Knowledge-System Technology: Ontologies and Problem-Solving Methods

V. Richard Benjamins[1] and Asunción Gómez Pérez[2]

[1] Dept. of Social Science Informatics (SWI), University of Amsterdam, Roetersstraat 15, 1018 WB Amsterdam, The Netherlands, richard@swi.psy.uva.nl, http://www.swi.psy.uva.nl/

[2] Facultad de Informática Universidad Politécnica de Madrid Campus de Montegancedo sn. Boadilla del Monte, Madrid, Spain, asun@fi.upm.es

Abstract

Ontologies and problem-solving methods are promising candidates for reuse in Knowledge Engineering. Ontologies define domain knowledge at a generic level, while problem-solving methods specify generic reasoning knowledge. Both type of components can be viewed as complementary entities that can be used to configure new knowledge systems from existing, reusable components. In this survey paper, we give an overview of approaches for ontologies and problem-solving methods.

Keywords: Ontologies, Problem-Solving Methods, Knowledge Engineering, Reuse

1 Introduction

In 1991, the ARPA Knowledge Sharing Effort [86] envisioned a new way in which intelligent systems could be built. They proposed the following: "Building knowledge-based systems today usually entails constructing new knowledge bases from scratch. It could be done by assembling reusable components. Systems developers would then only need to worry about creating the specialized knowledge and reasoners new to the specific task of their system. This new system would interoperate with existing systems, using them to perform some of its reasoning. In this way, declarative knowledge, problem-solving techniques and reasoning services would all be shared among systems. This approach would facilitate building bigger and better systems cheaply..."

Around the same time, several projects were carried out about methodologies to develop knowledge-based systems. Although these projects were not all talking directly about problem-solving methods and ontologies, they laid the foundation for these notions in the knowledge engineering community. These projects included Task Structures [30], Role-Limiting Methods [80], CommonKADS [96], Protégé [85], MIKE [4], Components of Expertise [102], EXPECT [106], GDM [111] and VITAL [36].

Since then, considerable progress has been made in developing the conceptual bases needed for building technology that allows knowledge-component reuse and sharing. However, we are still far from the ultimate objective. To enable sharing and reuse of knowledge and reasoning behavior across domains and tasks, Ontologies and Problem-Solving Methods (PSMs) have been developed. Ontologies are concerned with static domain knowledge and PSMs with dynamic reasoning knowledge. The integration of ontologies and PSMs is a possible solution to the "interaction problem" [27], which hampered reuse in the eighties. The interaction problem states that representing knowledge for the purpose of solving some problem is strongly affected by the nature of the problem and the inference strategy to be applied to the problem. Through ontologies and PSMs this interaction can be made explicit in the notion of assumptions and taken into consideration. PSMs and ontologies can be seen as complementary reusable components to construct knowledge systems from reusable components. In order to build full applications of information and knowledge systems from reusable components, both PSMs and ontologies are required in a tightly integrated way.

Ontologies aim at capturing domain knowledge in a generic way and provide a commonly agreed understanding of a domain, which may be reused and shared across applications and groups [31]. Ontologies provide a common vocabulary of an area and define -with different levels of formality- the meaning of the terms and the relations between them. They are usually organized in taxonomies and typically contain modeling primitives such as classes, relations, functions, axioms and instances [62]. Popular applications of

ontologies include knowledge management, natural language generation, enterprise modeling, knowledge-based systems, ontology-based brokers, and interoperability between systems.

Problem-solving methods (PSMs) describe the reasoning process of a knowledge-based system (KBS) in an implementation- and domain-independent manner. A PSM defines a way of how to achieve the goal of a task. It has inputs and outputs and may decompose a task into subtasks. In addition, a PSM specifies the data flow between its subtasks. Control knowledge determines the execution order and iterations of the subtasks of a PSM.

In Section 2, we will discuss several aspects of ontologies and in Section 3 we do the same for problem-solving methods. In Section 4, we discuss our findings and present conclusions. Finally, at the end of the paper in Section 5, we list a number of links to relevant website.

2 Ontologies

The aims of this section are to provide answers to the following questions: What is an ontology? What principles should I follow to build an ontology? What are the components of an ontology? What types of ontologies exist? How are ontologies organized in libraries? What methods should I use to build my own ontology? Which techniques are appropriate for each step? How do software tools support the process of building and using ontologies? What are the most well-known ontologies? What are the uses of ontologies? Which principles should I use to select the best ontology for my application? To answer the above questions, the section is organized as follows. First, the theoretical foundations of the ontological engineering field will be presented. This will be followed by a presentation of some existing ontologies. The second part will address methodologies for building ontologies. The third part will present tools for building ontologies. Finally, the last will be related to uses of ontologies in applications.

2.1 Theoretical Foundations

What is an ontology? The word ontology has been taken from Philosophy, where it means a systematic explanation of Existence. In the last decade, the word ontology became a fashionable word in the Knowledge Engineering Community. We have seen many definitions about what an ontology is and also how such definitions have changed and evolved over time. One of the first definitions is due to Neches and colleagues [86].

> "An ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary".

We can say that this definition tells us how to proceed to build an ontology, giving us vague guidelines: identify basic terms and relations between terms, identify rules to combine them, provide definitions of such terms and relations. Note that according to this definition, an ontology includes not only the terms that are explicitly defined in it, but also terms that can be inferred using rules. A few years later, Gruber's definition [62] became the most referenced in the literature:

> "An ontology is an explicit specification of a conceptualization",

Based on the definition of Gruber, many definitions of ontologies have been proposed. In 1995, Guarino and Giaretta [67] collected seven definitions and provided corresponding syntactic and semantic interpretations. In 1997, Borst [22] slightly modified Gruber's definition saying that:

> "Ontologies are defined as a formal specification of a shared conceptualization".

These two definitions have been explained by Studer et al [105] as follows:

> " Conceptualization refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine-readable. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private to some individual, but accepted by a group."

There are also ontology definitions based on the the process followed to build the ontology. For example, the definition given by Bernaras and colleagues [18] in the framework of the KACTUS project [97]:

> "An ontology provides the means for describing explicitly the conceptualization behind the knowledge represented in a knowledge base."

In this approach, the ontology is built (following a bottom-up approach) starting from an application knowledge base, by means of a process of abstraction. As more applications are built, the ontology becomes more general, and, therefore, moves further away from what would be a knowledge base.

Another strategy for building ontologies is to reuse large ontologies like SENSUS [107] (with more than 50.000 concepts) to build domain specific ontologies:

> "an ontology is a hierarchically structured set of terms for describing a domain that can be used as a skeletal foundation for a knowledge base"

Recently, some definitions, like the one in [115] dilute the original meaning of the word ontology. The aim is to export the notion of ontology to other disciplines (object oriented, databases, etc.) that also build domain models using concepts, relations, properties, etc., but with a different level of semantic formality:

> "An ontology may take a variety of forms, but necessarily it will include a vocabulary of terms and some specification of their meaning. This includes definitions and an indication of how concepts are inter-related which collectively impose a structure on the domain and constrain the possible interpretations of terms."

As a main conclusion to this section, we can say that, although there are several ontology definitions around, there is general consensus on the use of ontologies. Different definitions provide different and complementary viewpoints on the same reality. Some definitions are independent of the process followed to build the ontology and of its use in applications, i.e. [86, 62, 22]. Other definitions are influenced by the ontology development process, i.e., [107, 18].

What are the components of ontologies? An ontology describes the subject matter using the notions of concepts, instances, relations, functions and axioms [62]. Concepts in the ontology are organized in taxonomies through which inheritance mechanisms can be applied. Sometimes, the notion of ontology is diluted, in the sense that taxonomies are considered to be full ontologies [105].

- Concepts are used in a broad sense. A concept can be anything about which something is said and, therefore, could also be the description of a task, function, action, strategy, reasoning process, etc.
- Relations represent a type of interaction between concepts of the domain. They are formally defined as any subset of a product of n sets, that is: R: C1 x C2 x ... x Cn. Examples of binary relations include: subclass-of and connected-to.
- Functions are a special case of relations in which the n-th element of the relationship is unique for the n-1 preceding elements. Formally, functions are defined as: F: C1 x C2 x ... x Cn-1 $\rightarrow$ Cn. Examples of functions are Mother-of and Price-of-a-used-car that calculates the price of a second-hand car depending on the car-model, manufacturing date and number of kilometers.
- Axioms are used to model sentences that are always true.
- Instances are used to represent elements.

Once the main components of ontologies have been identified, the ontology can be implemented in a various kind of languages [113]: highly informal if they are in natural language; semi-informal, that is, in a restricted and structured form of natural language; semi-formal if they are expressed in an artificial and formally defined language (like Ontolingua); and rigorously formal if expressed in languages that provide

meticulously defined terms with formal semantics, theorems, and proofs of properties like soundness and completeness.

Most of the existing ontologies are implemented in formal languages (see section 2.3) like: Ontolingua [62], CycL [77], Loom [78] and FLogic [74]. It is important to mention here that there exist important connections between (1) the ontology components, (2) the knowledge representation paradigms (frame-based approach, Description Logic, Logic) used to formally represent such components, and (3) the languages used to implement them under a given knowledge representation paradigm. For example, concepts can be seen as classes in frame-based systems and as unary predicates in First Order Logic. Within the frames knowledge representation paradigm, a class could be implemented in Ontolingua using different primitives.

What principles should I follow to build ontologies? Here we summarize some design criteria and a set of principles that have shown to be useful for the development of ontologies. Since many of the principles are interrelated, we also provide some examples of why they are important.

- Clarity and Objectivity [63], which means that the ontology should provide the meaning of defined terms by providing objective definitions along with a natural language documentation. Usually, ontology developers provide poor informal natural language descriptions (or no description at all), which hinders the user's understanding of the more formal definition.

- Completeness [63], which means that a definition expressed in terms of necessary and sufficient conditions is preferred over a partial definition (defined only through necessary or sufficient conditions).

  Special mention deserves the principle of completeness of a taxonomy in terms of the partitions (disjoint and exhaustive) defined between its classes. An exhaustive subclass partition adds a completeness constraint to the represented subclasses. Three types of incompleteness have been identified in [58]. Incompleteness of concept classification happens whenever concepts existing in the domain are overlooked and not included in the taxonomy. For example, when categorizing musical instruments as string instruments and wind instruments and overlooking, for example, percussion instruments. The second type of incompleteness appears when classes are not declared disjoint while they are disjoint in reality, for example dogs and cats as subclasses of mammals. A related error happens when classes form an exhaustive partition (e.g. odd and even numbers) but this is not defined in the ontology.

- Coherence [63], to permit inferences that are consistent with the definitions. Consistency [56], refers to the possibility to check for contradictory knowledge from valid definitions.

  Examples of inconsistencies in taxonomies include [58]: circularities, partition errors and semantic inconsistency errors. Circularities occur when a class is defined as a specialization or generalization of itself. An example of a partition error is when dogs and cats form a disjoint partition of mammals, and Pluto is defined as an instance of both classes. Finally, semantic inconsistency errors occur when the developer makes an incorrect classification, that is, categorizes a concept as a subclass of a class to which it does not belong, for example, dog as a subclass of house.

- Minimization of the semantic distance between sibling concepts [5]. Similar concepts are usually grouped and represented as subclasses of one class and should be defined using the same set of primitives, whereas concepts which are less similar are represented further away in the hierarchy. A counter example of this principle (which thus violates it) can be seen in the Standard-Unit ontology [61] from the Ontolingua Server. An inspection of its code shows how definitions that refer to similar concepts (Ampere and Meter) are represented following different patterns. This may cause lack of understandability and extendibility. For example:

```
(Define-Frame Ampere
        : Own-Slots
        (( Documentation "Si electrical current unit.")
         (Instance-Of Unit-Of-Measure)
         (Quantity.Dimension Electrical-Current-Dimension))
        : Axioms
```

```
        ((= (Quantity.Dimesion Ampere) Electrical-Current-Dimension)))
(Define-Instance Meter (Unit-Of-Measure)
        "SI length unit. No conversion is given
         because this is a standard."
        : Axiom-Def
        (And (= (Quantity.Dimesion Meter) Length-Dimension)
            (Si-Unit Meter)))
```

To improve ontology understanding and ease of inclusion of new definitions, it would be advisable for definitions that are children of the same parent (SI-Unit) to be defined according to the same template.

- Maximum monotonic extendibility [63]. This means that new general or specialized terms should be included in the ontology in such a way that is does not require the revision of existing definitions.
- Minimal ontological commitments[3] [63], which means to make as few claims as possible about the world being modeled, giving the parties committed to the ontology freedom to specialize and instantiate the ontology as required.

  As an example of an ontological commitment, take a Mereology ontology, which defines, among others, the part-of relationship and its properties, such as transitivity. If a particular ontology reuses this Mereology ontology, the new ontology also commits to transitivity of the part-of relationship. Therefore, it is important to have such commitments explicitly defined.
- Ontological Distinction Principle [21], which means that classes corresponding to different identity criteria must be disjoint. An identity criterion isolates the core properties considered to be invariant for an instance of a class.
- Diversification of hierarchies to increase the power provided by multiple inheritance mechanisms [5]. The goal is to identify general concepts that are specialized into more specific and disjoint concepts down to domain instances.
- Standardization of names whenever is possible [5]. An example of violating this principle can be seen in the Standard-Unit ontology [61]. The different multiples and divisors of Ampere are called: Milli-Amp, Nano-Ampere and Pico-Ampere. To ease ontology understanding and improve clarity, it would have been better to use the same naming conventions. Therefore, the above-mentioned names should be standardized and denoted as: Milli-Ampere, Nano-Ampere and Pico-Ampere, respectively.

What types of ontologies already exist? One of the main motivations of this section is to provide a common understanding of the vocabulary used to classify ontologies. Although there exist some works to categorize ontologies, and each work uses its own terminology, we can say that there is basic semantic agreement concerning the meaning of the terms. Moreover, ontology developers usually classify their ontologies correctly under any of the classifications.

Figure 1 summarizes several relevant types of ontologies. Mizoguchi and colleagues [82] present a typology of ontologies based on the idea that knowledge should be de-compiled into context-dependent and context independent parts in order to enable knowledge sharing and reuse. Based on this, they classify ontologies on the following categories: domain ontologies, common sense ontologies, meta-ontologies and task ontologies.

According to Van Heijst and colleagues [120], ontologies are classified in two dimensions: the amount and type of structure of the conceptualization and the subject of the conceptualization. With respect to the first dimension, they distinguish three categories: Terminological ontologies such as lexicons, Information ontologies such database schemata and Knowledge modeling ontologies that specify conceptualizations of the knowledge. This kind of ontologies usually has a richer internal structure. The other dimension is the subject of the conceptualization. They distinguish four categories: application ontologies, domain ontologies, generic ontologies and representation ontologies.

[3] "Ontological commitments refer to agreement to use the shared vocabulary in a coherent and consistent manner. They guarantee consistency, but not completeness of an ontology" [61].

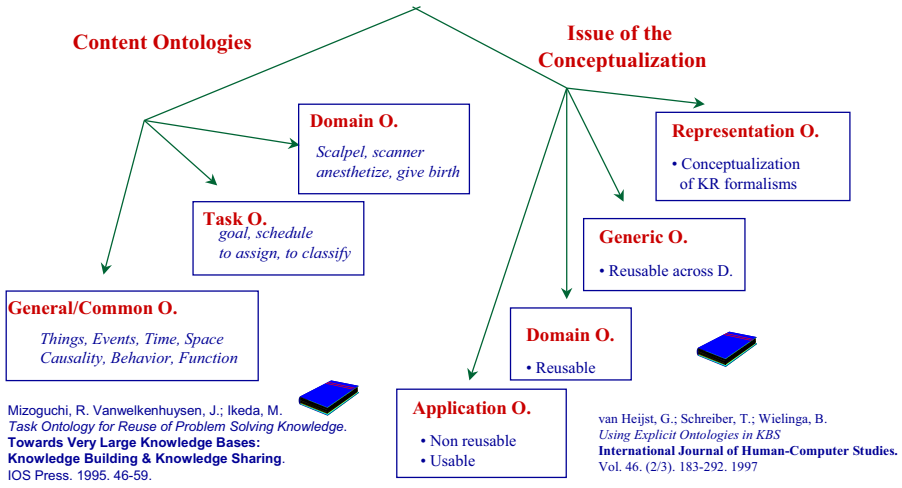

Figure 1: Types of ontologies according to Mizoguchi and colleagues [82] and van Heijst and colleagues citevanHeijst:97a

The aim of this section is not to provide an exhaustive typology of ontologies as presented in the above papers [120, 82]. Our aims are rather to present the most commonly used types of ontologies, and to provide a unified understanding of the vocabulary used to classify ontologies. Simultaneously, we also provide examples of existing and representative ontologies that could be classified under the types. Finally, we will describe how different types of ontologies could be combined to build a new ontology (see Figure 1).

.

- Knowledge Representation ontologies [120] capture the representation primitives used to formalize knowledge in knowledge representation paradigms. The most representative example is the Frame-Ontology [62] available at the Ontolingua Server. This ontology captures the representation primitives used in frame-based languages(i.e., classes, subclasses, attributes, values, relations and axioms). It allows other Ontolingua ontologies to be specified using frame-based conventions. It is implemented in KIF 3.0 [53].

- General/Common ontologies [82] include vocabulary related to things, events, time, space, causality, behavior, function, etc.

  The CYC ontology [77] is a common sense ontology that provides a vast amount of fundamental human knowledge. The CYC ontology is divided into many micro-theories. CYC Ontologies are implemented in CycL language.

- Top-Level Ontologies or Upper Level Ontologies provide general notions under which with all the terms in existing ontologies should be linked to. At the moment the main problem is that there does not exist a unified top-level ontology in the community. Examples of top level ontologies are: Sowa's boolean lattice [99], PANGLOSS [76], Penman Upper Level [9], CYC [77], Mikrokosmos [79] and Guarino's top level proposal [65, 66].

- Meta-ontologies, also called Generic Ontologies or Core Ontologies [120] are reusable across domains. The Mereology ontology [22] could be the most typical example. It defines the part-of relation and its properties. This relation allows to express that devices are assembled of components, each of which might -on its turn- be decomposed into subcomponents.

- Domain ontologies [82, 120] are reusable in a given domain. They provide vocabularies about the concepts within a domain and their relationships, about the activities that take place in that domain, and about the theories and elementary principles governing that domain.

In the domain of engineering ontologies, the EngMath ontology [61] and PhysSys [22] deserve special mention. EngMath is an Ontolingua ontology developed for mathematical modeling in engineering. PhysSys is an engineering ontology for modeling, simulating and designing physical systems. PhysSys reuses the EngMath and a Mereology ontology.

In the domain of enterprise modeling process, the Enterprise Ontology [4] [113] is a collection of terms and definitions relevant to business enterprises. Ontologies built at the TOVE [64] (Toronto Virtual Enterprise)[5] project are: Enterprise Design Ontology, Project Ontology, Scheduling Ontology, or Service Ontology.

An illustrative example of ontologies for Knowledge Management is the (KA) [2] ontology[6] [14], to be used by the Knowledge Annotation Initiative of the Knowledge Acquisition Community. This ontology is being built jointly and distributively with people at different locations.

- The most illustrative linguistic ontologies are the Generalized Upper Model [10], WordNet [81] and SENSUS [107]. The Generalized Upper Model (GUM[7]) is a general task and domain-independent linguistic ontology. To make it portable across different languages (English, German, Spanish, Italian, etc.), the GUM ontology only includes the main linguistic concepts and how they are organized across languages, and omits details that differentiate languages. WordNet is a lexical database for English based on psycholinguistic principles. Its information is organized in units called "synsets", which are sets of synonyms that are interchangeable in a particular context and are used to represent different meanings. SENSUS is a natural language based ontology whose goal is to provide a broad conceptual structure for work in machine translation. It was developed by merging and extracting information from existing electronic resources.

- Task ontologies [82] provide a systematic vocabulary of the terms used to solve problems associated with particular tasks (either domain independent or domain dependent). For instance, in diagnosis terms would include the concepts "observation", "hypothesis" and "goal", as well as the actions "hypothesis generation" and "hypothesis discrimination".

- Domain-Task ontologies are task ontologies reusable in a given domain, but not across domains.

- Method ontologies provide definitions of the relevant concepts and relations used to specify a reasoning process to achieve a particular task [112]. A slightly different definition is given in [55], where a method ontology defines the concepts and relationships that are used by the method to achieve its goal. Thus, a method ontology refers to a domain ontology from a method point of view. Method and task ontologies are used to define problem-solving methods (see Section 3).

- Application ontologies [120] contain the necessary knowledge for modeling a particular application.

The reusability-usability trade-off problem [75] applied to the ontology field states that the more reusable an ontology is, the less usable it is, and vice versa. Meta-ontologies, domain ontologies and applications ontologies capture static knowledge in a problem-solving independent way, whereas PSM ontologies, task ontologies and domain-task ontologies are concerned with problem solving knowledge. All these kind of ontologies can be taken from different libraries and combined to build a new ontology (see Figure 2). The first thing to do to model a new ontology using existing ontologies from the library is to decide which knowledge representation paradigm to use to formalize knowledge, which will then be committed to into a knowledge representation ontology. Having selected the knowledge representation ontology, the next step is to decide whether general/common ontologies are needed in the new ontology. If they are required, new ontologies are built and entered into the library or reused from the library. Then, simultaneously, domain knowledge and problem-solving knowledge can be modeled. So, when domain knowledge is modeled, first generic ontologies, then domain ontologies, and finally application domain ontologies are built. When problem-solving knowledge is modeled, first Task and PSMs ontologies, then

[4] http://www.aiai.ed.ac.uk/project/enterprise
[5] http://www.ie.utoronto.ca/EIL
[6] http://www.aifb.uni-karlsruhe.de/WBS/broker/KA2.html
[7] http://www.darmstadt.gmd.de/publish/komet/gen-um/newUM.html

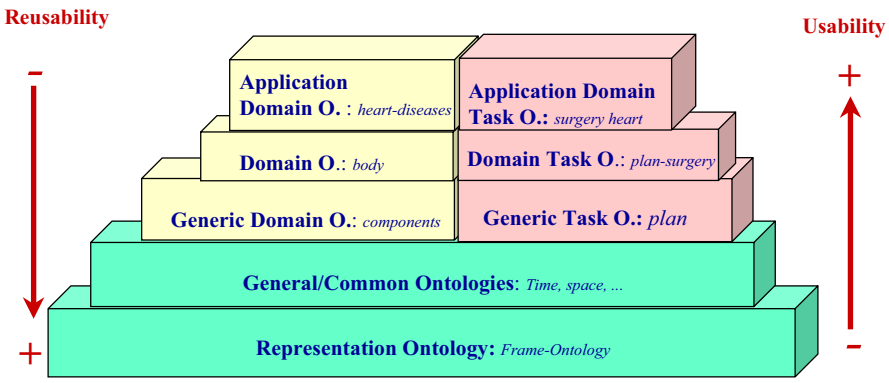

Figure 2: The reusability-usability trade-off problem.

domain task ontologies and finally application domain task ontologies are modeled. Method and task ontologies allow the interaction between problem-solving and domain ontologies to be explicitly stated in assumptions [16, 17, 47].

2.2 Methodologies for building ontologies

The ontology building process is a craft rather than an engineering activity. Each development team usually follows its own set of principles, design criteria and phases in the ontology development process. The absence of commonly agreed on guidelines and methods hinders the development of shared and consensual ontologies within and between teams, the extension of a given ontology by others and its reuse in other ontologies and final applications. If ontologies are built on a small scale, some activities can be skipped. But, if you intend to build large-scale ontologies with some guarantees of correctness and completeness, it is advisable to steer clear of anarchic constructions and to follow a methodological approach.

Until now, few domain-independent methodologies for building ontologies have been reported, such as Uschold's methodology [116, 113], Grüninger and Fox methodology [64], METHONTOLOGY [59, 48, 57, 49]. Other authors like Bernaras and colleagues [18] build the ontology by abstraction of the knowledge bases used in KBSs. Finally, a completely different approach is used when ontologies are built using large ontologies like SENSUS [107]. A comparative study of these methodologies in presented [50].

Uschold's methodology [116, 113] is based on the experience of building the Enterprise Ontology, which includes a set of ontologies for enterprise modeling, and proposes the following steps: (1) identify the purpose and scope of the ontology; (2) build the ontology by capturing knowledge, coding knowledge and integrating the knowledge with existing ontologies; (3) evaluate the ontology; (4) documentation; and (5) guidelines for each phase.

Grüninger and Fox's methodology [64] is based on the experience of building an enterprise modeling ontology in the framework of the TOVE project. Essentially, it involves building a logical model of the knowledge that is to be specified in the ontology. This model is not built directly. Firstly, the development of the ontology is motivated by scenarios that arise in the application. The scenario also provides a set of intuitively possible questions and solutions to the scenario problem, like a kind of specification. These questions receive the name of competency questions, which initially are described informally. This description is then formalized in a language based on first-order predicate calculus. The competency questions are the basis for a rigorous characterization of the knowledge that the ontology has to cover, and they specify the problem and what constitutes a good solution to the problem. By a composition and decomposition mechanism, competency questions and their answers can be used to answer more complex competency questions in other ontologies, allowing the integration of ontologies.

The METHONTOLOGY framework [59, 48, 57, 49] enables the construction of ontologies at the knowledge level. It includes: (a) the identification of the ontology development process, which refers to

which tasks (planning, control, specification, knowledge acquisition, conceptualization, integration, implementation, evaluation, documentation, configuration management, etc.) one should carry out when building ontologies; (b) a life cycle based on evolving prototypes, which identifies the stages through which the ontology passes during its lifetime; and (c) the methodology itself, which specifies the steps to be taken to perform each activity, the techniques used, the products to be output and how they are to be evaluated. The main phase is the conceptualization phase. During both specification and conceptualization, a process of integration was completed using in-house and external ontologies. This framework is partially supported by a software environment called Ontology Design Environment (ODE) [20], [49]. Several ontologies have been developed using METHONTOLOGY and ODE: CHEMICALS [49], Environmental pollutants ontologies [60], the Reference-Ontology [5] and the restructured version of the (KA) [2] ontology [20].

All these methodologies have in common that they start from the identification of the purpose of the ontology and the need for domain knowledge acquisition. However, having acquired a significant amount of knowledge, Uschold's methodology proposes coding in a formal language and METHONTOLOGY proposes expressing the idea as a set of intermediate representations (IR). Then the ontology is generated using translators. These IRs bridge the gap between, on the one hand, how people see a domain and, on the other hand, the languages in which ontologies are formalized. These intermediate representations provide a user-friendly approach for both knowledge acquisition and evaluation by computer scientists and domain experts who are not knowledge engineers [3].

The need for ontology evaluation is also identified in the three above methodologies. Uschold's methodology includes this activity but it does not state how it should be carried out. Grüninger and Fox propose identifying a set of competency questions. Once the ontology has been expressed formally, it is compared against this set of competency questions. Finally, METHONTOLOGY proposes that evaluation activities be carried out throughout the entire lifetime of the ontology development process. Most of the evaluation is done in the conceptualization phase. This methodology has been proposed to build ontologies by the Foundation for Intelligent Physical Agents (FIPA[8]).

As stated in [50], the main conclusion at this point is that none of the methodologies is fully mature. Each group has and uses its own methodology and there does not yet exist a common methodology that everybody agrees on. Therefore, additional research has to be performed in this direction.

2.3 Languages and environments for building ontologies

Which are the most commonly used languages to build ontologies? Basically, several representation systems have been reported for formalizing ontologies under a frame-based modeling approach, a logic-based approach or even both. The most representative languages are Ontolingua [62], CycL [77], Loom [78] and FLogic [74].

Ontolingua is a language based on KIF and on the Frame Ontology, and is the ontology-building language used by the Ontolingua Server. The Ontolingua language allows ontologies to be built in any of the following three manners: (1) using KIF expressions; (2) using exclusively the Frame Ontology vocabulary; (3) using both languages at the same time, depending on ontology developer preferences. In any case, the Ontolingua definition is composed of a heading, an informal definition in natural language, and a formal definition written in KIF or using the frame ontology vocabulary. A GFP [33] application is required in order to reason with Ontolingua Ontologies. Currently, GFP has evolved into OKBC (Open Knowledge Base Connectivity) [32].

CycL is Cyc's knowledge representation language. CycL is a declarative and expressive language, similar to first-order predicate calculus with extensions. CycL uses a form of circumscription, includes the unique names assumption, and can make use of the closed world assumption where appropriate. CycL has an inference engine to perform several kinds of reasonings.

LOOM is a high-level programming language based on first-order logic which belongs to the KL-ONE family. The LOOM language provides: an expressive and explicit declarative model specification language, powerful deductive support, several programming paradigms, and knowledge-base services.

FLogic is an integration of frame-based languages and first-order predicate calculus. It includes objects (simple and complex), inheritance, polymorphic types, query methods and encapsulation. Its deductive

[8] http://www.fipa.org

system works with the theory of predicate calculus and structural and behavioral inheritance.

More recently, the Ontology Interchange Language (OIL) [72] has been proposed as a standard for specifying and exchanging ontologies, and to integrate them with the web. OIL is based on notions from Description Logics, Frame systems and Web standards.

How do software tools support the process of building and using ontologies? The main tools for building ontologies are: The Ontolingua Server [38], Ontosaurus [9] [107], ODE [20, 49] and Tadzebao and Webonto [35]. See [37] a comparison of these ontology tools.

The Ontolingua Server is the best known environment for building ontologies in the Ontolingua language. It is a set of tools and services that support the building of shared ontologies between geographically distributed groups. It was developed in the context of the ARPA Knowledge Sharing Effort by the Knowledge Systems Laboratory at Stanford University. The Ontolingua Server architecture provides access to a library of ontologies, translators to languages (Prolog, CORBA's IDL, CLIPS, Loom, KI) and an editor to create and browse ontologies. There are three modes of interaction: remote collaborators that are able to write and inspect ontologies; remote applications that may query and modify ontologies stored at the server over the Internet using the generic frame protocol; and stand-alone applications.

Ontosaurus is being developed by the Information Sciences Institute at the University of South California. It consists of two parts: an ontology server that uses Loom as knowledge representation system and an ontology browser server that dynamically creates html pages (including image and textual documentation) that displays the ontology hierarchy and it uses html forms to allow the user to edit the ontology. Translators from loom to Ontolingua, KIF, KRSS and C++ have also been developed.

ODE (Ontology Design Environment) is being developed by the Computer Science department of the Technical University of Madrid. The main advantage of ODE is the conceptualization module for building ontologies, which allows the ontologist to develop the ontology at the knowledge level using a set of intermediate representations that are independent of the target language in which the ontology will be implemented. Once the conceptualization is complete, the code is generated automatically using ODE code generators (Ontolingua, FLogic and a relational database). So, non-experts in the languages in which ontologies are implemented could specify and validate ontologies using this environment.

Tadzebao and WebOnto are complementary tools that are being developed by the Knowledge Media Institute at The Open University. Tadzebao enables knowledge engineers to hold synchronous and asynchronous discussion about ontologies and WebOnto supports the collaborative browsing, creation and editing of ontologies.

Concerning the maturity of ontology tools, we agree with the conclusion in [37]: current ontology tools are not yet ready for direct use by domain experts, but neither do they require knowledge representation experts.

2.4 Applications that use ontologies

Although ontologies can be used to communicate between systems, people, and organizations, interoperate between systems, and support the design and development of knowledge-based and general software systems [113], the number of applications built that use ontologies to model the application knowledge is small. Moreover the ontologies that have been built, often are built just for a given application without special consideration for sharing and reuse. Several problems make difficult the reuse of existing ontologies in applications [5]: Ontologies are dispersed over several servers; the formalization differs depending on the server on which the ontology is stored; ontologies on the same server are often described with different levels of detail; and there is no common format for presenting relevant information about the ontologies so users can decide which ontology best suits their purpose. These problems are probably the cause for the relatively small number of known applications until now. Several applications that use ontologies can be found in the proceedings of the workshop on Applications of ontologies and PSMS held in conjunction with ECAI98 (see `http://delicias.dia.fi.upm.es/WORKSHOP/ECAI98/index.html`)

There exist several applications that use natural language ontologies. The GUM is being used in natural language generation applications in different languages: Penman [9], KOMET [8], TechDoc [93], AlFresco

[9] `http://indra.isi.edu:8000`

[104], GIST [73], OntoGeneration [3], and the language of [51]. WordNet is used by Hermes [70] and OntoSeek [68].

In the domain of enterprise modeling, the Enterprise tool set (see: `http://www.aiai.ed.ac.uk/project/enterprise` for more information) is the most relevant environment built with the Enterprise ontology. The Enterprise Design Workbench and the Integrated Supply Chain Management Project use TOVE Ontologies.

Recently, ontologies are being used by WWW brokers in different domains. Ontobroker [10] [42] for knowledge management in the context of the Knowledge Annotation Initiative of the Knowledge Acquisition Community, (Onto)2Agent [5] for selecting ontologies that satisfy a given set of constraints and Chemical OntoAgent [5] for teaching chemistry.

In the domain of information systems design, Comet [121] supports the design of software systems, and Cosmos [121] supports engineering negotiation. Both systems give design feedback to their users.

KACTUS [97] was an ESPRIT project on modeling knowledge of complex technical systems for multiple use and the role of ontologies to support it.

Plinius [119] is a semi-automatic knowledge acquisition system from natural language text in the domain of ceramic materials, their properties and their production processes.

3 Problem-Solving Methods

Problem-Solving Methods (PSMs) are nowadays recognized as valuable components for constructing knowledge-based systems (KBSs). This is manifested by the fact that the notion of PSM is present in leading knowledge engineering frameworks such as Task Structures [30], Role-Limiting Methods [80], CommonKADS [96], Protégé [85], MIKE [4], Components of Expertise [102], EXPECT [106], GDM [111] and VITAL [36]. PSMs describe the reasoning process of a knowledge-based system (KBS) in an implementation- and domain-independent manner.

Work on PSMs covers different areas such as the identification of task-specific PSMs (for diagnosis, planning, assessment, etc.), how to store and index PSMs in libraries, how to formalize PSMs, etc. The issues involved in reusing PSMs include finding the right PSM (that does -part of- the job), checking whether it is applicable in the situation at hand, and modifying it to fit the domain. In order to reuse PSMs successfully in a real-life application, one has to understand these processes. A PSM may be characterized as follows:

- A PSM specifies which inference steps have to be carried out for achieving the goal of a task.
- A PSM defines one or more control structures over these steps.
- Knowledge roles specify the role that domain knowledge plays in each inference step. These knowledge roles define a domain-independent generic terminology (part of the method ontology). There are two types of roles: static roles describe the domain knowledge needed by the PSM; dynamic roles form the input and output of inference steps.

PSMs play an important role in knowledge engineering and knowledge acquisition. They can for instance be used to efficiently achieve goals of tasks through the application of domain knowledge [47], they can guide the acquisition process of domain knowledge, and they can facilitate KBS development through their reuse.

Before discussing different approaches to PSMs, we will briefly present a general architecture of PSMs (taken from[16]).

3.1 Architecture of PSMs

Most approaches agree that a PSM consists of three related parts, describing *what* a PSM can achieve, *how* it achieves it and what it *needs* to achieve it, respectively referred to as the PSM's *competence*, *operational specification* and *requirements/assumptions* (see Figure 3).

[10] `http://www.aifb.uni-karlsruhe.de/WBS/broker/`

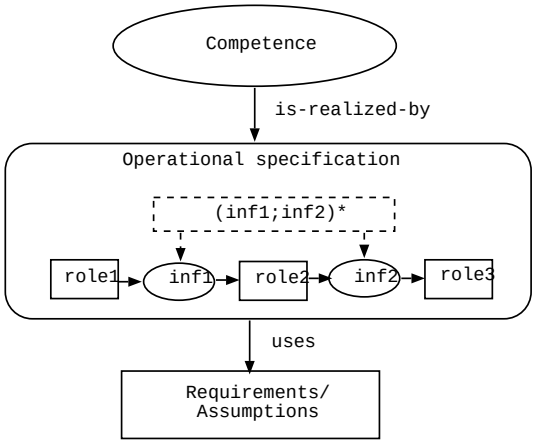

Figure 3: The architecture of a PSM.

Competence The competence of a PSM is a declarative description of the input-output behavior and describes what can be achieved by the PSM.

Operational specification The operational specification of a PSM describes the reasoning process which delivers the specified competence if the required knowledge is provided. It consists of inference steps and the knowledge and control-flow between them. The inference steps specify the reasoning steps that together accomplish the competence of the method. They are described by their input/output relation and can be achieved by either a method (which means that a PSM can be hierarchically decomposed) or a primitive inference (an atomic reasoning step which is not further decomposed). The knowledge flow takes place through dynamic roles, which are stores that act as input and output of inferences. Finally, the control of a PSM describes the order of execution of the inference steps. Control knowledge can be specified in advance, if known, or can be opportunistically determined at run time depending on the dynamic problem-solving situation [12]. Problem-solving methods can be used to efficiently achieve goals of tasks through the application of domain knowledge [47].

Requirements/Assumptions Requirements/assumptions of a PSM describe the domain knowledge needed by the PSM to achieve its competence. Examples of such requirements in a parametric design task include the availability of heuristics that link violated constraints to possible repair actions (fixes), and the fact that a preference relation must describe a complete ordering. The requirements describe what a PSM expects in return for the competence it provides.

The internal relationship between the competence and operational descriptions of the method is that it has to be ensured that, assuming that the knowledge requirements are satisfied, the operational description describes a way to achieve the competence [46].

A PSM in context PSMs can be used to realize tasks by applying domain knowledge. Thus, the external context of a PSM is formed by two parties: a task to be realized and domain knowledge to be applied. When we want to use a PSM to build a knowledge-based system, we have thus to connect the PSM with both the task and the domain knowledge. Since PSMs are generic, reusable components, they may not fit perfectly in the context, or, in other words, there may be gaps (see Figure 4).

These gaps can exist for several reasons. In both directions (i.e. towards the domain knowledge and the task) the PSM may use different terminology than that of the domain knowledge and task, in which case a renaming process can bridge the gap. In the direction of the task, it may happen that the PSM's competence is not strong enough to realize what is specified by the task. In this case, to bridge the gap, the task may be weakened by making simplifying assumptions. Towards the domain knowledge, the knowledge required by the PSM may not be fully given by the domain knowledge, in which case additional knowledge needs to be acquired. Recently, an overall architecture has been developed for describing KBS development from PSMs and ontologies -UPML [41]- in which PSMs, ontologies and bridges are formalized.

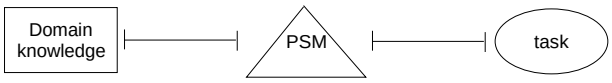

Figure 4: The two possible gaps that may prevent a PSM from being applied in its context.

3.2 Issues in PSM research

Problem-solving methods play an important role in knowledge acquisition and knowledge engineering where they have several purposes:

- KBS construction (knowledge engineering): a PSM can be helpful to describe the process of *creating* a problem solver that achieves the goal of a particular task. Often this implies a task decomposition approach.
- KBS specification (reasoning): a PSM can describe an efficient *reasoning* process that achieves the goal of a task. In this sense, a PSM concerns the product of the creation process, and is related to the design model of a KBS.
- Cognitive modeling: a PSM can describe a cognitive model of human problem-solving. An interesting question is to what extent PSMs can be used to generate cognitively adequate explanations of the reasoning process of a knowledge-based system.

PSM development Work in this area is concerned with how PSMs are constructed in the first place. One way to do this, is by analyzing human problem-solving behavior and representing this behavior computationally. This has been traditionally the focus of Cognitive Psychology. Another way to do this, is to perform reverse engineering of existing expert systems, as has been performed by Clancey [34] when he "discovered" Heuristic Classification. These two ways of developing PSMs essentially involve a creative activity, for which no methodological support exists. In the last decade, several methodologies have been developed to support knowledge modeling and the development of knowledge-based systems, such as CommonKADS [94, 96], Protégé [85], MIKE [4], Components of Expertise [102], GDM [111] and VITAL [36].

Other approaches propose principled or even semi-automatic approaches to PSM development. One can for example start with specifying the global required competence of the problem-solving method and then step-by-step refine this competence description into an operational problem solver [122]. Another approach views the construction process of PSMs as a specific type of a configuration problem [110] and applies a well-known problem-solving method to solve this problem: propose-critique-modify. Coming up with PSMs is one thing, but coming up with correct PSMs is another (a PSMs is correct if it actually provides what is specified in its competence). Formal methods are applied to develop such correct PSMs [89, 45].

More recently, an encompassing framework has emerged, based on a unified view of current PSM approaches [44]. According to the framework, each PSM can be characterized along three independent dimensions: the domain dimension, the task dimension and the problem-solving paradigm dimension. The domain dimension refers to the type and knowledge properties of the knowledge structures required to apply the PSM. For example, the Propose and Revise method requires knowledge about "fixes" to fix designs that violate constraints. This dimension also expresses how domain dependent or independent a PSM is. For instance, is the PSM formulated in terms of a particular domain, committing to a particular domain ontology (say cars or houses), or in domain independent terms like hypothesis and observations, committing only to a particular task ontology? The task dimension does the same for the dependency of a PSM with respect to a particular task. For instance, a PSM can speak about hypothesis, in which case it is task (diagnosis) dependent, or about elements and sets, in which case it is task independent. The problem-solving paradigm refers to the algorithmic structure of the PSM, such as generate-and-test, or local search. Such a paradigm fixes some basic data structures, provides an initial task-subtask decomposition and a generic control regime. A move in the resulting 3D-space means a corresponding change in the problem-solving method. Notice that this framework does not mean that all tasks are derivable from each other. That

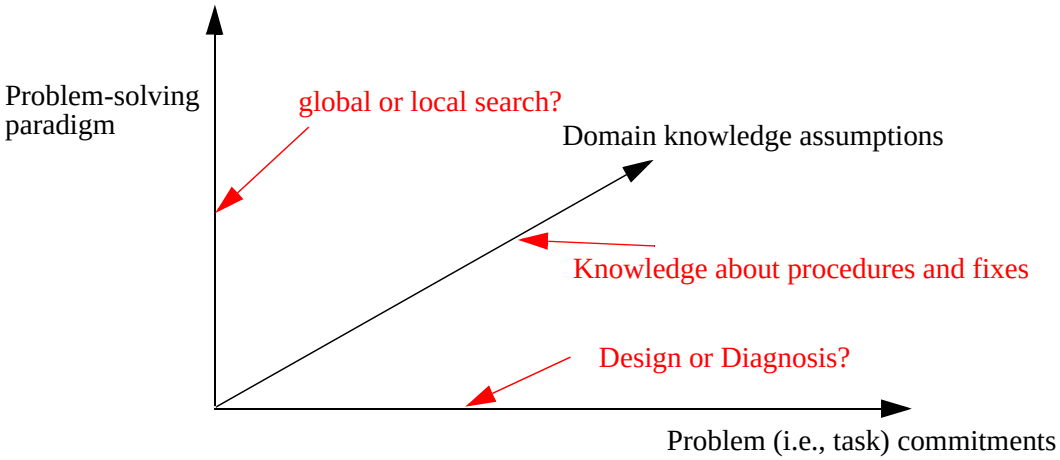

Figure 5: Three dimensional space to characterize problem-solving methods (taken from [44]).

is, we can probably not derive design from diagnosis, but we can derive parametric design from design. The framework nicely captures and integrates previous diverging approaches in a unified way. The framework is part of the so-called "Unified Problem-Solving Method Development Language" (UPML).

Reuse and libraries of PSMs When PSMs have been successfully developed for a particular application, it is worthwhile to formulate the PSMs at a generic level. That is, the reusable parts of the PSMs are identified and stored in a repository or a library. When building a new application, this library can then be consulted, preventing the system engineer from developing a complete new system from scratch. Generally, reuse of PSMs includes the following questions: which generic PSMs exist and how should a library of these methods be organized? How can PSMs be indexed in a way to support their selection for a given application? How can we support the process of adapting a generic PSM to the specific circumstances of a given application? How can individual PSMs from a library be configured into a coherent problem solver? PSM libraries are of central importance if our aim is to reuse as much as possible in a correct way.

Current work in the PSM area focuses on method-description languages such as UPML [41, 54]. Problem-solving methods that reside in libraries can be annotated with such languages, so that they become more accessible to others (people and software agents).

3.3 Libraries of PSMs

PSMs represent a kind of best practice in KBS construction. Instead of that knowledge engineers have to construct problem solvers from scratch, they can benefit from previous successful experiences of other developers. The use of best-practice components has as benefits that they reflect years of experience, enabling thorough validation and verification of the components, which enhances the quality of the software. In this sense, PSMs bear similarity to so-called *Design Patterns* in object-oriented approaches [52]. Design patterns embody re-occurring patterns in object-oriented design of software systems. These patterns are catalogued by describing them with common characteristics such as (among others) the *pattern name*, the *problem*, the *solution* and the *consequences*. These characteristics are reminiscent of the features used to describe PSMs for their selection and application. For instance, the *consequences* characteristic describes the results and trade-offs of applying the pattern. One pattern is of particular interest in the context of PSMs, namely the Strategy Pattern, which defines a family of algorithms.

Once we have a collection of such reasoning patterns, interesting issues arise such as how to structure and organize the collection and how to index the components.

3.3.1 Types of PSM libraries

Currently, there exist several libraries with PSMs. They all aim at facilitating the knowledge-engineering process, yet they differ in various ways. In particular, libraries differ along dimensions such as generality, formality, granularity and size.

- The generality dimension describes whether PSMs in a library are developed for a particular task. Task-specific libraries contain PSMs that are specialized in solving (parts of) a specific task such as diagnosis or design. Their "task-specificness" resides mainly in the terminology in which the PSMs are formulated. Examples include libraries for design [29, 84], assessment [118], diagnosis [11] and planning [7, 6]. The CommonKADS library can be viewed as an extensive collection of task-specific PSMs [26]. Task-independent libraries provide problem-solving methods that are not formulated in task-specific terminology [1].

- The formality dimension divides the libraries in informal, formal and implemented ones. Implemented libraries provide operational specifications of PSMs, which are directly executable [90, 55]. Formal libraries allow for formal verification of properties of PSMs [1, 2, 13, 109]. Finally, informal libraries provide structured textual representations of PSMs. Note that within the informal approaches, PSM descriptions can vary from just textual descriptions [29], to highly structured descriptions using diagrams [11].

- The granularity dimension distinguishes between libraries with complex components, in the sense that the PSMs realize a complete task [84], and libraries with fine-grained PSMs that perform a small part of the task. Several libraries contain both large and small building-blocks where the former are built up from the latter [11, 29, 7].

- The size dimension. The most comprehensive general library is the CommonKADS library [26] which contains PSMs for diagnosis, prediction of behavior, assessment, design, planning, assignment and scheduling and engineering modeling. The most extensive library for diagnosis [11] contains 38 PSMs for realizing 14 tasks related to diagnosis. The library for parametric design [84] consists of components from which many different PSMs can be configured. The design library of [29] mentions about 15 PSMs.

The type of a library is determined by its characterization in terms of the above dimensions. Each type has a specific role in the knowledge engineering process and has strong and weak points. The more general (i.e. task-neutral) PSMs in a library are, the more reusable they are, because they do not make any commitment to particular tasks. However, at the same time, applying such a PSM in a particular application requires considerable refinement and adaptation. This phenomenon is known as the reusability–usability trade-off [75]. Recently, research has been conducted to overcome this dichotomy by introducing adapters that gradually adapt task-neutral PSMs to task-specific ones [43] and by semi-automatically constructing the mappings between task-neutral PSMs and domain knowledge [19].

Libraries with informal PSMs provide above all support for the conceptual specification phase of the KBS, that is, they help significantly in constructing the reasoning part of the expertise model of a KBS [95]. Because such PSMs are informal, they are relatively easy to understand and malleable to fit a particular application. The disadvantage is – not surprisingly – that still much work has to be done before arriving at an implemented system. Libraries with formal PSMs are particularly important if the PSMs need to have some guaranteed properties, e.g. for use in safety-critical systems such as nuclear power plants. Their disadvantage is that they are hard to understand for humans [23] and limit the expressiveness of the knowledge engineer. Apart from the possibility to prove properties, formal PSMs have the additional advantage of being a step closer to an implemented system. Libraries with implemented PSMs allow the construction of fully operational systems. The other side of the coin is, however, that the probability that operational PSMs exactly match the requirements of the knowledge engineer, is lower.

Developing a KBS using libraries with coarse-grained PSMs, amounts to selecting the most suitable PSM and then adapt it to the particular needs of the application [84]. The advantage is that this process is quite simple as it involves only one component. The disadvantage is, however, that it is unlikely that such a library will have broad coverage, since each application might need a different (coarse-grained) PSM. The

alternative approach is to have a library with fine-grained PSMs, which are then combined together (i.e. configured) into a reasoner, either manually [91] or automatically [12, 6].

3.3.2 Organization of libraries

There are several alternatives for organizing a library and each of them has consequences for indexing PSMs and for their selection. Finding the "best" organization principle for such libraries is still an issue of debate. In the following, we will present some organization principles.

Libraries can be organized, based on the functionality of PSMs, in which case PSMs with similar functionality are stored together. In addition, the functionality of PSMs can be configured from pre-established parameters and values [109].

Another criterion to structure libraries of PSMs is based on assumptions, which specify under what conditions PSMs can be applied. Assumptions can refer to domain knowledge (e.g. a certain PSM needs a causal domain model) or to task knowledge (a certain PSM generates locally optimal solutions). To our knowledge, there does not exist a library organized following this principle, but work is currently being performed to shed more light on the role of assumptions in libraries for knowledge engineering [16, 39, 43].

Several researchers propose to organize libraries as a *task–method decomposition* structure [30, 91, 103, 124], and some available libraries are organized in this way [11, 24, 7, 84]. According to this organization structure, a task can be realized by several PSMs, each consisting of primitive and/or composite subtasks. Composite subtasks can again be realized by alternative methods, etc. Principles for library design according to this principle are discussed in [88, 87]. In a library organized according to the task-method principle, PSMs are indexed, based on two factors: (1) on the competence of the PSMs – which specifies what a PSM *can* achieve, and (2) on their assumptions – which specify the assumptions under which the PSM can be applied correctly, such as its requirements on domain knowledge. Selection of PSMs from such libraries first considers the competence of PSMs (selecting those whose competences match the task at hand), and then the assumptions of PSMs (selecting those whose assumptions are satisfied).

A last proposal to organize libraries of PSMs is based on a *suite* of so-called problem types (or tasks, for the purpose of this article tasks and problem types are treated as synonyms) [24, 25]. The suite describes problem types according to the way that problems *depend* on each other. The solution to one problem forms the input to another problem. For example, the output of a prediction task is a certain state, which can form the input to a monitoring task that tries to detect problems, which on their turn can be the input to a diagnosis task. It turns out that these problem dependencies recur in many different tasks. According to this principle, PSMs are stored under the problem type they can solve. Selection of PSMs in such a library would first identify the problem type involved (or task), and then look at the respective PSMs for this task.

3.4 Industrial applications

Building KBSs from reusable components in an academic setting is one thing. Doing the same for real industrial applications is another. So far, several industrial applications have been built, but only a few have been reported in the literature. Unilever reports on the successful use of a library with diagnostic problem-solving methods for building a knowledge-based system for diagnosing chemical production processes [100, 101]. A road traffic management knowledge-based system [83] is operational in the cities of Madrid and Barcelona in Spain. IBM, Japan reports a knowledge system for job scheduling of production processes [71]. The system has been built by using a domain-oriented library of scheduling problem-solving methods. Metrics show that a significant percentage of existing code has been reused in the new application. Knowledge-based systems for plant classification, service support for printing machines, and rheumatology have been developed from reusable methods, as reported in [92].

4 Discussion and conclusions

In this paper, we reviewed recent work in the area of ontologies [108, 117, 120, 69] and problem-solving methods [15]. The current state-of-the-art is that there is now a fairly good understanding of what ontologies and PSMs are and what they do.

One of the trends that is manifest in research on PSMs, is that the component-based view is becoming more and more dominant (as it happened in the OO field). Were early approaches concerned with developing specific applications and corresponding methodologies and tools, current approaches are more oriented towards configuring new systems by combining existing components. Issues that are inherent to such composition-approach include *(i)* the definition of proper interfaces to components through which they can communicate with each other, and *(ii)* the development of connectors between components in case their interfaces are incompatible (bridges). This area of knowledge engineering can learn a great deal from the Software Architectures area of Software Engineering [98].

Another conclusion that emerges from our analysis is that previous competing PSM approaches can be unified in one encompassing framework: the Unified Problem-Solving Method Development Language (UPML). Through this framework the issues of the domain- and task-specifity of PSMs and of their task-neutrality more or less disappear. Each PSM can unambiguously be characterized in the three dimensional framework of UPML [44].

Concerning ontologies, we see a division in two types: lightweight ontologies and heavyweight ontologies. Heavyweight ontologies are those traditionally developed in the Knowledge Representation area. "Heavy" in this sense refers to aspects such as included axioms in the ontology, associated inference mechanisms to equip ontologies with deductive power (e.g. inheritance) and the degree of formality of the ontology (e.g. underlying formal semantics). Lightweight ontologies refer to ontologies lacking these aspects, but that do define a vocabulary of terms and some specification of their meaning [114]. An extreme example of such an ontology is a list of standard/shared terms in a particular domain along with their definition in natural language. Internet directories (Yahoo, Excite, Netscape, etc.) are other examples of lightweight ontologies. Many of these directories are not even pure taxonomies, but rather a mixture of taxonomic, part-of and vague "related-to" relations. Another area where lightweight ontologies have come to play a prominent role is electronic commerce, where standard vocabularies are defined for particular branches (e.g. automobile industry). Such ontologies consist for example of XML tag sets (e.g. RosettaNet.org). It is worthwhile to remark that especially lightweight ontologies have exported the notion of ontology from the AI community to larger, mainstream communities. In order to increase the popularity of the "real" (heavyweight) ontologies, adhering to widely recognized standards (e.g. XML, RDF, etc.) is probably essential (see also efforts on OIL [72]. This does not necessarily imply that these ontologies should be constructed in the standards, rather, they should have translators from more powerful KR languages into the mainstream standards.

Another issue that we see in the areas of PSMs and ontologies is the following. In the PSM field, efforts are being made to characterize PSMs in terms of their capabilities and requirements. This makes PSMs accessible for software programs (i.e. brokers) that can find them and configure them for specific users. The European IBROW project (`http://www.swi.psy.uva.nl/projects/ibrow/home.html`) aims at building a brokering service that can configure knowledge system out of reusable PSMs that reside in libraries on the Internet. For ontologies, such efforts have received less attention, and it is therefore not easy to characterize the competence of ontologies such that software agents can localize and select them, but see [5], which discusses a web-based broker to find ontologies.

Finally, looking back at the history of Knowledge Engineering, expert systems started out as rule-based systems where domain and reasoning knowledge where tightly integrated in the rules. This strongly limited reuse in the sense that for related applications, but different domains, a complete new rule set had to be developed. As a reaction to this, a movement started to separate the domain specific knowledge from generic inference patterns that could be applied to a variety of domains. Typical examples of this approach include Generic Tasks [28] and KADS [123], which tried to identify generic reusable components. The interaction problem [27] showed however that separating domain and reasoning knowledge was not so trivial as hoped. It turned out that the purpose (task) for which a knowledge base was constructed, has a significant influence on the nature and structure of the knowledge base. This meant a step backwards in the development of generic reusable components. The next step introduced so-called *assumptions* that explicitly capture the interaction between problem solving and domain knowledge [17, 40]. A PSM could only be applied to a particular domain knowledge base if its assumptions were respected by that knowledge base. In this sense, assumptions captured -in a domain-independent way- how domain and reasoning knowledge depend on each other. Assumptions made reuse to some extent more methodological and less ad-hoc. Another problem with reuse is called the "reusability-usability" trade-off [75], which says that the

more generic a PSM is, the more reusable it is because it can be applied in many different situations, but -on the other hand- the less usable it is, because significant work must be performed before the generic PSM can be applied to a specific domain. With the birth of UPML, we hope that this trade-off has been remedied: generic components can be extended with adapters that turn generic components into more specific ones, according to one of UPML's three dimensions.

UPML enables a methodological integration of PSMs and ontologies. Although currently, ontologies and PSMs are not investigated in an integrated manner, we expect that, in the near future, this will take place. Ontology people are excited by the vision of having shared and common understanding of domains, both for people and machines. However, once ontologies are in place, and communication is optimal, a new need will arise: namely, the need to provide services on top of the ontologies. It is our vision that these services will be provided by problem-solving methods. For instance. a PSM service that automatically checks the ontology's consistency, or indexing PSMs that use ontologies to automatically index and cross-reference large databases with documents. Such PSMs will provide added value, making ontologies even more valuable than they already are.

One could argue that if we envision such a tight integration between PSMs and ontologies, aren't we then back where we started 15 years ago: namely, systems where domain knowledge and reasoning knowledge cannot be separated? In our opinion the answer is negative: we are now in the situation that we are able to configure customizable services for users (consisting of ontologies and PSMs). So, we exploit the power of component-based software synthesis, while retaining the strength of coupling domain and reasoning knowledge.

5 Relevant links

For an extensive collection of (alphabetically ordered) links to work on ontologies and problem-solving methods, including proceedings and events, see:
`http://www.cs.utexas.edu/users/mfkb/related.html`. The homepage of the PSM mailing list can be accessed at:
`http://www.swi.psy.uva.nl/mailing-lists/kaw-psm/home.html`.

A list of relevant workshops that are accessible on the WWW is included below:

- Applications of Ontologies and Problem-Solving Methods (ECAI'2000),
  `http://delicias.dia.fi.upm.es/WORKSHOP/ECAI00/index.html`
- Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends (IJCAI'99),
  `http://www.swi.psy.uva.nl/usr/richard/workshops/ijcai99/home.html`
- Applications of Ontologies and Problem-Solving Methods, ECAI'98 (European Conference on AI),
  `http://delicias.dia.fi.upm.es/WORKSHOP/ECAI98/index.html`
- Building, Maintaining, and Using Organizational Memories, ECAI'98,
  `http://www.aifb.uni-karlsruhe.de/WBS/ECAI98OM/`
- Formal Ontologies in Information Systems (FOIS'98),
  `http://krr.irst.itc.it:1024/fois98/program.html`
- Intelligent Information Integration, ECAI'98,
  `http://www.tzi.de/grp/i3/ws-ecai98/`
- Sharable and Reusable Components for Knowledge Systems, KAW'98 (Workshop on Knowledge Acquisition, Modeling, and Management),
  `http://ksi.cpsc.ucalgary.ca/KAW/KAW98/KAW98Proc.html`
- Ontological Engineering, AAAI Spring Symp. Series, Stanford, Calif., 1997,
  `http://www.aaai.org/Symposia/Spring/1997/sss-97.html`
- Problem-Solving Methods, IJCAI'97 (Int'l Joint Conf. AI),
  `http://www.aifb.uni-karlsruhe.de/WBS/dfe/PSM/main.html`

- Ontological Engineering, ECAI'96,
  `http://wwwis.cs.utwente.nl:8080/kbs/EcaiWorkshop/homepage.html`
- Sharable and Reusable Ontologies, KAW'96,
  `http://ksi.cpsc.ucalgary.ca/KAW/KAW96/KAW96Proc.html`
- Sharable and Reusable Problem-Solving Methods, KAW'96,
  `http://ksi.cpsc.ucalgary.ca/KAW/KAW96/KAW96Proc.html`

References

[1] M. Aben. Formally specifying re-usable knowledge model components. *Knowledge Acquisition*, 5:119–141, 1993.

[2] M. Aben. *Formal Methods in Knowledge Engineering*. PhD thesis, University of Amsterdam, Amsterdam, 1995.

[3] G. Aguado, J. Bateman, A. Bañon, S. Bernardos, M. Fernändez, A. Gómez-Pérez, E. Nieto, A. Olalla, R. Plaza, and A. Sanchez. ONTOGENERATION: Reusing domain and linguistic ontologies for spanish text generation. In A. Gomez-Perez and V. R. Benjamins, editors, *Proceedings of the Workshop on Applications of Ontologies and Problem-Solving Methods, held in conjunction with ECAI-98*, pages 1–10, Brighton, UK, August 1998. ECAI. http://delicias.dia.fi.upm.es/WORKSHOP/ECAI98/schedule.html.

[4] J. Angele, D. Fensel, and R. Studer. Developing knowledge-based systems with mike. *Journal of Automated Software Engineering*, to appear, 1998.

[5] J. C. Arpirez-Vega, A. Gomez-Perez, A. Lozano-Tello, and H. Sofia Pinto. $(ONTO)^2$Agent: an ontology-based WWW broker to select ontologies. In A. Gomez Perez and V. R. Benjamins, editors, *Proc. of the workshop on Applications of Ontologies and Problem-Solving Methods of the 13th ECAI*, pages 16–24. ECAI-98, 1998.

[6] Barros, L. Nunes de, J. Hendler, and V. R. Benjamins. Par-KAP: a knowledge acquisition tool for building practical planning systems. In M. E. Pollack, editor, *Proc. of the 15th IJCAI*, pages 1246–1251, Japan, 1997. International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers, Inc. Also published in Proceedings of the Ninth Dutch Conference on Artificial Intelligence, NAIC'97, K. van Marcke, W. Daelemans (eds), University of Antwerp, Belgium, pages 137–148.

[7] Barros, L. Nunes de, A. Valente, and V. R. Benjamins. Modeling planning tasks. In *Third International Conference on Artificial Intelligence Planning Systems, AIPS-96*, pages 11–18. American Association of Artificial Intelligence (AAAI), 1996.

[8] J. A. Bateman. KPML: The KOMET-Penman (multilingual) development environment. Technical report, GMD/IPSI, Darmstadt (Germany), 1994.

[9] J. A. Bateman, R. T. Kasper, J. D. Moore, and R. A. Whitney. A general organization of knowledge for natural language processing: the penman upper model. Technical report, USC/ISI, Marina del Rey, CA (USA), 1990.

[10] J. A. Bateman, B. Magnini, and G. Fabris. The generalized upper model knowledge bare: Organization and use. In N. J. I. Mars, editor, *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing.*, pages 60–72. IOS Press, Amsterdam, NL, 1995.

[11] V. R. Benjamins. *Problem Solving Methods for Diagnosis*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, 1993.

[12] V. R. Benjamins. Problem-solving methods for diagnosis and their role in knowledge acquisition. *International Journal of Expert Systems: Research and Applications*, 8(2):93–120, 1995.

[13] V. R. Benjamins and M. Aben. Structure-preserving KBS development through reusable libraries: a case-study in diagnosis. *International Journal of Human-Computer Studies*, 47:259–288, 1997.

[14] V. R. Benjamins and D. Fensel. Community is knowledge! in $(KA)^2$. In B. R. Gaines and M. A. Musen, editors, *Proceedings of the 11th Banff Workshop on Knowledge Acquisition, Modeling and Management (KAW'98)*, pages KM–2–1–KM–2–18, Alberta, Canada, 1998. SRDG Publications, University of Calgary. http://ksi.cpsc.ucalgary.ca/KAW/KAW98/KAW98Proc.html, [also in Proceedings of KEML-98, Karlsruhe, Germany].

[15] V. R. Benjamins and D. Fensel. Editorial: Problem-solving methods. *International Journal of Human-Computer Studies*, 49(4):305–313, October 1998. Special issue on Problem-Solving Methods.

[16] V. R. Benjamins, D. Fensel, and R. Straatman. Assumptions of problem-solving methods and their role in knowledge engineering. In W. Wahlster, editor, *Proc. ECAI–96*, pages 408–412. J. Wiley & Sons, Ltd., 1996.

[17] V. R. Benjamins and C. Pierret-Golbreich. Assumptions of problem-solving methods. In N. Shadbolt, K. O'Hara, and G. Schreiber, editors, *Lecture Notes in Artificial Intelligence, 1076, 9th European Knowledge Acquisition Workshop, EKAW-96*, pages 1–16, Berlin, 1996. Springer-Verlag.

[18] A. Bernaras, I. Laresgoiti, and J. Corera. Building and reusing ontologies for electrical network applications. In *Proceedings of the 12th ECAI*, pages 298–302, 1996.

[19] P. Beys, V. R. Benjamins, and G. van Heijst. Remedying the reusability-usability tradeoff for problem-solving methods. In B. R. Gaines and M. A. Musen, editors, *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 2.1–2.20, Alberta, Canada, 1996. SRDG Publications, University of Calgary. http://ksi.cpsc.ucalgary.ca:80/KAW/KAW96/KAW96Proc.html.

[20] M. Blázquez, M. Fernández, J. M. García-Pinar, and A. Gómez-Pérez. Building ontologies at the knowledge level using the ontology design environment. In *Proceedings of the Eleventh Workshop on Knowledge Acquisition, Modeling and Management, KAW'98*, Banff, Canada, 1998.

[21] S. Borgo, N. Guarino, and C. Masolo. Stratified ontologies: the case of physical objects. In *Proceedings of the Workshop on Ontological Engineering, held in conjunction with ECAI-96*, pages 5–15, Budapest, August 1996. ECAI.

[22] W. N. Borst. *Construction of Engineering Ontologies*. PhD thesis, University of Twente, Enschede, 1997.

[23] J. P. Bowen and M. G. Hinchey. Ten commands of formal methods. *IEEE Computer*, 28(4):56–63, 1995.

[24] J. Breuker. Components of problem solving and types of problems. In L. Steels, A. T. Schreiber, and W. van de Velde, editors, *Lecture Notes in Artificial Intelligence, 867, 8th European Knowledge Acquisition Workshop, EKAW-94*, pages 118–136, Berlin, Germany, 1994. Springer-Verlag.

[25] J. Breuker. A suite of problem types. In J. Breuker and W. van de Velde, editors, *CommonKADSLibrary for Expertise Modeling*, pages 57–87. IOS Press, Amsterdam, The Netherlands, 1994.

[26] J. Breuker and W. van de Velde, editors. *CommonKADS Library for Expertise Modeling*. IOS Press, Amsterdam, The Netherlands, 1994.

[27] T. Bylander and B. Chandrasekaran. Generic tasks in knowledge-based reasoning: The right level of abstraction for knowledge acquisition. In B. Gaines and J. Boose, editors, *Knowledge Acquisition for Knowledge Based Systems*, volume 1, pages 65–77. Academic Press, London, 1988.

[28] B. Chandrasekaran. Generic tasks in expert system design and their role in explanation of problem solving. In *Proceedings of the National Academy of Science/Office of Naval Research Workshop on AI and Distributed Problem Solving*, Washington, DC., 1985. National Academy of Sciences.

[29] B. Chandrasekaran. Design problem solving: A task analysis. *AI Magazine*, 11:59–71, 1990.

[30] B. Chandrasekaran, T. R. Johnson, and J. W. Smith. Task-structure analysis for knowledge modeling. *Communications of the ACM*, 35(9):124–137, 1992.

[31] B. Chandrasekaran, J.R. Josephson, and V.R. Benjamins. Ontologies: What are they? why do we need them? *IEEE Intelligent Systems and Their Applications*, 14(1):20–26, 1999. Special Issue on Ontologies.

[32] V. Chaudhri, A. Farquhar, R. Fikes, P. Karp, and J. Rice. Okbc: A programmatic foundation for knowledge base interoperability. In *Proc. AAAI'98 Conference, Madison, WI, July 1998*. AAAI Press, 1998.

[33] V. K. Chaudhri, A. Farquhar, R. Fikes, P. Karp, and J. Rice. The generic frame protocol 2.0. Technical report, Stanford, 1997.

[34] W. J. Clancey. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985.

[35] J. Domingue. Tadzebao and webonto: Discussing, browsing, and editing ontologies on the web. In *Proceedings of the Eleventh Workshop on Knowledge Acquisition, Modeling and Management, KAW'98*, Banff, Canada, 1998.

[36] J. Domingue, E. Motta, and S. Watt. The emerging VITAL workbench. In Aussenac et al., editor, *EKAW'93 Knowledge Acquisition for Knowledge-Based Systems. Lecture Notes in Artificial Intelligence, LNCS 723*, Berlin, Germany, 1993. Springer-Verlag.

[37] A. J. Duineveld, R. Stoter, M. R. Weiden, B. Kenepa, and V. R. Benjamins. WonderTools? a comparative study of ontological engineering tools. *International Journal of Human-Computer Studies*, July:in press, 2000.

[38] A. Farquhar, R. Fikes, and J. Rice. The ontolingua server: a tool for collaborative ontology construction. *International Journal of Human-Computer Studies*, 46(6):707–728, June 1997.

[39] D. Fensel and V. R. Benjamins. Assumptions in model-based diagnosis. In B. R. Gaines and M. A. Musen, editors, *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 5.1–5.18, Alberta, Canada, 1996. SRDG Publications, University of Calgary. http://ksi.cpsc.ucalgary.ca:80/KAW/KAW96/KAW96Proc.html.

[40] D. Fensel and V. R. Benjamins. The role of assumptions in knowledge engineering. *International Journal on Intelligent Systems (IJIS)*, 13(7):715–748, 1998.

[41] D. Fensel, V. R. Benjamins, E. Motta, and B. Wielinga. UPML: A framework for knowledge system reuse. In *Proceedings of the 16th International Joint Conference on AI (IJCAI-99)*, pages 16–21, Sweden, 1999.

[42] D. Fensel, S. Decker, M. Erdmann, and R. Studer. Ontobroker: The very high idea. In *Proceedings of the 11th International Flairs Conference (FLAIRS-98)*, Sanibal Island, Florida, 1998.

[43] D. Fensel and R. Groenboom. Specifying knowledge-based systems with reusable components. In *Proceedings 9th Int. Conference on Software Engineering and Knowledge Engineering SEKE'97*, pages 349–357, Madrid, 1997.

[44] D. Fensel and Enrico Motta. Structured development of problem solving methods. *IEEE Transactions on Knowledge and Data Engineering*, page to appear, 2000.

[45] D. Fensel and A. Schönegge. Using KIV to specify and verify architecturs of knowlege-based systems. In *Proc. of 12th IEEE International Conference on Automated Software Engineering (ASEC-97)*. IEEE, 1997.

[46] D. Fensel, A. Schönegge, R. Groenboom, and B. Wielinga. Specification and verification of knowledge-based systems. In *Proc. Validation and Verification workshop at ECAI–96*, 1996.

[47] D. Fensel and R. Straatman. The essence of problem-solving methods: making assumptions to gain efficiency. *IJHCS*, 48:181–215, 1998.

[48] M. Fernandez, A. Gómez Pérez, and N. Juristo. METHONTOLOGY: From ontological art towards ontological engineering. In *Spring Symposium Series on Ontological Engineering*, Stanford, 1997. AAAI Press.

[49] M. Fernandez, A. Gomez-Perez, J. Pazos, and Alejandro Pazos. Building a chemical ontology using methontology and the ontology design environment. *IEEE Intelligent Systems and Their Applications*, 14(1):37–46, January/February 1999.

[50] M. Fernandez-Lopez. Overview of methodologies for building ontologies. In V. R. Benjamins, B. Chandrasekaran, A. Gomez Perez, N. Guarino, and M. Uschold, editors, *Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods*, pages 4.1–4.12, Sweden, 1999.

[51] M. Frohlich and R. P. van de Riet. Using multiple ontologies in a framework for natural language generation. In A. Gomez-Perez and V. R. Benjamins, editors, *Proceedings of the Workshop on Applications of Ontologies and Problem-Solving Methods, held in conjunction with ECAI-98*, pages 67–77, Brighton, UK, August 1998. ECAI. http://delicias.dia.fi.upm.es/WORKSHOP/ECAI98/schedule.html.

[52] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[53] M. R. Genesereth and R. E. Fikes. Knowledge interchange format, version 3.0, reference manual. Technical report, Logic-92-1, Computer Science Dept., Stanford University, 1992. http://www.cs.umbc.edu/kse/.

[54] J. H. Gennari, W. Grosso, and M. Musen. A method-description language: An initial ontology with examples. In B. R. Gaines and M. A. Musen, editors, *Proceedings of the 11th Banff Workshop on Knowledge Acquisition, Modeling and Management (KAW'98)*, pages SHARE.11.–SHARE.11.18, Alberta, Canada, 1998. SRDG Publications, University of Calgary. http://ksi.cpsc.ucalgary.ca/KAW/KAW98/KAW98Proc.html.

[55] J. H. Gennari, S. W. Tu, T. E. Rotenfluh, and M. A. Musen. Mapping domains to methods in support of reuse. *International Journal of Human-Computer Studies*, 41:399–424, 1994.

[56] A. Gomez Perez. A framework to verify knowledge sharing technology. *Expert Systems with Application*, 11:519–529, 1996.

[57] A. Gómez-Pérez. Knowledge sharing and reuse. In J. Liebowitz, editor, *The Handbook of Applied Expert Systems*. CRC, 1998.

[58] A. Gomez Perez. Evaluation of taxonomic knowledge in ontologies and knowledge bases. In B. R. Gaines, R. Kremer, and M. A. Musen, editors, *Proceedings of the 12th Banff Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*, pages 6.8.1–6.8.18, Alberta, Canada, 1999. SRDG Publications, University of Calgary. http://sern.ucalgary.ca/KSI/KAW/KAW99/papers.html.

[59] A. Gomez-Perez, A. Fernandez, and M. De Vicente. Towards a method to conceptualize domain ontologies. In *Working notes of the workshop on Ontological Engineering, ECAI'96*, pages 41–52, Budapest, 1996. ECCAI.

[60] A. Gomez-Perez and M. D. Rojas. Ontological reengineering for reuse. In D. Fensel and R. Studer, editors, *Proceedings of the EKAW-99*, page to appear. Springer-Verlag, 1999.

[61] T. Gruber and R. Olsen. An ontology for engineering mathematics. Technical report, Knowledge Systems Laboratory, Stanford University, CA, 1994.

[62] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

[63] T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43:907–928, 1995.

[64] M. Grüninger and M. Fox. Methodology for the design and evaluation of ontologies. In *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing held in conjunction with IJCAI-95*, Montreal, Canada, 1995.

[65] N. Guarino. Some organizing principles for a unified top-level ontology. In *Spring Symposium Series on Ontological Engineering*, pages 57–63, Stanford, 1997. AAAI Press.

[66] N. Guarino. Some ontological principles for designing upper level lexical resources. In *Proceedings of the First International Conference on Language Resources and Evaluation*, pages –, Granada, 1998.

[67] N. Guarino and P. Giaretta. Ontologies and knowledge bases: Towards a terminological clarification. In N. J. I. Mars, editor, *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing*, pages 25–32. IOS Press, Amsterdam, NL, 1995.

[68] N. Guarino, C. Masolo, and G. Vetere. Ontoseek: Using large linguistic ontologies for accessing on-line yellow pages and product catalogs. *IEEE Intelligent Systems and Their Applications*, page to appear, 1999.

[69] N. Guarino and R. Poli. The role of ontology in the information technology. *International Journal of Human-Computer Studies*, 43(5/6):623–965, 1995. Special issue on ontology.

[70] E. Hoenkamp. Spotting ontological lacunae through spectrum analysis of retrieved documents. In A. Gomez-Perez and V. R. Benjamins, editors, *Proceedings of the Workshop on Applications of Ontologies and Problem-Solving Methods, held in conjunction with ECAI-98*, pages 73–77, Brighton, UK, August 1998. ECAI. http://delicias.dia.fi.upm.es/WORKSHOP/ECAI98/schedule.html.

[71] M. Hori and T. Yoshida. domain-oriented library of scheduling methods: design principal and real-life application. *International Journal of Human-Computer Studies*, 49(4):601–626, 1998. Special issue on Problem-Solving Methods.

[72] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. van Harmelen, M. Klein, S. Staab, and R. Studer. The ontology inference layer oil. In *draft*, 2000.

[73] G. Ist. Generating instructional text. final report. Technical report, IRST, Trento (Italy), 1996.

[74] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 1995.

[75] G. Klinker, C. Bhola, G. Dallemagne, D. Marques, and J. McDermott. Usable and reusable programming constructs. *Knowledge Acquisition*, 3:117–136, 1991.

[76] K. Knight and S. Luk. Building a large knowledge base for machine translation. In *AAAI–94*, 1994.

[77] D. B. Lenat and R. V. Guha. *Building large knowledge-based systems. Representation and inference in the Cyc project*. Addison-Wesley, Reading, Massachusetts, 1990.

[78] R. MacGregor. Inside the LOOM classifier. *SIGART Bulletin*, 2(3):70–76, June 1991.

[79] K. Mahesh. Ontology development for machine translation: Ideology and methodology. Technical report, Computer Research Laboratory, New Mexico State University, 1996.

[80] S. Marcus, editor. *Automating knowledge acquisition for expert systems*. Kluwer, Boston, 1988.

[81] G. A. Miller. WORDNET: an online lexical database. *International Journal of Lexicography*, 3(4):235–312, 1990.

[82] R. Mizoguchi, J. Vanwelkenhuysen, and M. Ikeda. Task ontology for reuse of problem solving knowledge. In N. J. I. Mars, editor, *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing.*, pages 46–57. IOS Press, Amsterdam, NL, 1995.

[83] M. Molina, J. Hernández, and J. Cuena. A structure of problem-solving methods for real-time decision support in traffic control. *International Journal of Human-Computer Studies*, 49(4):577–600, 1998. Special issue on Problem-Solving Methods.

[84] E. Motta and Z. Zdrahal. A library of problem-solving components based on the integration of the search paradigm with task and method ontologies. *International Journal of Human-Computer Studies*, 49(4):437–470, 1998. Special issue on Problem-Solving Methods.

[85] M. Musen. An overview of knowledge acquisition. In J. M. David, J. P. Krivine, and R. Simmons, editors, *Second Generation Expert Systems*. Springer Verlag, 1993.

[86] R. Neches, R. E. Fikes, T. Finin, T. R. Gruber, T. Senator, and W. R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, 1991.

[87] K. Orsvärn. *Knowledge Modelling with Libraries of Task Decomposition Methods*. PhD thesis, Swedish Institute of Computer Science, 1996.

[88] K. Orsvärn. Principles for libraries of task decomposition methods – conclusions from a case-study. In N. Shadbolt, K. O'Hara, and G. Schreiber, editors, *Lecture Notes in Artificial Intelligence, 1076, 9th European Knowledge Acquisition Workshop, EKAW-96*, pages 48–65. Springer-Verlag, 1996.

[89] C. Pierret-Golbreich. Supporting organization and use of problem-solving methods libraries by formal methods. *International Journal of Human-Computer Studies*, 49(4):471–495, 1998. Special issue on Problem-Solving Methods.

[90] A. Puerta, S. W. Tu, and M. A. Musen. Modeling tasks with mechanisms. In *Workshop on Problem-Solving Methods*, Stanford, July 1992. GMD, Germany.

[91] A.R. Puerta, J. Egar, S. Tu, and M. Musen. A multiple-method shell for the automatic generation of knowledge acquisition tools. *Knowledge Acquisition*, 4:171–196, 1992.

[92] F. Puppe. Knowledge reuse among diagnostic problem-solving methods in the shell-kit D3. *International Journal of Human-Computer Studies*, 49(4):627–649, 1998. Special issue on Problem-Solving Methods.

[93] D. Rosner. Generating multilingual documents from a knowledge base: The techdoc project. Technical report, FAW Ulm, Ulm (Germany), 1994.

[94] A. Th. Schreiber, J. M. Akkermans, A. A. Anjewierden, R. de Hoog, N. R. Shadbolt, W. Van de Velde, and B. J. Wielinga. *Knowledge Engineering and Management, The CommonKADS methodology*. MIT Press, 2000.

[95] A. Th. Schreiber, B. J. Wielinga, and J. A. Breuker, editors. *KADS: A Principled Approach to Knowledge-Based System Development*, volume 11 of *Knowledge-Based Systems Book Series*. Academic Press, London, 1993.

[96] A. Th. Schreiber, B. J. Wielinga, R. de Hoog, J. M. Akkermans, and W. Van de Velde. CommonKADS: A comprehensive methodology for KBS development. *IEEE Expert*, 9(6):28–37, December 1994.

[97] A. Th. Schreiber, B. J. Wielinga, and W. H. J. Jansweijer. The KACTUS view on the 'O' word. In *IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

[98] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[99] J. F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Book draft, 1997.

[100] P-H. Speel and M. Aben. Applying a library of problem solving methods on a real-life task. *International Journal of Human-Computer Studies*, 46(May):627–652, 1997.

[101] P-H. Speel and M. Aben. Preserving conceptual structures in design and implementation of industrial KBSs. *International Journal of Human-Computer Studies*, 49(4):547–575, 1998. Special issue on Problem-Solving Methods.

[102] L. Steels. Components of expertise. *AI Magazine*, 11(2):28–49, Summer 1990.

[103] L. Steels. The componential framework and its role in reusability. In Jean-Marc David, Jean-Paul Krivine, and Reid Simmons, editors, *Second Generation Expert Systems*, pages 273–298. Springer-Verlag, Berlin Heidelberg, Germany, 1993.

[104] O. Stock, G. Carenini, F. Cecconi, E. Franconi, A. Lavelli, B. Magnini, F. Pianesi, M. Ponzi, V. Samek-Lodovici, and C. Strapparava. Alfresco: Enjoying the combination of natural language processing and hypermedia for information exploration. In Mark T. Maybury, editor, *Intelligent Multimedia Interfaces*, chapter 9, pages 197–224. The MIT Press, 1993.

[105] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering, principles and methods. *Data and Knowledge Engineering*, 25(1-2):161–197, 1998.

[106] W. Swartout and Y. Gil. Expect: Explicit representations for flexible acquisition. In *Proceedings of the Ninth Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1995.

[107] W. Swartout, R. Patil, K. Knight, and T. Russ. Toward distributed use of large-scale ontologies. In *Spring Symposium Series on Ontological Engineering*, pages 33–40, Stanford, 1997. AAAI Press.

[108] W. Swartout and A. Tate. Coming to terms with ontologies. *IEEE Intelligent Systems and Their Applications*, 14(1):19–19, January/February 1999.

[109] A. ten Teije. *Automated Configuration of Problem Solving Methods in Diagnosis*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, 1997.

[110] A. ten Teije, F. van Harmelen, A. Th. Schreiber, and B. Wielinga. Construction of problem-solving methods as parametric design. *International Journal of Human-Computer Studies*, 49(4):363–389, 1998. Special issue on Problem-Solving Methods.

[111] P. Terpstra, G. van Heijst, B. Wielinga, and N. Shadtbolt. Knowledge acquisition support through generalised directive models. In Jean-Marc David, Jean-Paul Krivine, and Reid Simmons, editors, *Second Generation Expert Systems*, pages 428–455. Springer-Verlag, Berlin Heidelberg, Germany, 1993.

[112] Y. A. Tijerino and R. Mizoguchi. MULTIS II: enabling end-users to design problem-solving egines via two-level task ontologies. In Aussenac et al., editor, *EKAW'93 Knowledge Acquisition for Knowledge-Based Systems. Lecture Notes in Artificial Intelligence, LNCS 723*, pages 340–359, Berlin, Germany, 1993. Springer-Verlag.

[113] M. Uschold. Building ontologies: Towards a unified methodology. In *Expert Systems 96*, 1996.

[114] M. Uschold. Knowledge level modelling: concepts and terminology. *The Knowledge Engineering Review*, 13(1), 1998. Special Issue on Ontologies.

[115] M. Uschold and R. Jasper. A framework for understanding and classifying ontology applications. In V. R. Benjamins, B. Chandrasekaran, A. Gomez Perez, N. Guarino, and M. Uschold, editors, *Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods*, pages 11.1–11.12, Sweden, 1999.

[116] M. Uschold and M. King. Towards a methodology for building ontologies. In *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*. IJCAI, 1995.

[117] M. Uschold and A. Tate. Special issue on ontologies. *The Knowledge Engineering Review*, 13(1), 1998.

[118] A. Valente and C. Löckenhoff. Organization as guidance: A library of assessment models. In *Proceedings of the Seventh European Knowledge Acquisition Workshop (EKAW'93), Lecture Notes in Artificial Intelligence, LNCS 723*, pages 243–262, 1993.

[119] P. E. van de Vet, P.-H. Speel, and N. J. I. Mars. The plinius ontology of ceramic materials. In N. J. I. Mars, editor, *Working papers European Conference on Artificial Intelligence ECAI'94 Workshop on Implemented Ontologies*, pages 187–206, Amsterdam, 1994. ECCAI.

[120] G. van Heijst, A. T. Schreiber, and B. J. Wielinga. Using explicit ontologies in KBS development. *International Journal of Human-Computer Studies*, 46(2/3):183–292, 1997.

[121] J. Dukes-Schlossberg W. Mark and R. Kerber. Ontological commitments and domain-specific architectures: Experience with comet and cosmos. In N. J. I. Mars, editor, *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing.*, pages 33–44. IOS Press, Amsterdam, NL, 1995.

[122] B. J. Wielinga, J. M. Akkermans, and A. Th. Schreiber. A competence theory approach to problem solving method construction. *International Journal of Human-Computer Studies*, 49(4):315–338, 1998. Special issue on Problem-Solving Methods.

[123] B. J. Wielinga, A. Th. Schreiber, and J. A. Breuker. KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition*, 4(1):5–53, 1992. Special issue 'The KADS approach to knowledge engineering'. [Reprinted in: B. Buchanan & D. Wilkins (1992), *Readings in Knowledge Acquisition and Learning*, San Mateo, CA: Morgan Kaufmann, pp. 92–116].

[124] B. J. Wielinga, W. Van de Velde, A. Th. Schreiber, and J. M. Akkermans. The CommonKADS framework for knowledge modelling. In B. R. Gaines, M. A. Musen, and J. H. Boose, editors, *Proc. 7th Banff Knowledge Acquisition Workshop*, volume 2, pages 31.1–31.29. SRDG Publications, University of Calgary, Alberta, Canada, 1992.

Several problems make it difficult to reuse existing ontologies in applications: Ontologies are dispersed over several servers; the formalization differs depending on the server on which the ontology is stored; ontologies on the same server are usually described with different levels of detail; and there is no common format for presenting relevant information about the ontologies so that users can decide which ontology. best suits their purpose.Â Problem-Solving Methods. PSMs describe the reasoning process of a knowledge-based system in an implementation- and domain-independent manner. â€¦ SUBSCRIBE TODAY! Subscribe to Questia and enjoy Ontologies and problem-solving methods are promising candidates for reuse in Knowledge Engineering. Ontologies define domain knowledge at a generic level, while problem-solving methods specify generic reasoning knowledge. Both type of components can be viewed as complementary entities that can be used to configure new knowledge systems from existing, reusable components. In this survey paper, we give an overview of approaches for ontologies and problem-solving methods. View PDF. Save to Library. New ontologies improve problem solving within that domain. Translating research papers within every field is a problem made easier when experts from different countries maintain a controlled vocabulary of jargon between each of their languages.[1]. Since Google started an initiative called Knowledge Graph in 2012, a substantial amount of research has used the phrase knowledge graph as a generalized term.Â What ontologies in both information science and philosophy have in common is the attempt to represent entities, ideas and events, with all their interdependent properties and relations, according to a system of categories.