],
  direct_part:
    { {1 begin_delimiter_for_prefix_context_links
       (1 presentation_of_set_of_link nodes  description_medium of: ?sl)
       1 end_delimiter_for_context_links
      }_[kind: sequence] __[?p -> 0..1 complete,  //may or may not be "here"
                    <= AND{ [?lc direct_part of: a set_of_context_links_on_a_relation ?sl]
                            [?p kind: presentation_of_link_core_with_context_links_before] }]
      1 binary_out-link_header
        __[<= AND{ [?l  kind: out-link] [?lc direct_part of: binary_relation_node] }
      1 binary_in-link_header
        __[<= AND{ [?l  kind: in-link] [?lc  direct_part of: binary_relation_node] }
      1 non-binary_link_header __[<= [?l  direct_part of: non-binary_relation_node]]
      (1 presentation_of_set_of_type_references_or_lambda_abstractions
          description_medium of: (a set_of_type_references_or_lambda_abstractions
                                    direct_part of: ?lc) )
      (1 presentation_of_coreference_variable  description_medium of: ?v
            //this assumes that the stored variable is a string (e.g., the provided string)
      )__[<= [?lc  direct_part: (a relation_variable ?v  direct_part of: ?lc)]]
      1 binary_out-link_core_end_with_presentation_of_assertion/definition_mark
         description_medium of: (a assertion/definition_mark direct_part of: ?lc))
        __[<= AND{ [?l kind: out-link] [?lc direct_part of: binary_relation_node] }
      1 binary_in-link_core_end_with_presentation_of_assertion/definition_mark
        __[<= AND{ [?l kind: in-link] [?lc direct_part of: binary_relation_node] }
      1 non-binary_link_core_end __[<= [?l direct_part of: non-binary_relation_node]]
      {1 begin_delimiter_for_postfix_context_links
       (1 presentation_of_set_of_link nodes  description_medium of: ?sl)
       1 end_delimiter_for_context_links
      }_[kind: sequence] __[?p -> 0..1 complete,  //may or may not be "here"
                     <= AND{ [?lc direct_part of: ?sl]
                             [?p kind: presentation_of_link_core_with_context_links_after] }]
    }_[kind: sequence]  __[?p -> ? complete];
```

Table 4.5.4.6. Link headers

```
binary_out-link_header  < KR_begin_delimiter,  .> "",
  > (binary_out-link_header_in_CGLF  < KR_in_CGLF,  .> "->(")
    (binary_out-link_header_in_FL_and_FCG_and_FE  < KR_in_FL_and_FCG_and_FE,
       direct_part of: (presentation_of_link_node_core
                           description_medium of: a link_node_core ?lc) __[any ?p -> ?],
       part: {1 (. header_of_out-link_core_with_modal_possibility  .>  ""  "may have for")
                __[<= [?lc direct_part of: (a relation_node  direct_part:
                          (a set_of_context_links_on_a_relation ?scl  member:
                            (a link_node  direct_part: (a relation_node_core  direct_part:
                               (a set_of_relation_types ?st  member: rfol_for_modal_possibility
                               ))))))]]
              1 (. header_of_outlink_core_with_physical_possibility  .>  ""  "can have for")
                __[<= [?st  member: rfol_for_physical_possibility] ]
              1 (. header_of_outlink_core_without_possibility  .>  ""  "has"  "has for")
                __[<= [?st  not member: rfol_for_modality] ]
             }_[kind: sequence]  __[?p -> ? complete]  );


binary_in-link_header  < KR_begin_delimiter,  .> "",
  > (binary_in-link_header_in_CGLF  < KR_in_CGLF,  .> "<-(")
    (binary_in-link_header_in_FL_and_FCG_and_FE   < KR_in_FL_and_FCG_and_FE,
       direct_part of: (presentation_of_link_node_core
                           description_medium of: a link_node_core ?lc) __[any ?p -> ?],
       part: {1 (. header_of_in-link_core_with_modal_possibility  .>  ""  "may be")
                __[<= [?lc direct_part of: (a relation_node  direct_part:
                          (a set_of_context_links_on_a_relation ?scl  member:
                            (a link_node  direct_part: (a relation_node_core  direct_part:
                               (a set_of_relation_types ?st  member: rfol_for_modal_possibility
                               ))))))]]
              1 (. header_of_in-link_core_with_physical_possibility  .>  ""  "can be")
                __[<= [?st  member: rfol_for_physical_possibility] ]
              1 (. header_of_in-link_core_without_possibility  .>  ""  "is")
                __[<= [?st  not member: rfol_for_modality] ]
             }_[kind: sequence]  __[?p -> ? complete]  );


non-binary_link_header  .> "",
  direct_part of:
     (presentation_of_link_node_core  description_medium of:
           (a link_node_core  direct-part of: (a link_node  direct_part of:
                                        a set_of_link_nodes direct_part of: a KR_node ?k),
                               direct_part of: (a relation_node direct_part:
                                        (a set_of_connected_KR_nodes ?sk  ^Nk member: ?k))
           )) __[any ?p -> ?],
  part: {1 (. begin_delimiter_of_non-binary_link_header  < KR_begin_delimiter,
              >  (begin_delimiter_of_non-binary_link_header_in_CGLF < KR_in_CGLF,  .> "<-"))
         (1 description_medium  description_medium of: ^Nk)
         1 (. end_delimiter_of_non-binary_link_header  < KR_begin_delimiter,
              >  (end_delimiter_of_non-binary_link_header_in_CGLF < KR_in_CGLF,  .> "-"))
        }_[kind: sequence]  __[?p -> ? complete];
```

Table 4.5.4.7. Link core ends

```
non-binary_link_core_end  < KR_end_delimiter,  .> "",   exclusion: KR_in_FL_and_FCG_and_FE,
  > (non-binary_link_core_end_in_CGLF < KR_in_CGLF,  .> "-{");


binary_out-link_core_end_with_presentation_of_assertion/definition_mark  < KR_end_delimiter,
  .> "",   description_medium of=> an assertion/definition_mark ?m,
  > (binary_out-link_end_in_CGLF  < KR_in_CGLF,  .> ")->")
    (binary_out-link_end_in_FL_and_FCG_and_FE  < KR_in_FL_and_FCG_and_FE,
       part: { (1 presentation_of_assertion/definition_mark  description_medium of: ?m)
             }_[kind: sequence]  __[?p -> ? complete] );


binary_in-link_core_end_with_presentation_of_assertion/definition_mark  < KR_end_delimiter,
  .> "",   description_medium of=> an assertion/definition_mark ?m,
  > (binary_in-link_end_in_CGLF  < KR_in_CGLF,  .> ")<-")
    (binary_out-link_end_in_FL_and_FCG_and_FE  < KR_in_FL_and_FCG_and_FE,
       part: { (1 (end_delimiter_of_direction_for_binary_in-link_node-core  > "of",
                     < KR_end_delimiter  KR_in_FL_and_FCG_and_FE)
               (1 presentation_of_assertion/definition_mark  description_medium of: ?m)
             }_[kind: sequence]  __[?p -> ? complete] );

   presentation_of_assertion/definition_mark
     description_medium of=> a assertion/definition_mark,
     < KR_end_delimiter  KR_in_FL_and_FCG_and_FE,
     > { (presentation_of_mark_of_simple_assertion  .> ":",
            description_medium of=> a mark_of_simple_assertion)
         (presentation_of_mark_of_assertion_of_sufficient_condition  .> "<=",
            description_medium of=> a mark_of_sufficient_condition)
         (presentation_of_mark_of_mandatory_destination  .> "=>",
            description_medium of=> a mark_of_mandatory_destination)
         (presentation_of_mark_of_full_definition  .> ":=",
            description_medium of=> a mark_of_full_definition)
         (presentation_of_mark_of_definition_of_sufficient_condition  .> ":<=",
            description_medium of=> a mark_of_sufficient_condition)
         (presentation_of_mark_of_definition_of_necessary_condition  .> ":=>",
            description_medium of=> a mark_of_necessary_condition)
       };
```

Table 4.5.4.8. Presentation of type references or lambda abstractions

```
presentation_of_set_of_type_references_or_lambda_abstractions
  description_medium of=> (a set_of_type_references_or_lambda_abstractions ?s),
  := (a sequence  size: (a size ?sl  size of: ?s),
       1st member:
        {1 (. begin_delimiter_for_presentation_of_set_of_types_or_lambda_abstractions  .> "(",
              < KR_begin_delimiter) __[<= [?sl > 1]]   //"(" in FL, FE or FCG; optional in FCG
                                 //in FL and FE, it is required for their grammars to be LALR(1)
         (1 presentation_of_type_reference_or_lambda_abstraction
            description_medium of: (a type_reference_or_lambda_abstraction 1st member of: ?s))
        }_[kind: sequence],
       ( ^Nth != {1, last}) member:
        {1 (. separator_between_type_references_or_lambda_abstractions  < KR_separator, .> "")
         (1 presentation_of_type_reference_or_lambda_abstraction
           description_medium of: (a type_reference_or_lambda_abstraction ^Nth member of: ?s))
        }_[kind: sequence],
       last member:
        {1 separator_between_type_references_or_lambda_abstractions
         (1 (. presentation_of_type_reference_or_lambda_abstraction
              > {( (presentation_of_type_reference  < presentation_of_term)
                   presentation_of_lambda_abstraction )} )
           description_medium of: (a type_reference_or_lambda_abstraction last member of: ?s))
         1 (. end_delimiter_for_presentation_of_set_of_types_or_lambda_abstractions  .> ")",
              < KR_end_delimiter) __[<= [?sl > 1]]   //")" in FL, FE or FCG; optional in FCG
        }_[kind: sequence]
     );
```

Table 4.5.4.9. Relation centric presentation of set of connected KR nodes

```
relation_centric_presentation_of_set_of_connected_KR_nodes
  description_medium of=> a set_of_connected_KR_nodes ?s,
  := (a sequence  size: (a size  size of: ?s),
        ^Nth member:
            {1 (. begin_delimiter_of_KR_nodes_in_set  < KR_begin_delimiter) //white space
             (1 presentation_of_KR_node
                description_medium of: (the KR_node  ^Nth member of: ?s)
            }_[kind: sequence] );
```

4.5.5. Start of Presentation Ontology of Concept Nodes and Examples of Parsing/Presentation Control

For the following reasons, this subsection only presents the very beginning of the presentation ontology of concept nodes: Table 4.5.5.1.

- Fully developed, this ontology would be longer than the presentation ontology of relation nodes given in the previous subsection.
- All the mechanisms to build it have now been presented and this ontology would not provide a new viewpoint compared to the structures presented in the tables 4.3.4 to 4.3.7.
- It would also not provide many interesting terms to re-use for parsing control.

The specifications of the presentation of definitions, lambda-abstractions, relation/concept signatures and function parameters are meant to be included in this presentation ontology of concept nodes.

Table 4.5.5.1. Presentation of the core of a concept node

```
presentation_of_concept_node_core
  > {presentation_of_concept_node_core_with_types_before_the_referent
     (presentation_of_concept_node_core_with_quantifiers_first
         > FCG/FE/FL_presentation_of_concept_node_core  //quantifiers, type(s), all the rest
     ) },
  description_medium of=> a concept_node_core;


  presentation_of_concept_node_core_with_types_before_the_referent
    > (CGLF/CGIF_presentation_of_concept_node_core      //type(s) ":" referent
       < KR_in_CGLF  KR_in_CGIF),
    exclusion: KR_in_FL_or_FCG_or_FE,
    description_medium of=> concept_node_core __[any ?p -> ?c],
    direct_part:
      { (1 presentation_of_set_of_concept_type_references_or_lambda_abstractions
          description_medium of: (a set_of_concept_type_references_or_lambda_abstractions
                                   direct_part of ?c) )
        (1 presentation_of_concept_referent
          description_medium of: (a concept_referent  direct_part of ?c) )
      }_[kind: sequence]  __[?p -> ? complete];
```

The next table lists the presentation approaches declared in the last two subsections and the previous table. Some approaches correspond to a syntactic feature (e.g., 'relation-centric_presentation' and the prefix presentations) but most also correspond to a logical feature. Some examples of features not listed here are: the use of collection nodes, the definition of quantifiers, the representation of measures in a way similar to the use of numerical quantifiers, etc.

Table 4.5.5.2. Types of presentation approaches declared above

```
relation_centric_presentation_of_KR_graph
  presentation_of_relation_with_predefined_relation_type_position
    prefix_presentation_of_relation_node
    postfix_presentation_of_relation_node
    predefined_infix_presentation_of_relation_node
      infix_presentation_of_binary_relation_node
  presentation_of_relation_with_variable_relation_type_position
concept_centric_presentation_of_KR_graph
  presentation_of_concept_graph_with_delimiters     //as in FCG
  presentation_of_concept_graph_without_delimiters  //as in CGLF
  presentation_of_concept_graph_with_context_links
    presentation_of_concept_graph_with_context_links_before
    presentation_of_concept_graph_with_context_links_after
  presentation_of_set_of_link_nodes_with_sharing_of_link_core
  presentation_of_link_core_with_set_of_context_links  //only in FL, FCG and FE
    presentation_of_link_core_with_context_links_before
    presentation_of_link_core_with_context_links_after
    presentation_of_context_after_link_other_KR_nodes
  presentation_of_non-binary_link_core
  presentation_of_concept_node_core_with_types_before_the_referent
  presentation_of_concept_node_core_with_quantifiers_first
```

The next table shows how parsing directives re-using some of the above introduced types of delimiters could permit to dynamically modify the currently used notation (here, an FL/FCG/FE like notation) if a parser took those directives into account. Supporting these simple kinds of parsing directives requires modifications to only the lexical sub-parser.

Table 4.5.5.3. How the above presented delimiters could be used to dynamically modify an FL-like language

```
[_ parsing][
 AND{[presentation_of_FS_code  begin_delimiter: "@(", end_delimiter:  ")@"],
     [multi-line_comment  .> ("/^"  begin_delimiter of=> (a comment  end_delimiter: "^/"))],
     [begin_delimiter_for_prefix_context_links  .> "{_"]
     [begin_delimiter_for_a_set_of_context_links_on_link_core_after_its_destination  .> "__{"]
     [end_delimiter_for_context_links  .> "}"]
     [begin_delimiter_of_concept_graph .> "$["]
     [begin_delimiter_of_concept_node  .> "%["]
     [end_delimiter_of_concept_node    .> "]"]
     [begin_delimiter_of_relation_node_with_variable_relation_type_position  .> "(~"]
     [end_delimiter_of_direction_for_binary_in-link_node-core  > "de"] //French for "of"
     [header_of_in-link_core_without_possibility  .> "" "est"] //French for "is"
     [header_of_outlink_core_without_possibility  .> "" "a" "a pour"] //French for "has for"
     [begin_delimiter_for_presentation_of_set_of_types_or_lambda_abstractions  .> "(%"]
    }];
```

As seen in Paragraph 2.1.1.19, parsing directives can be context dependent:

```
[(pm#formal_term  pm#part of: pm#relation_node) pm#default_creator: pm]_[parsing];
```

(Here, the lexical sub-parser must know which node has just been grammatically parsed; this is not difficult to implement).

This last directive allows the use of a formal term without explicit prefix or suffix specifying its creator. This is because by default unquoted strings only represent formal terms:

```
[unquoted_string  kind: presentation_of_formal_term __[any=>., .<=any]
]_[parsing];  //parsing default; parsing definitions can be overrided (Paragraph 2.1.1.19)
```

However, in certain cases, e.g., when many natural languages sentences have to be represented, the use of unquoted strings for informal terms (and mandatory prefixes/postfixes for formal terms) is more handy and readable. In the current version of FS, this is allowed by the special command 'every_unprefixed_term_is_informal;' (the converse command being 'every_unprefixed_term_is_formal;') which sets a variable that is then exploited by term parsing procedures. Ideally, for genericity and knowledge sharing purposes, instead of this predefined command the statement '[every_unprefixed_term_is_formal] _[parsing];' would be given and the following definition would be taken into account by the parser:

```
every_unprefixed_term_is_formal
  := [every (unquoted_string  not kind: presentation_of_formal_term_with_user_ID)
        kind: presentation_of_informal_term];
```

The definitions in Table 4.5.2.5 can (at least in theory) permit a parser to be sure that if an unquoted string has a user-ID as prefix or postfix it is of kind `presentation_of_formal_term_with_user_ID' (because of the '? <= any' cardinalities, highlighted via bold characters in this table).

With the use of informal terms or unprefixed formal terms, ambiguities often arise and some of them may not be automatically solved using relation signatures and other existing constraints. Hence, there should exist a parsing directive to allow or not a certain number of (unsolved) ambiguities. In the current version of FS, this is allowed by the special command 'ambiguity_acceptation N;' where N stands for a positive integer. Ideally, instead of this predefined command the statement '[any_FS_element_has_at_most_N_ambiguous_terms _(5)] _[parsing];' would be given and the following definition would be taken into account by the parser:

```
every_FS_element_has_at_most_N_ambiguous_terms .(?N)
  := [every (parsing_of_FS_model_element  //see Table 4.5.2.1
             input:  (a FS_model_element  part: 1..* presentation_of_informal_term))
        result: (a FS_model_element  part: at most ?N informal_term)];
```

RDF/XML allows the use of terms before their declarations or definitions, i.e., before they are typed or supertyped. Doing so when this can be avoided is a bad practice since this prevents early validation, does not force the knowledge provider to explore and insert its formal terms in the ontology before using them (and thus, may lead him/her to create partial redundancies or make inadequate representation choice), and decreases the readability and normalization of the input file containing the code. Except for the use of certain relation types, this can always be avoided. Thus, by default, the current version of FS does not allow forward declaration and provides special commands such as "a_link_type_may_be_forward_declared;" and "no_term_may_be_forward_declared;". These two commands should be used respectively before and after a statement using a link with a yet undeclared type. Instead, ideally, a FS parser would take into account parsing directives re-using definitions such as the following ones:

```
no_term_may_be_forward_declared
  := [every (formal_term_presentation  description_medium: a formal_term ?t)
       previous_presentation: 0 (presentation_of_term_definition
                                  description_medium: (a definition  definition of: ?t))];
a_link_type_may_be_forward_declared
  := [every (presentation_of_relation_type  part of: a presentation_of_link_node_core,
                                            description_medium: a formal_term ?t)
       previous_presentation: 0..1 (presentation_of_term_definition
                                    description_medium: (a definition  definition of: ?t))];
//However, for these last definitions to work as intended, more `predecessor_presentation'
//  relations should be specified by the definitions in the previous subsection
```

Similarly, to specify that HTML tags should be considered as spaces, like comments and white spaces (as is the case in WebKB-2 for parsing and presentation purposes, except when RDF/XML is used):

```
[HTML_tag  not < space]_[parsing];
```

By replacing _[parsing] with _[presentation], directives such as the above ones should ideally be usable for changing default presentation directives. Similarly to parsing directives and as is currently the case in WebKB-2,

- presentation directives entered in a form within a browser (or, generated via such a form) and sent to the server would change the presentation of the results of follow-up queries sent from the same browser (in WebKB-2, this is implemented by storing the default presentation parameters in each generated Web page: form and/or query results),
- presentation directives in an input file change the presentation of the results of subsequent queries made in this file.

There are presentation directives the core statement of which cannot also be used for parsing directives. As an example, as noted in Table 4.5.2.1, the following presentation directive for specifying the output language should ideally be taken into account (assuming that the current input KRL is FL):

```
[_ presentation]_[every presentation_of_FS_model_element  language: some RDF/XML];
```

Ideally too, the ontology used by the (re-)presentations could be similarly specified.

In the same line, to specify that no KR node from a certain source should be presented in query results, a presentation directive and related definition could be:

```
no_such_KR_node_from_such_source
  _(?KR_node_kind, (a person believer of: `any bird is agent of: a flight'));
no_such_KR_node_from_such_source .(?KR_node_kind, ?term_or_KR_node_for_a_user_ID)
  := [every (?KR_node_kind  source_or_creator: ?term_or_KR_node_for_a_user_ID)
       description_medium: ""];  //this use of a statement as parameter may need additional
      // syntactic sugar even though no special one is required in KIF for this simple case
```

A definition permitting to specify that no creator should be displayed when a KR_node is presented could be:

```
no_presentation_of_creator_on_such_KR_node .(?KR_node_kind)
  := [every (?KR_node_kind  source_or_creator: ?user_ID)  description_medium:
       no (presentation_of_KR_node
            part: (a presentation_of_link_node  description_medium of:
                     (a link_node  direct_part: (a link_node_core  direct_part:
                         (a set_of_relation_type  member: pm#creator)))))];
```

Similarly built presentation directives could be used for specifying the display or not of other kinds of metadata (this includes "explanations on the used format") and more generally be used to filter out parts of query results.

5. Conclusion and Future Works

This conclusion is shorter than the introduction since this one was already an extended high-level summary. It is essentially meant to give some indications about the completion status of the presented results or on their relationships with the initially stated long-term high-level ***research goal: "supporting generic, scalable, efficient and secure ways for people to share, retrieve, understand and evaluate private or public information on a personal computer, an intranet or the Internet".*** In the introduction this expression was abbreviated as "supporting scalable general knowledge sharing" (here, "general" means "not business-to-business" and hence implies that the future usages of the knowledge cannot be predicted). For this goal, it was argued that approaches had to be *"centered around interconnected knowledge server supporting a collaboratively-built&evaluated global well-organized secure Semantic Web" (cgosSW)*. More particularly, it was argued that

- module (document, paragraph, whole ontology, ...) based approaches to knowledge sharing – as well as fully formal or fully informal approaches and non loss-less knowledge integration approaches – are unscalable (however, this does not negate the fact that building a cgosSW also requires i) techniques permitting to re-use document/modules, ii) some fully formal KBs, and iii) statistical techniques), and that
- an *"undecomposable knowledge object" centered approach for designing new resources is both necessary and, in the medium/long term, socially possible.*

For this goal, the introduction also argued that certain kinds of improvements need to be made regarding notations, formal/informal knowledge normalization methodologies, general ontologies, KM ontologies and knowledge retrieval/comparison procedures. The hypothesis behind the selected approach were given. The supporting preliminary results (notations, methodologies, techniques, tools, ontologies, ...) presented in the chapters of this document show that the selected approach is *technically possible*.

These ***preliminary results (or their underlying ideas) seem necessary*** to "support" this goal. WebKB-2, its approach and its components (in particular, FL and the MSO) proved necessary to the author of this document for representing a large amount of knowledge in an application-independent way and in a relatively intuitive manner. However, these results are clearly not sufficient for the above cited goal and all other research directions in knowledge sharing are also useful. These results are also not complete; this is why the titles of the main chapters of this document begin by "Towards". However, like this goal, these sub-goals are, in some senses, never-ending ones. The next paragraphs describe the future works that should ideally be done for each of the main results presented in this document. The order and titles used in the Table of Content are re-used. This conclusion ends with a paragraph on the exploitation of these results for information security purposes and the Internet of Things.
As noted in the introduction, another important viewpoint on the above presented results is that they can ***help answer the following recurrent research questions***: i) ***what are the criteria*** for judging the quality of knowledge representations, notations and libraries, ii) ***what kinds of techniques*** can satisfy those criteria and help *design or generate* these artifacts, and iii) ***how to semantically organize these criteria, techniques and artifacts***.

Some concepts and techniques of Knowledge Management (Section 2.1). Many top-level concept types and all relation types usually needed to represent and organize KM related concepts in a scalable way have been given, argued for and illustrated. KM related research domains should be gathered (e.g., from the lists given in KM related Calls For Papers), represented as processes and organized via specialization and subprocess relations. Many more KM tool features should also be represented and organized. The input file about Logics should be refined and other information from Wikipedia about KM related theories should be represented in the MSO. Once there is a sufficient amount of represented KM knowledge – plus adequate interfaces and checking mechanisms – to lead researchers in making additions in a scalable way (i.e., "at the right places" and in a principled way), KM related researchers should be invited to represent their research ideas or tools and hence advertise them and compare them with other ones.

Knowledge Sharing (KS): modularization, indexation, distribution, collaboration, ... (Section 2.2). This section and the introduction presented arguments about the unscalability of current approaches for *general* knowledge sharing. It also presented knowledge distribution/collaboration/valuation protocols/frameworks supporting the selected approach and gave top-level categorizations of the various kinds of techniques or approaches. These arguments, techniques and protocols should be represented in the MSO and, more generally, the categorization of techniques or approaches should be deepened. Thus, people who do not agree with some of the above claims can use WebKB-2 to relate them to other claims and thereby contribute to the building of the ontology of KM in the MSO.
Once the "structural change" of the KB of WebKB-2 is made (Subsection 2.4.2), the implementation of its protocols and techniques should be completed or refined (e.g., several avenues were listed in Section 2.2.4 for the case of a cgosSW in a P2P network). They should also be (better) tested with applications needing the collaborative update of one or several KBs by many persons. The collaborative building of the KM ontology by KM researchers is one such application. Another one will likely be the collaborative update by biology researchers of a KB about coral reef at La Reunion and nearby islands.

Following normalization rules or best practices when representing knowledge (Section 2.3). This section categorized (and argued for) various lexical, structural and semantic normalization rules for the representation of knowledge and the organization of input files. Their representation in the MSO should be completed, e.g., more examples should be given, and if possible, more rules should be added.

Knowledge comparison and knowledge-based indexation and retrieval (Section 2.4). This section proposed (and began the categorization of) various operators needed for querying, filtering and comparing knowledge. Some examples of static and generated knowledge querying/entering interfaces of WebKB-2 were given, one of them showing presentation options. The implementation of the proposed operators should be "refined" (in the context of the "structural change"), "completed" (in the case of the generation of "scalable comparison tables") or "begun" (in the case of the "organization of long lists of statements"). As noted in this section, the categorization of knowledge search/generation operators should be deepened and a categorization of presentation options should be made.

A general top-level ontology of concepts and relations (Section 3.1). This section gave an organized list of a large number of concept/relation types needed to guide and ease i) the representation and sharing of knowledge in a scalable way, and ii) the categorization of types from many ontologies. To further ease these two goals, more general ontologies would be worth integrating to the MSO, especially the FrameNet lexical database [www-FrameNet 2009] and OpenCYC [www-CYC 2009]. Another major work and an even more important one would be to update the concept type hierarchy to permit the relation type to be fully derivable from it and hence, in practice, removed.

Integrating WordNet-like resources (Section 3.2). The full content of the last version of WordNet should be integrated in the ways indicated in that section: loss-less integration, logging of the updates made to WordNet for fixing its internal ontological inconsistencies or its lexical problems, generation of intuitive identifiers, categorization of adjectives and adverbs as attributes or measures, distinctions between attributes and measures whenever possible, export of the results in various formats, etc.

Some structured discussions about notations (Section 4.1). Extending such semi-formal discussions is part of the previously cited extensions. Since some pages of Wikipedia are about hotly debated topics, the content of some of these pages should also be represented via structured discussions (this is already the case for some topics like "abortion") and references to them should be added to these Wikipedia pages for enabling people to i) see the relationships between the various claims, arguments and objections, and ii) contribute to deepen these relationships in a scalable way and without fear of their contributions being removed by other users of Wikipedia or its selection committee. This might be a way to make (parts of) the proposed knowledge sharing approach popular.

Comparison of the three main notations of WebKB with other knowledge representation languages (Section 4.2). This comparison could be extended to include more features and perhaps other languages, especially querying languages such as SPARQL. More importantly, it should be formalized, at least by representing it into a well structured ontology. This task complements and re-uses the above cited extensions to further design an ontology of KM related tasks, tools, techniques, operators, language features, language components (e.g., the various relations/quantifiers defined in KIF within this section) and presentation options.

Towards a shared LR(1) grammar for parsing FL, FCG, FE, CGLF, CGIF and KIF (Section 4.3). This work should be completed for this grammar to include all of CGIF and KIF. The current parsers for all these languages in WebKB-2 should be merged. Their implementation should take into account values in the ontology of knowledge presentation (Section 4.5) in order to allow users to change the syntactic sugar of their languages by changing these values. Then, as noted in Paragraph 2.1.1.6, the parser for FE can be independently extended towards natural English. The parser for RDF/XML currently appears too different to be worth being integrated into this unique parser.

Summary of the future data model of WebKB-2 (Section 4.4). The current code of WebKB-2 should be adapted to fit this new model. This will be an occasion for making this code and its API even more independent from the data model. The use of different back-ends, especially back-ends accessible via OKBC, has been prepared. Once this implementation is made, many other ones can be completed or begun. When doing so, another kind of genericity will be tackled by taking into account presentation options selected by the user in the ontology cited in the next paragraph.

A general ontology for notations and knowledge presentation (Section 4.5). A part of this ontology has very recently been developed: the part related to meta-model elements and their basic presentation. However, several parts have not yet been represented: those related to parameters (especially presentation related parameters) for the collaboration protocols, valuation procedures, querying/filtering/comparison/merging operators and structuring of long lists of statements.

Security and semantics for discovery services in RFID like information systems and, more generally, the Internet of Things (IoT). This paragraph is a summary of [Martin & Roudier, 2009] (please see this article for details and references).
Devices of the IoT have a unique identifier (ID) and can communicate information (e.g., their location) to databases or other devices. The ID permits to refer to the object (or "thing") to which the device is associated and hence permits to store and retrieve information about this object in databases. Discovery services are often seen as the "next big step" in the exploitation of IoT devices – RFID tags in particular – but are yet only envisaged as i) providing a list of pointers to databases containing information about an object, and ii) optionally forwarding queries to each of these databases. Thus, it is not yet envisaged to design "semantic discovery services" answering queries on "kinds of objects" and organizing or merging information from multiple databases (in a logical/semantic valid way). Current IoT infrastructures do not support any genuine sharing of "knowledge" (they only support the sharing of certain predefined kinds of data). Yet, doing so and, more precisely, using the precision oriented techniques (or "undecomposable knowledge object" based techniques) proposed in this document to support a cgosSW for an IoT in a particular domain activity, e.g., supply-chain management in France or the manufacturing of aeronautics related components in Europe,

- is less difficult than in the general case because of the limited variety of actors, kinds of objects, kinds of queries and kinds of collaboration, as well as the possibility for the cgosSW to be managed within a P2P network; (however, the limited variety of the kinds of objects does not mean that there cannot be hundreds of thousands of different types of objects);
- is even more relevant than in the general case because of the nature of the involved objects and queries;
- would permit tremendous improvements in information sharing, retrieval and exploitation possibilities (see [Martin & Roudier, 2009] for details on such possibilities and on their economic, social and regulatory context).

Such improvements in information retrieval require improvements in the *possibilities or flexibility* given to each actor (e.g., each manufacturer, retailer and consumer of a particular product or kind of products) to define complex access-control rules on any piece of information she provided and to associate these rules to these pieces of information so that the rules are respected wherever the pieces of information are stored. The need for this is for example stressed by [Weitzner, 2007]. However, nowadays, almost no flexibility is proposed: only a few kinds of simple predefined rules can be associated to *groups of* objects (of generally predefined kinds) and only within particular systems. This is because currently

- there is no general shared ontology of security features that a user could choose from and compose to design her rules, and that security tool authors could add to and hence collaboratively design (such an ontology need not be unique, there may be several competing ones; the MSO includes some top-level categories for such an ontology, as illustrated by Table 2.1.2.3 and Table 2.3.4.1),
- no security system permits a rule to use identifiers from a general collaboratively updatable ontology (such as the MSO of WebKB-2) for referring to the particular object or kinds of objects that the rule applies to – the identifiers for the objects of kinds of objects may be entered by the users themselves or the manufacturers of these objects.

Several security systems already take into account ontologies defined by the users but not yet in a way that would permit the last two points. On the other hand, it does not seem that these two points require extensions to the mechanisms that were presented to support the collaborative update of an ontology.
To conclude, a cgosSW is not solely useful for actually achieving a genuinely Semantic Web but also for the IoT and information security.

6. References

1. AFIA (2002). Améliorer ou supprimer les referees ? Bulletin de l'AFIA (Association Française pour l'Intelligence Artificielle) No 48, pp. 8-11.
2. Allen J. (1999). Different Kinds of Controlled Languages. TC-Forum magazine, 1(99):4-5, 1999.
3. Alkhateeb F., Baget J.-F. & Euzenat J. (2009). *Extending SPARQL with regular expression patterns (for querying RDF).* Journal of web semantics 7(2), pp. 57-73, 2009.
4. Baget J.-F., Corby O., Dieng-Kuntz R., Faron-Zucker C., Gandon F., Giboin A., Gutierrez A., Leclère M., Mugnier M.-L. & Thomopoulos R. (2008). *Griwes: Generic Model and Preliminary Specifications for a Graph-Based Knowledge Representation Toolkit.* Proceedings of ICCS 2008 (16th International Conference on Conceptual Structures), July 2008, Toulouse.
5. Baget J-F, Chein M., Croitoru M., Fortin J., Genest D., Gutierrez A, Leclère M., Mugnier M.-L. & Salvat E. (2009). RDF to Conceptual Graphs Translations. Proceedings of the Third Conceptual Structures Tool Interoperability Workshop, collocated with the 17h International Conference on Conceptual Structures (ICCS 2009), 25 July 2009. http://www-sop.inria.fr/acacia/project/griwes/wakka.php?wiki=ColorGriwes
6. Barwise J. & Perry J. (1983). *Situation and Attitudes.* MIT Press, Cambridge, MA.
7. Barwise J., Gawron J.M., Plotkin G. & Tutiya S. (1991). *Situation Theory and its Applications*. CSLI, Stanford, CA.
8. Benjamins V. R., Fensel D., Gomez-Perez A., Decker S., Erdmann M., Motta E. & Musen M. (1998). *Knowledge Annotation Initiative of the Knowledge Acquisition Community: (KA)2.* Proceedings of KAW 1998, 11th Knowledge Acquisition for Knowledge Based System Workshop, Banff, Canada, April 18-23, 1998. http://ksi.cpsc.ucalgary.ca/KAW/KAW98/benjamins2/
   The ontology is at http://ontobroker.semanticweb.org/ontologies/ka2-onto-2000-11-07.flo
9. Berners-Lee T. (1999) *The Semantic Toolbox: Building Semantics on top of XML-RDF.* W3C Note, 24 May 1999. http://www.w3.org/DesignIssues/Toolbox.html; see also the "Semantic Web Road map" at http://www.w3.org/DesignIssues/Semantic.html
10. Berners-Lee T., Hendler J. & Lassila O. (2001). *The Semantic Web.* Scientific American, May 17th, 2001.
11. Bizer C., Heath T. & Berners-Lee T. (2010). *Linked data – the story so far*. International Journal on Semantic Web and Information Systems, 5(3), pp. 1-22.
12. Brandom R. (2000). *Articulating Reasons: An Introduction to Inferentialism.* Harvard University Press, Cambridge, MA, 2000.
13. Breuker J. (1994). *A Suite of Problem Types*. In [Breuker & van de Velde, 1994], pp. 57-88. Partially available online via Google books.
14. Breuker J. & van de Velde W. (1994). *CommonKADS Library for Expertise Modelling: Reusable Problem Solving Components.* IOS Press, Amsterdam 1994. See also http://arti.vub.ac.be/previous_projects/kads/D1.3.ps ("The CommonKADS Library", May 4, 1992).
15. Buckingham-Shum S., Motta E. & Domingue J. (1999). *Representing Scholarly Claims in Internet Digital Libraries: A Knowledge Modelling Approach.* Proceedings of ECDL 1999 (pp. 423-442), 3rd European Conference on Research and Advanced Technology for Digital Libraries, Paris, France, September 1999. http://citeseer.ist.psu.edu/shum99representing.html
16. Buffa M., Gandon F., Erétéo G., Sander P. & Faron C. (2008). *SweetWiki: A Semantic Wiki.* Special Issue of the Journal of Web Semantics on Semantic Web and Web 2.0, Volume 6, Issue 1, pp. 84-97, February 2008.
17. Caraciolo C., Euzenat J., Hollink L., Ichise R., Isaac A., Malaisé V., Meilicke C., Pane J., Shvaiko P., Stuckenschmidt H., Svab O. & Svatek V. (2008). *Results of the Ontology Alignment Evaluation Initiative 2008.* Proceedings of the 3rd ISWC workshop on ontology matching (eds: Shvaiko P., Euzenat J., Giunchiglia F. & Stuckenschmidt H.; pp. 73-119), Karlsruhe, Germany, 2008.
18. Casanovas P., Casellas N., Tempich C., Vrandecic D., Benjamins R. (2007). *Opjk and diligent: ontology modeling in a distributed environment*. Artificial Intelligence Law, 15-(2), pp. 171-186.

19. Chaudhri V., Rodriguez A., Thoméré J., Mishra S., Gil Y., Hayes P. & Reichherzer T. (2001). *Knowledge entry as the graphical assembly of components.* Proceedings of K-Cap 2001 (pp. 22-29), 1st international Conference on Knowledge Capture British Columbia, Canada, October 22-23, 2001.
20. Chein M. & Mugnier M.-L. (1997). *Positive Nested Conceptual Graphs.* Proceedings of ICCS 1997 (Springer Verlag, LNAI 1257, pp. 95-109), Seattle, USA, August 4-8, 1997.
21. Clark P. (2008). *Some Ongoing KBS/Ontology Projects and Groups*. http://www.cs.utexas.edu/users/mfkb/related.html
22. Colomb R. (1997). *Impact of Semantic Heterogeneity on Federating Databases.* The Computer Journal, 40(5), pp. 235-244, 1997.
23. Colomb R., Chang D., Kendall E., Boger M., Emery P., Raymond K., Martin Ph., Ye Y., Dutra M., Frankel D., Hart L., Hayes P., McGuinness D. & Garshol L.M. (2005). *Ontology Definition Metamodel.* Third Revised Submission to OMG/RFP_ad/2003-03-40, August 22, 2005.
24. Conen W., Klapsing R. & Köppen E. (2001). *RDF M&S revisited: From Reification to Nesting, from Containers to Lists, from Dialect to pure XML* Proceedings of SWWS 2001 (pp. 2-7), Semantic Web Working Symposium.
25. Corby O., Dieng-Kuntz R. & Faron-Zucker C. (2004). *Querying the Semantic Web with the Corese Search Engine.* ECAI, IOS Press pp. 705-709, 2004. See also Sewese [www-Sewese, 2009], the KB server based on Corese.
26. Corby O., Dieng-Kuntz R., Faron-Zucker C. & Gandon F. (2007). *Searching the Semantic Web: Approximate Query Processing based on Ontologies.* IEEE Intelligent Systems Journal, Vol. 21, No. 1, January/February 2006.
27. Corby O., Kefi-Khelif L., Cherfi H., Gandon F. & Khelif K. (2009). *Querying the Semantic Web of Data using SPARQL, RDF and XML.* INRIA Research Report 6847, February 2009.
28. Corcho O. (2005). *A layered declarative approach to ontology translation with knowledge preservation.* Frontiers in Artificial Intelligence and its Applications. Dissertations in Artificial Intelligence. IOS Press. January 2005.
29. Correia J.H. & Pöschel R. (2006). *The Teridentity and Peircean Algebraic Logic*. In "Conceptual Structures: Inspiration and Application", LNCS 4068/2006, Springer Berlin/Heidelberg, pp. 229-246, August 22, 2006. http://www.springerlink.com/content/42v7r03v4x08l831/
30. Crawford J. M. & Kuipers B. J. (1991). *Algernon – a tractable system for knowledge representation*. SIGART Bulletin 2(3), June 1991, pp. 35-44.
31. Decker S., ErdmannM., Fensel D. & Studer R. (1999) *Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information.* Semantic Issues in Multimedia Systems (eds: R. Meersman et al.), Kluwer Academic Publisher, pp. 351-369, Boston, 1999.
32. Dehainsala H., Pierra G. & Bellatreche L. (2007) *OntoDB: An Ontology-Based Database for Data Intensive Applications*. Springer, LNCS 4443/2008, pp. 497-508, August 2, 2007.
33. Denny M. (2004). *Ontology Tools Survey, Revisited*. http://www.xml.com/pub/a/2004/07/14/onto.html, July 14, 2004.
34. Djedidi R. & Aufaur A. (2010). Define Hybrid Class Resolving Disjointness due to Subsumption. http://ontologydesignpatterns.org/wiki/Submissions:Define_Hybrid_Class_Resolving_Disjointness_due_to_Subsumption
35. Devedzic V. (2004). *Education and the Semantic Web.* International Journal of Artificial Intelligence in Education, 14, pp. 39-65.
36. Downes S. (2001). *Learning Objects: Resources For Distance Education Worldwide.* International Review of Research in Open and Distance Learning, Vol. 2, No 1., 1 October, 2001.
37. Dromey G.R. (2006). *Scaleable Formalization of Imperfect Knowledge.* Proceedings of AWCVS-2006, 1st Asian Working Conference on Verified Software, 29-31 October 2006, Macao SAR, China. http://www.iist.unu.edu/www/workshop/AWCVS2006/

38. Ellis G. & Levinson R. (1992). *The Birth of PEIRCE: A Conceptual Graphs Workbench.* Proceedings of of the First International Workshop on "PEIRCE: A Conceptual Graphs Workbench", LNAI 754 ("Conceptual Structures: Theory and Implementation"), 1992.
39. Erétéo G., Gandon F., Corby O. & Buffa M. (2009). *Semantic Social Network Analysis.* Proceedings of WebSci 2009 (Web Science), Athens, Greece, March 2009.
40. Euzenat J. (1996). *Corporate memory through cooperative creation of knowledge bases and hyper-documents*. Proceedings of 10th KAW (36, pp.1-18), Banff, Canada, November 1996.
http://ksi.cpsc.ucalgary.ca/KAW/KAW96/euzenat/euzenat96b.html
See also http://www.inrialpes.fr/exmo/papers/exmo1995.html#Euzenat1995a
41. Euzenat J. (2001) *Construction collaborative de bases de connaissance et de documents pour la capitalisation.* Ingénierie et capitalisation des connaissances, Hermès Science, 2001, pp. 25-48.
42. Euzenat J., Stuckenschmidt H. & Yatskevich M. (2005). *Introduction to the Ontology Alignment Evaluation 2005.* Proceedings of K-Cap 2005 (pp. 61-71), workshop on Integrating ontology, Banff, Canada, 2005.
43. Euzenat J. (2005). *Alignment infrastructure for ontology mediation and other applications.* Proceedings of ICSOC 2005 (pp. 81-95), 1st ICSOC international workshop on Mediation in semantic web services, Amsterdam, Netherlands.
44. Euzenat J. & Shvaiko P. (2007) *Ontology Matching*. Springer-Verlag, Berlin Heidelberg, 2007, 341 pages. http://book.ontologymatching.org/
45. Euzenat J., Allocca C, David J., d'Aquin M., Le Duc C. & Svab-Zamazal O. (2009). *Ontology distances for contextualisation.*, IST NeOn IP, deliverable 3.3.4, 50 pages, 2009.
46. Fielding R.T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Ph.D. Thesis, University of California, Irvine, Irvine, California, 2000.
47. Flater D., Martin Ph. & Crane M. (2009). *Rendering UML Activity Diagrams as Human-Readable Text.* Proceedings of IKE 2009 (pp. 207-213), international conference on Information and Knowledge Engineering, Las Vegas, USA, July 13-16, 2009.
http://www.world-academy-of-science.org/worldcomp09/ws/conferences/ike09
Also published as NISTIR report 7469, National Institute of Standards & Technology, Gaithersburg, MD, 2007.
48. Friedland N.S, Allen P., Mathews G., Witbrock M., Baxter D., Curtis J., Shepard B., Miraglia P., Angele J., Staab S., Moench E., Opperman H., Wenke D., Israel D., Chaudhri V., Porter B., Barker K., Fan J., Chaw S.Y., Yeh P., Tecuci D., and Clark P. (2004). *Project Halo: Towards a Digital Aristotle.* AI Magazine, 25(4), pp. 29-48. http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/1783
See also http://www.projecthalo.com/
49. Fuchs N.E., Schwertel U. & Torge S. (1999). *Controlled Natural Language Can Replace First-Order Logic.* Proceedings of ASE 1999, 14th IEEE International Conference on Automated Software Engineering, Cocoa Beach, Florida, 1999. http://attempto.ifi.uzh.ch/site/
50. Gandon F., Lo M. & Niang C. (2008). *Un modèle d'index pour la résolution distribuée de requête sur un nombre restreint de bases d'annotations RDF.* Proceedings of IC 2008, 19èmes Journées Francophones d'Ingénierie des Connaissances, Loria, Nancy, France, 18-20 June 2008.
51. Gangemi A., Navigli R. & Velardi P. (2003). *The OntoWordNet Project: extension and axiomatization of conceptual relations in WordNet.* Proceedings of ODBASE 2003 (international conference on Ontologies, Databases and Applications of SEmantics), Catania, (Italy), November 3-7, 2003.
52. Gangemi A., Guarino N. & Oltramari A. (2002). *Restructuring Wordnet's Top-Level.* AI Magazine, 40(5):235-244, fall 2002.
53. Genest D. & Salvat E. (1998). *A Platform Allowing Typed Nested Graphs: How CoGITo Became CoGITaNT.* Proceedings of ICCS 1998 (Springer LNAI 1453, pp. 154-161), Montpellier, France, August 1998.
54. Genesereth M.R. (1998). *Knowledge Interchange Format.* Draft proposed American National Standard (dpANS), NCITS.T2/98-004. http://logic.stanford.edu/kif/dpans.html

55. Gerbé O., Mineau G. & Keller R. (2001). *Conceptual Graphs and Metamodeling.* Proceedings of ICCS 2001 (LNAI 2120, pp. 245-259), Stanford, CA, USA, July/August 2001.
56. Gerbé O., Mineau G. & Keller R. (2007). *Un métamodèle des graphes conceptuels.* Revue d'intelligence artificielle, Vol. 21, No 2/2007, pp. 255-284, May 2007.
57. Gouardères G., Saber M., Nkambou R., & Yatchou R. (2005). *The Grid-E-Card: Architecture to Share Collective Intelligence on the Grid.* Applied Artificial Intelligence, Vol. 19, No. 9-10, pp. 1043-1073.
58. Groh B. & Eklund P. (1999). *Algorithms for Creating Relational Power Context Families from Conceptual Graphs.* Springer LNCS 1640/1999, Conceptual Structures: Standards and Practices, 23 July 2007.
59. Guarino N., Masolo C. & Vetere G. (1999). *Ontoseek: Content-based Access to the Web.* IEEE Intelligent Systems, Vol. 14, No. 3, 1999, pp. 70-80.
60. Guarino N. & Welty C. (2002). *Evaluating Ontological Decisions with OntoClean.* Communications of the ACM, 45(2): pp. 61-65
61. Haemmerlé O. & Guinaldo O. (1999). *CoGITo v3.3 : plate-forme de développement d'applications sur les graphes conceptuels.* Technique et Science Informatique, 18 (9), pp. 933-965, November 1999.
62. Haemmerlé O., Buche P & Thomopoulos R. (2007). *The MIEL system: uniform interrogation of structured and weakly structured imprecise data.* Journal of Intelligent Information Systems, Springer, Volume 29, Number 3, December 2007, pp. 279-304.
63. Hayes P. (2005). *Translating Semantic Web languages into Common Logic.* http://www.ihmc.us/users/phayes/CL/SW2SCL.html
64. Hayes P. & Welty C. (2006). *IKL Specification Document.* www.ihmc.us/users/phayes/IKL/SPEC/SPEC.html See also http://www.ihmc.us/users/phayes/ikl/guide/guide.html
65. Hayes P. & Menzel C. (2006). *Defining N-ary Relations on the Semantic Web.* W3C Working Group Note 12 April 2006. http://www.w3.org/TR/swbp-n-aryRelations/
66. Hendler J. (2001). *Agents and the Semantic Web.* IEEE Intelligent Systems, 16(2):30-37, 2001.
67. Heijst G., Schreiber A.T. & Wielinga B.J. (1996). *Using Explicit Ontologies in KBS Development.* International Journal of Human-Computer Studies/Knowledge Acquisition, Fall 1996. http://hcs.science.uva.nl/usr/gertjan/postscript/ijhcs-HSW.ps.gz
68. Hillis W.D. (2004). *"Aristotle" (The Knowledge Web).* Edge Foundation, Inc., No 138, May 6, 2004. http://www.edge.org/3rd_culture/hillis04/hillis04_index.html
69. Hodgins W. (2006). *Out of the past and into the future: Standards for technology enhanced learning.* Handbook on Quality and Standardisation in E-Learning (Eds; U. Ehlers and J. Pawlowski), Springer Berlin Heidelberg, pp. 309-327.
70. Holland J.H. (1998). *Emergence: From Chaos to Order.* Oxford University Press, 272 pages.
71. IEEE-SUO-MSO (2004). *MSO Ballot Results.* http://suo.ieee.org/email/msg12552.html 12 May 2004
72. ISO/IEC 24707 (2007). *Information technology – Common Logic (CL): a framework for a family of logic-based languages*. ISO/IEC 24707:2007(E), JTC1/SC32. http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html
73. Kalfoglou Y., Alani H., Schorlemmer M. & Walton C. (2004). *On the Emergent Semantic Web & Overlooked Issues.* Proceedings of ISWC 2004 (LNCS 3298, pp. 576-591), 3rd International Semantic Web Conference, LNCS 3298, Hiroshima, Japan, November 2004.
74. Keeler M. & Majumdar A. (2008). *Revelator's Complex Adaptive Reasoning Methodology for Resource Infrastructure Evolution.* Proceedings of ICCS 2008 (pp. 88-103), 16th International Conference on Conceptual Structures, Toulouse, France, July 7-11, 2008.
75. Kerdiles G. (2001). *Saying it with Pictures: A Logical Landscape of Conceptual Graphs.* PhD thesis, University of Amsterdam and University of Montpellier II, November 2001.
76. Knight K. & Luk S. (1994). *Building a Large-Scale Knowledge Base for Machine Translation.* Proceedings of AAAI 1994, 773-778, Seattle, USA, July 1994.

77. Knizhnik K. (2007). *FastDB: a main-memory database object-relational database system.* Available at http://www.garret.ru/knizhnik/fastdb.htm
78. Kwok C., Etzioni O. & Weld D. (2001). *Scaling Question Answering to the Web.* ACM Transactions on Information Systems, 19(3), July 2001.
79. Le Duc C. & Le Thanh N. (2003). *Combining Revision Production Rules and Description Logics.* Proceedings of KES 2003 (Springer, pp. 89-98), 7th International Conference on Knowledge-Based Intelligent Information & Engineering Systems, University of Oxford, United Kingdom,3-5 September 2003.
80. Le Pham T., Le Thanh N. & Sander P. (2008). *Decomposition-based reasoning for large knowledge bases in description logics.* Integrated Computer-Aided Engineering 15(1), pp. 53-70.
81. Lausen H., Ding Y., Stollberg M., Fensel D., Lara R. & Han S. (2005). *Semantic web portals: state-of-the-art survey.* Journal of Knowledge Management, Vol. 9, No. 5, pp. 40-49.
82. Leclère M. & Mugnier M.-L. (2008). *An Algorithmic Study of Deduction in Simple Conceptual Graphs with Classical Negation.* Proceedings of ICCS 2008 (LNAI 5113, pp. 119-132), Toulouse, France, 2008.
83. Lee J., Park J., Park M., Chung C. & Min J. (2010). *An intelligent query processing for distributed ontologies*. Systems and Software, 83(1), January 2010, pp. 85-95.
84. Levinson R. & Ellis G. (1992). *Multilevel hierarchical retrieval*. Knowledge-Based Systems, 5(3), pp. 233-244.
85. Lewen H., Supekar K.S., Noy N.F., & Musen M.A. (2006). *Topic-Specific Trust and Open Rating Systems: An Approach for Ontology Evaluation.* Proceedings of EON 2006, Workshop on Evaluation of Ontologies for the Web, at WWW 2006, 15th International World Wide Web Conference, Edinburgh, UK 2006.
86. Liu W. (2005). *Design of Text-based Questions from the Study of Typology of Questions.* Sino-US English Teaching, USA, May 2005, Volume 2, No.5 (Serial No.17)
87. Lowe D. (1985). *Co-operative Structuring of Information: The Representation of reasoning and debate.* International Journal of Man-Machine Studies. Volume 23, Number 2, pp. 97-111, August 1985.
88. Makni B., Khelif K., Dieng-Kuntz R. & Cherfi H. (2008). *Building a Semantic Portal from the Discussion Forum of a Community of Practice.* Proceedings of I-Semantics 2008, International Conference on Semantic Systems, Graz, Austria.
89. Martin Ph. (1996). *Exploitation de graphes conceptuels et de documents structurés et hypertextes pour l'acquisition de connaissances et la recherche d'informations.* Ph.D. thesis (378 pages), University of Nice - Sophia Antipolis, France, October 14, 1996.
90. Martin Ph. & Eklund P. (1999a). *WebKB and the Sisyphus-I problem.* Proceedings of ICCS 1999 (Springer, LNAI 1640, pp. 315-333), 7th International Conference on Conceptual Structures, Blacksburg, Virginia, USA, July 12-15, 1999.
91. Martin Ph. & Eklund P. (2000). *Knowledge Indexation and Retrieval and the Word Wide Web.* IEEE Intelligent Systems, special issue "Knowledge Management and Knowledge Distribution over the Internet", pp. 18-25, May/June 2000.
92. Martin Ph. (2002). *Knowledge representation in CGLF, CGIF, KIF, Frame-CG and Formalized-English.* Proceedings of ICCS 2002, 10th International Conference on Conceptual Structures (Springer Verlag, LNAI 2393, pp. 77-91), Borovets, Bulgaria, July 15-19, 2002.
93. Martin Ph. (2002a). *How WebKB could contribute to PORT.* Proceedings of PORT 2002, 2nd PORT workshop, first day of ICCS 2002.
94. Martin Ph. (2003a). *Knowledge Representation, Sharing and Retrieval on the Web.* Chapter of a book titled "Web Intelligence", (Eds.: N. Zhong, J. Liu, Y. Yao; Springer-Verlag, pp. 263-297), January 2003.
95. Martin Ph. (2003b). *Correction and Extension of WordNet 1.7.* Proceedings of ICCS 2003 (Springer Verlag, LNAI 2746, pp. 160-173), Dresden, Germany, July 2003. See also [Martin, 2003c].
96. Martin Ph. (2003c). *Integration of WordNet 1.7 in WebKB-2.* http://www.webkb.org/doc/wn/
97. Martin Ph. (2003d). *KRM (the "FS meta-model" in MOF).* http://www.webkb.org/doc/model/KRM.html

98. Martin Ph. (2004). *Comparison of six CG tools according to 160 criteria.* http://www.webkb.org/kb/it/fs/CG_tools.html
99. Martin Ph. (2005). *Services on the Sunshine Coast.* http://www.webkb.org/SC/
100. Martin Ph., Blumenstein M. & Deer P. (2005). *Toward cooperatively-built knowledge repositories*. Proceedings of ICCS 2005, 13th International Conference on Conceptual Structures, (Springer Verlag, LNAI 3596, pp. 411-424), Kassel, Germany, July 18-22, 2005.
101. Martin Ph. (2006). *Documents related to my Griffith E-Learning Fellowship for Semester 2, 2006.* http://www.webkb.org/doc/papers/GEL06/
102. Martin Ph. (2006b). *Structured discussions & Semantic classification of some resources.* http://www.webkb.org/kb/it/
All MSO input files are accessible from http://www.webkb.org/kb/
103. Martin Ph. (2006c). *The WebKB languages.* http://www.webkb.org/doc/languages/
104. Martin Ph. (2007). *Making node relations explicit.*
http://www.textop.org/wiki/index.php?title=Talk:The_Outline#Making_node_relations_explicit
105. Martin Ph. (2007b). *Structured discussion on XML for knowledge representation.* http://www.webkb.org/kb/it/o_knowledge/p_kPresentation/sd_XMLforKR.html
106. Martin Ph. (2008). *Semantic Classification of Word Processors.* http://www.webkb.org/kb/it/o_software/d_wordProcessors.html
107. Martin Ph. & Eboueya M. (2008). *For the ultimate accessibility and re-usability*. Chapter XXIX (14 pages) of the Handbook of Research on Learning Design and Learning Objects: Issues, Applications and Technologies, IGI Global, ISBN 978-1-59904-861-1, July 14, 2008.
108. Martin Ph. (2009). *Managing Knowledge to Enhance Learning.* International Journal of Knowledge Management & E-Learning (ISSN 2073-7904), Vol.1, No.2, 2009, pp. 103-119. http://www.kmel-journal.org/ojs/index.php/online-publication/article/view/12/19
109. Martin Ph. & Roudier Y. (2009). *Security and Semantics for Discovery Services in RFID like information systems.* http://www.webkb.org/doc/papers/wmnc09/
110. Mcdermott D. & Dou D. (2002). *Representing Disjunction and Quantifiers in RDF.* Proceedings of ISWC 2002 (LNCS 2342/2002, pp. 250-263), International Semantic Web Conference.
111. Mihalcea R. & Moldovan D. (2000). *Semantic Indexing using WordNet senses.* Proceedings of ACL Workshop on IR & NLP, Hong Kong, October, 2000.
112. Miller G. (1995). *Wordnet: a Lexical Database for English.* Communications of the ACM, 11:39-41, 1995. http://wordnet.princeton.edu/
113. Mugnier M.L. & Chein M. (1992). Polynomial algorithms for projection and matching. LNAI 754, pp 49-58.
114. Nanard J., Nanard M., Massotte A., Djemaa A., Joubert A., Betaille H. & Chauché J. (1993). *Integrating Knowledge-based Hypertext and Database for Task-oriented Access to Documents.* Proceedings of DEXA 1993 (Springer Verlag, LNCS Vol. 720, pp. 721-732), Prague, 1993.
115. Newman S. & Marshall C. (1992). *Pushing Toulmin Too Far: Learning From an Argument Representation Scheme.* Technical Report SSL-92-45, Xerox Palo Alto Research Center, Palo Alto, CA, 1992.
116. Nguyen T.D.T. & Le Thanh N. (2007). *Identification constraints in SHOIN(D).* Proceedings of RCIS 2007, IEEE Conference on Research Challenges in Information Science, Ouarzazate, Morocco, 23-26 April 2007.
117. Noy N.F. & Deborah L. (2000). *Ontology Development 101: A Guide to Creating Your First Ontology.* http://ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html
118. Noy N.F., Chugh A., Liu W. & Musen M. A. (2006). *A framework for ontology evolution in collaborative environments.* Proceedings of ISWC 2006, 5th International Semantic Web Conference, Athens, GA.
119. Noy N.F. & Tudorache T. (2008). *Collaborative ontology development on the (semantic) web*. Proceedings of SWKE 2008, AAAI Spring Symposium on Semantic Web and Knowledge Engineering, March 2008.

120. Novak J.D. (2004). *Reflections on a Half Century of Thinking in Science Education and Research: Implications from a Twelve-year Longitudinal Study of Children's Learning.* Canadian Journal of Science, Mathematics, and Technology Education, 4(1): 23-41.

121. Page K., Michaelides D., Buckingham-Shum S., Chen-Burger Y., Dalton J., De Roure, D. Eisenstadt M., Potter S., Shadbolt N., Tate A., Bachler M. & Komzak J. (2005). *Collaboration in the Semantic Grid: a Basis for e-Learning.* Journal of Applied Artificial Intelligence, 19(9-10), pp. 881-904.

122. Palma R., Haase P., Wang Y. & D'Aquin M. (2008). *Propagation models and strategies*. Deliverable D1.3.1 of NeOn – Lifecycle Support for Networked Ontologies; NEON EU-IST-2005-027595, January 2008.

123. Patel-Schneider P.F. (2005). *A Revised Architecture for Semantic Web Reasoning.* Proceedings of PPSWR 2005 (LNCS 3703, pp. 32-36), 3rd Workshop on Principles and Practice of Semantic Web Reasoning, Dagstuhl, Germany, 11th-16th September 2005.

124. Pietriga E., Bizer C., Karger D., Lee R. (2006). *Fresnel: A Browser-Independent Presentation Vocabulary for RDF*. Proceedings of ISWC 2006 (LNCS 4273, pp. 158-171), 5th International Semantic Web Conference, November 2006, Athens, GA, USA.

125. Pool J. (2006). *Can Controlled Languages Scale to the Web?*. CLAW 2006 (5th International Workshop on Controlled Language Applications) at AMTA 2006.
See http://www.webkb.org/doc/languages/PoolSentencesInFE.html for the FE translations that permitted Pool to compare FE w.r.t. other controlled languages.

126. Puder A. & Romer K. (1997) *Generic Trading Service in Telecommunication Platforms.* Proceedings of ICCS 1997, LNAI 1257, pp. 551-565, August 1997. http://www.vsb.informatik.uni-frankfurt.de/projects/aitrader/

127. [p2p-Pitoura 2006] Pitoura E. *Resource Discovery: State of the Art Survey and Algorithmic Solutions.* IST FP6 Aeolus Deliverable D2.1.1., September 2006. http://aeolus.ceid.upatras.gr

128. [p2p-semanticDrivenHashing 2004] Sangpachatanaruk C. & Znati T. *Semantic Driven Hashing (SDH): An Ontology-Based Search Scheme for the Semantic Aware Network (SA Net)* Proceedings of Fourth International Conference on Peer-to-Peer, pp. 270-271, 2004.

129. [p2p-semanticOverlay 2005] Crespo A., Garcia-Molina H. *Semantic Overlay Networks for P2P Systems.* Proceedings of Agents and Peer-to-Peer Computing: 1-13 LNCS 3601 Springer 2005.

130. [p2p-semanticRouting 2004] Loser A., Schubert K., Zimmer F. *Taxonomy-based routing overlays in P2P networks.* Proceedings of IDEAS 2004 (pp. 407-412), Database Engineering and Applications Symposium, 7-9 July 2004.

131. [p2p-semanticQuerying 2006] Rousset M.C., Adjiman P., Chatalic P., Goasdoué F. & Simon L. *SomeWhere: A Scalable Peer-to-Peer Infrastructure for Querying Distributed Ontologies.* Proceedings of ODBASE 2006, LNCS 4275, pp. 698-703, October 29 – November 3, 2006.

132. Quint V. & Vatton I. (1992). *Combining Hypertext and Structured Documents in Grif.* Proceedings of ECHT'92 (ed.: D. Lucarella, ACM Press, pp. 23-32), Milan, December 1992.

133. Raymond K., Martin Ph. & Colomb B. (2003). *Ontology Definition Metamodel.* OMG document ad/03-08-01 (DSTC Initial Submission to the Ontology Definition Metamodel RFP of the Object Management Group), August 18, 2003. The four proposals received by the OMG have been merged into [Colomb et al., 2005].

134. Rousset M-C. (2004). *Small Can Be Beautiful in the Semantic Web.* Proceedings of ISWC 2004, International Semantic Web Conference, pp. 6-16.

135. Rogers J.E. & Rector A.L.(2000). *GALEN's Model of Parts and Wholes: Experience and Comparisons.* Annual Fall Symposium of American Medical Informatics Association, Los Angeles CA Hanley & Belfus Inc Philadelphia PA, pp. 714-718. See also:

A-M Rassinoux, RH Baud, C Lovis, JC Wagner, J-R Scherrer (1998). *Tuning up Conceptual Graph Representation for Multilingual Natural Language Processing in Medicine.* Proceedings of ICCS 1998 (pp. 390-397), Montpellier, France, August 10-12, 1998.

136. Rector A.L. & Rogers J.E. (2006). *Ontological and Practical Issues in Using a Description Logic to Represent Medical Concept Systems: Experience from GALEN.* Reasoning Web 2006, pp. 197-231.

137. Rodriguez M., Steinbock D., Watkins J., Gershenson C., Bollen J., Grey V. & Degraf B. (2007). *Smartocracy: Social Networks for Collective Decision Making.* Proceedings of HICSS 2007 (pp. 90-100, System Sciences), 40th Annual Hawaii International Conference on In System Sciences, 2007.
138. Salvat E. & Mugnier M.-L. (1996). *Sound and Complete Forward and Backward Chainings of Graph Rules.* Proceedings of ICCS 1996 (LNAI 1115, pp. 248-262), Sydney, Australia, August 1996.
139. Schroder E. (1899). *Pasigraphy. Its present state and the pasigraphic movement in Italy.* The Monist, 9, pp. 44-62, 1899. http://www.jfsowa.com/logic/es99.pdf
140. Shadbolt N. R. & Burton M. (1995) *Knowledge Elicitation: A Systematic Approach.* Evaluation of Human Work: A Practical Ergonomics Methodology, Chapter 14, pp. 406-440.
141. Shadbolt N., Berners-Lee T. & Hall W. *The Semantic Web Revisited.* IEEE Intelligent Systems, 21(3) pp. 96-101, May/June 2006.
142. Sanger L.M. (2006). *The Future of Free Information.* Digital Universe Electronic Journal, 2006-01. Retrieved July 2, 2007, from http://www.dufoundation.org/downloads/Article_2006_01.pdf
143. Seaborne A., Manjunath G., Bizer C., Breslin J., Das S., Davis I., Harris S., Idehen K., Corby O., Kjernsmo K. & Nowack B. (2008). *SPARQL/Update A language for updating RDF graphs.* W3C Member Submission, July 2008. http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/
144. Siorpaes K. & Hepp M. (2007). *OntoGame: Towards Overcoming the Incentive Bottleneck in Ontology Building*. Proceedings of SWWS 2007 (LNCS 4806, pp. 1222-1232), 3rd International IFIP Workshop On Semantic Web & Web Semantics, November 29-30, 2007.
145. Shipman F.M., & Marshall C.C. (1999). *Formality considered harmful: experiences, emerging themes, and directions on the use of formal representations in interactive systems.* Computer Supported Cooperative Work, 8, pp. 333-352.
146. Schuler W. & Smith J.B. (1990). *Author's Argumentation Assistant (AAA): A Hypertext-Based Authoring Tool for Argumentative Texts.* Proceedings of ECHT 1990 (Cambridge University Press, pp. 137-151), INRIA, France, Nov. 1990.
147. Siorpaes K. & Hepp M. (2007). *OntoGame: Towards Overcoming the Incentive Bottleneck in Ontology Building.* Proceedings of SWWS 2007 (LNCS 4806, pp. 1222-1232), 3rd International IFIP Workshop on Semantic Web & Web Semantics, November 29-30, 2007.
148. Skuce D. & Lethbridge T.C. (1995). *CODE4: A Unified System for Managing Conceptual Knowledge.* International Journal of Human-Computer Studies, 42, pp. 413-451. http://citeseer.ist.psu.edu/skuce95code.html Fact Guru (http://www.factguru.com) is the commercial version of CODE4.
149. Smeaton A., Kelledy F., O'Donnell R., Quigley I. & Townsend E. (1995). *Expansion with WordNet and POS tagging of Spanish.* Proceedings of IA 1995, Montpellier, France, 1995.
150. Sowa J.F. (1984). *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, Reading, MA, 1984.
151. Sowa J.F. (1992). *Conceptual Graphs Summary.* Conceptual Structures: current research and practice (pp. 3-51; editors: Nagle T.E., Nagle J.A., Gerholz). Ellis Horwood, 1992.
152. Sowa J.F. (1993). *Relating Diagrams to Logic.* Proceedings of ICCS 1993 (pp. 1-35), Springer Verlag, LNAI~699, Laval, Quebec, 1993).
153. Sowa J.F. (2000). *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., Pacific Grove, CA, 2000. See also http://www.jfsowa.com/pubs/index.htm and http://users.bestweb.net/~sowa/ontology/
154. Sowa J.F. (2000b). *Thematic Roles.* 24 Jan 2000. http://users.bestweb.net/~sowa/ontology/thematic.htm
155. Sowa J.F. (2001). *Roles and Relations.* 4 July 2001. http://www.jfsowa.com/ontology/roles.htm
156. Sowa J.F. (2003). *Re: Ontology Registry.* Message 11841 of the SUO list. 26 November 2003. http://suo.ieee.org/email/msg11841.html
157. Sowa J.F. (2004). *Graphics and languages for the Flexible Modular Framework.* Proceedings of ICCS 2004 (LNAI 3127, pp. 31-51), August 2004.

158. Sowa J.F. (2005). *Theories, Models, Reasoning, Language, and Truth.* http://www.jfsowa.com/logic/theories.htm
159. Sowa J.F. (2006). *Concept Mapping.* http://www.jfsowa.com/talks/cmapping.pdf
160. Sowa J.F. (2007). *Common Logic Controlled English.* 15 March 2007. http://www.jfsowa.com/clce/clce07.htm
161. Stumme G. & Maedche A. (2001). *FCA-Merge: A Bottom-Up Approach for Merging Ontologies*. IJCAI 2001 (5th International Joint Conference on Artificial Intelligence, pp. 1-6, Morgan Kaufmann), Seattle, USA, August 2001.
162. Stutt, A. & Motta, E. (2004). *Semantic Learning Webs.* Journal of Interactive Media in Education, Special Issue on the Educational Semantic Web, 10.
163. Smith M.K., Welty C. & McGuinness D.L. (2004). *OWL Web Ontology Language Guide.* W3C Recommendation 10 February 2004. http://www.w3.org/TR/owl-guide/#UMLSyntaxExample
164. Swick R., Schreiber G., & Wood D. (2006) *Semantic Web Best Practices and Deployment Working Group.* http://www.w3.org/2001/sw/BestPractices/
165. Trombert-Paviot B., Rodrigues J.M., Rogers J.E., Baud R., van der Haring E., Rassinoux A.M., Abrial V., Clavel L. & Idir H. (2000). *GALEN: a third generation terminology tool to support a multipurpose national coding system for surgical procedures.* International Journal of Medical Informatics, Vol. 58-59 (pp. 71-85), 1 September 2000.
166. Verheij B. (1999). *Automated argument assistance for lawyers.* Proceedings of the International Conference on Artificial Intelligence and Law archive (pp. 43-52), Oslo, Norway, 1999.
167. Vrandecic D., Pinto H.S., Sure Y. & Tempich C. (2005). *The DILIGENT Knowledge Processes.* Journal of Knowledge Management, 9(5), pp. 85-96, October 2005.
168. Welty C.A. & Jenkins J. (1999). *Formal Ontology for Subject.* Journal of Knowledge and Data Engineering, 31(2), pp. 155-181. http://dx.doi.org/10.1016/S0169-023X%2899%2990021-6
169. Weitzner D. (2007). *Reciprocal Privacy (ReP) for the Social Web*. Web document created on December 12th 2007. http://dig.csail.mit.edu/2007/12/rep.html
170. Witbrock M., Baxter D., Curtis J., Schneider D., Kahlert R., Miraglia P., Wagner P., Panton K., Matthews G., & Vizedom A. (2003). *An Interactive Dialogue System for Knowledge Acquisition in Cyc.* Proceedings of IJCAI 2003 (pp. 138-145), Workshop on Mixed-Initiative Intelligent Systems, Acapulco, Mexico, August 9, 2003.
171. www-ACM-in-FL (2008). *Classification of the ACM*. October 2008.
http://www.webkb.org/kb/it/o_domain/d_ACM_classification.html
172. www-AI-Trader (2003). *AI-Trader*. September 2003. http://www.puder.org/aitrader/
173. www-AKT (2006). *The AKT Reference Ontology*. http://www.aktors.org/publications/ontology/
174. www-CG (2001). *Conceptual Graphs.* Draft of the specification of CGs for the ISO/JTC1/SC32/WG2 committee. Last update: 2 April 2001. http://users.bestweb.net/~sowa/cg/cgstand.htm
175. www-CG-tools (2009). *Conceptual Graphs – Tools.* http://conceptualgraphs.org/#Tools
176. www-CL (2009). *Controlled Natural Languages*. http://sites.google.com/site/controllednaturallanguage/
See also http://web.archive.org/web/19981202121830/http://www-uilots.let.uu.nl/Controlled-languages/
177. www-CYC (2009). *Cycorp* http://www.cyc.com/ and http://en.wikipedia.org/wiki/Cyc. See also OpenCYC at http://www.cyc.com/opencyc
178. www-COM (2009). *Cognitive map.* http://en.wikipedia.org/wiki/Cognitive_map
179. www-CM (2009). *The Concept Mapping Homepage.* http://users.edte.utwente.nl/lanzing/cm_home.htm
180. www-DC (2009). *The Dublin Core Metadata Initiative.* http://dublincore.org/
181. www-DPpedia (2009). *DPpedia.* http://wiki.dbpedia.org
182. www-FrameNet (2009). *FrameNet.* http://framenet.icsi.berkeley.edu/
183. www-GKBE (1998). *Generic Knowledge Base Editor.* http://www.ai.sri.com/~gkb/
184. www-GRDDL (2006). *Gleaning Resource Descriptions from Dialects of Languages (GRDDL)*. W3C Working Draft 24 October 2006. http://www.w3.org/TR/2006/WD-grddl-20061024/

185. www-IEML (2009). *Information Economy Meta Language.* http://www.ieml.org/
186. www-KIF (1998). *Knowledge Interchange Format.* Draft of the specification of KIF for the American National Standard (dpANS). NCITS.T2/98-004.
http://logic.stanford.edu/kif/dpans.html See also: http://www-ksl.stanford.edu/knowledge-sharing/kif/
187. www-KM (2006). *KM: The Knowledge Machine.* http://www.cs.utexas.edu/users/mfkb/km.html
188. www-Knizhnik-tools (2009). *Object Relational/Oriented Database Management Systems.* http://www.garret.ru/databases.html
189. www-LO (2009). *Learning object.* http://en.wikipedia.org/wiki/Learning_object
190. www-Linked-Data (2009). *Linked Data.* http://en.wikipedia.org/wiki/Linked_Data
191. www-LODP (2009). *Linked Open Data.*
http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
192. www-LOR (2009). *NMC Learning Object Repositories.* http://archive.nmc.org/projects/lo/repositories.shtml
193. www-LOS (2009). *Learning Objects and Standards*.
http://www.e-learningcentre.co.uk/eclipse/Resources/contentmgt.htm
194. www-Mizar (2009). *Mizar Home Page.* 18 September 2009. http://mizar.uwb.edu/teaching/pl/
195. www-MCF/XML (1997). *Meta Content Framework Using XML.* http://www.w3.org/TR/NOTE-MCF-XML/
196. www-MOF2 (2009). *Meta Object Facility (MOF) Core Specification.*
OMG Available Specification, formal/06-01-01. http://www.omg.org/docs/formal/06-01-01.pdf
197. www-MQL (2009). *MQL cheat sheet.* http://download.freebase.com/MQLcheatsheet-081208.pdf
198. www-ODP (2009). *Google/ODP directory*. http://www.google.com.au/dirhp?hl=en
199. www-ODM-RFP (2003). *Ontology Definition Metamodel RFP*. http://www.omg.org/cgi-bin/doc?ad/2003-03-40
200. www-OKBC (1998). *Open Knowledge Base Connectivity.* http://www.ai.sri.com/~okbc/
201. www-Ontolingua (1995). *Ontolingua ontology server*. http://ontolingua.stanford.edu
202. www-Ontolingua-FO (1994). *Theory FRAME-ONTOLOGY*.
http://www-ksl.stanford.edu/knowledge-sharing/ontologies/html/frame-ontology/index.html
203. www-Ontolingua-Library (1994). *Library of Ontologies Table of Contents*.
http://www-ksl.stanford.edu/knowledge-sharing/ontologies/html/index.html
(when available, http://ontolingua.stanford.edu gives a newer version).
204. www-Ontopedia (2009). *Ontopedia*. http://psi.ontopedia.net/
205. www-OpenGALEN (2009). *OpenGALEN.* http://en.wikipedia.org/wiki/OpenGALEN + www.opengalen.org
206. www-Powerloom (2009). *PowerLoom Knowledge Representation & Reasoning System.* http://www.isi.edu/isd/LOOM/PowerLoom/
207. www-QED (1995). *The QED Manifesto* http://ftp.mcs.anl.gov/pub/qed/manifesto
208. www-ReSIST (2006). *ReSIST ontology*. November 28th 2006. http://resist.ecs.soton.ac.uk/ontology/resist
209. www-ReSIST-in-FL (2008). *(Re-)presentation of the ReSIST ontology in FL*. October 2008. http://www.webkb.org/kb/it/p_securing/d_resist.html
210. www-RDF/XML (2004). *RDF Primer.* W3C Recommendation of 10 February 2004.
http://www.w3.org/TR/REC-rdf-syntax/
211. www-RDF/XML-reification (2004). *RDF Reification.* W3C Recommendation of 10 February 2004. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#reification
212. www-RDFS (2004). *RDF Vocabulary Description Language 1.0: RDF Schema.* W3C Recommendation of 10 February 2004. http://www.w3.org/TR/PR-rdf-schema/
213. www-RIF (2009). *RIF Basic Logic Dialect.* 2 October 2009. http://www.w3.org/2005/rules/wiki/BLD

214. www-Sewese (2009). *SeWeSe : Semantic Web Server.* http://www-sop.inria.fr/acacia/personnel/Fabien.Gandon/research/www_2007_dev_track/ See also http://www-sop.inria.fr/acacia/soft/sewese.html
215. www-SKOS (2007). *SKOS Simple Knowledge Organization System – Home Page.* http://www.w3.org/2004/02/skos/
216. www-SUO-D7 (2007). *D7 – Which languages are better than OWL?* Discussion on the SUO Mailing list. December 2007. All its messages are accessible from this one: http://suo.ieee.org/email/msg13472.html
217. www-SWTO (2006). *OWL specification of the Semantic Web (SW) Topics Ontology.* http://lsdis.cs.uga.edu/library/resources/ontologies/swtopics.owl
218. www-ToscanaJ (2009). *Welcome to the ToscanaJ Suite.* http://toscanaj.sourceforge.net/
219. www-XML-case-sensitive (2004). *Why is XML case-sensitive?* 29 November 2004. http://www.tkachenko.com/blog/archives/000354.html
220. www-XTM (2009). *XTM – TopicMaps.Org* http://www.topicmaps.org/
221. Yao H. & Etzkorna L. (2006). *Automated conversion between different knowledge representation formats.* Knowledge-Based Systems, Volume 19, Issue 6, October 2006, pp. 404-412.
222. Yessad A., Faron-Zucker C., Dieng-Kuntz C. & Laskri M.T. (2009). *Ontology-based Semantic Relatedness for Detecting the Relevance of Learning Resources.* Interactive Learning Environments Journal, Special issue "Semantic Technologies for Multimedia-enhanced Learning Environments", 2009.
223. Zhang L. (2002). *Knowledge graph theory and structural parsing.* PhD thesis, Twente University Press. http://doc.utwente.nl/38647/
224. Zarri G.P. (2009). *Representation and Management of Narrative Information: Theoretical Principles and Implementation.* Springer, "Series: Advanced Information and Knowledge Processing", 2009. http://www.springerlink.com/content/978-1-84800-077-3