Handbook of Knowledge Representation
Edited by F. van Harmelen, V. Lifschitz and B. Porter
© 2008 Elsevier B.V. All rights reserved
DOI: 10.1016/S1574-6526(07)03025-8

Chapter 25

Knowledge Engineering

Guus Schreiber

25.1 Introduction

The discipline of knowledge engineering grew out of the early work on expert systems in the seventies. With the growing popularity of knowledge-based systems (as these were by then called), there arose also a need for a systematic approach for building such systems, similar to methodologies in main-stream software engineering. Over the years, the discipline of knowledge engineering has evolved into the development of theory, methods and tools for developing knowledge-intensive applications. In other words, it provides guidance about when and how to apply particular knowledge-presentation techniques for solving particular problems.

In this chapter we first discuss (Section 25.2) a number of principles, that have become the baseline of modern knowledge engineering. These include the common distinction made in knowledge engineering between task knowledge and domain knowledge. In Section 25.3 we explore the notion of problem-solving tasks in detail and present typical patterns and methods user for solving such tasks. In Section 25.4 we focus on the domain perspective, in particular the representation and use of ontologies. Finally, Section 25.5 summarizes the main techniques that are being used in knowledge engineering, examples of their use.

25.2 Baseline

The early expert systems were based on an architecture which separated domain knowledge, in the form a knowledge base of rules, from a general reasoning mechanism. This distinction still is still valid in knowledge engineering practice. In the early eighties a number of key papers were published that set the scene for a systematic approach to knowledge engineering.

In 1982 Newell published a paper on “The Knowledge Level” [28] in which he argued the need for a description of knowledge at a level higher the level of symbols in knowledge-representation systems. The knowledge-level was his proposal for realizing a description of an AI system in terms of its rational behavior: why does the system (the "agent") perform this "action", independent of its symbolic representation in rules, frames or logic (the "symbol" level). Descriptions at the knowledge level has since become a principle underlying knowledge engineering.

Two other key publications came from Clancey. His "Epistemology of a rule-based system" [8] can be viewed as a first knowledge-level description of a knowledge-based system, in which he distinguished various knowledge *types*. Two tears later his article "Heuristic classification" appeared [9] which described a standard problem-solving pattern in knowledge-level terms. Such patterns subsequently became an important focus of knowledge-engineering research; these patterns typically serve as reusable pieces of task knowledge. We treat these in more depth in Section 25.3.

In the nineties the attention of the knowledge-engineering shifted gradually to domain knowledge, in particular reusable representations in the form of ontologies. A key paper, which also quite wide attention outside the knowledge-engineering community was Gruber's paper on portable ontologies [16]. During this decade ontologies are getting widespread attention as vehicles for sharing concepts within a distributed community such as the web (e.g., see Chapter 21 on the Semantic Web). Similar to task knowledge, patterns also play an important role on modeling domain knowledge. In Section 25.4 we describe in some detail the main issues in ontology engineering.

25.3 Tasks and Problem-Solving Methods

In the early expert systems task knowledge was embedded in the reasoning engine and in rules in the knowledge base. The key point of Clancey's "epistemology" paper was to explicate the underlying problem-solving method. Since then, the knowledge-engineering community has developed a range of such problem-solving methods. We can define a problem-solving method as follows:

> *A* problem-solving method (*PSM*) *is a knowledge-level specification of a reasoning pattern that can used to carry out a knowledge-intensive task.*

To categorize problem-solving-methods we need a typology of knowledge-intensive tasks. Various (partial) task typologies have been reported in the literature. Stefik [41] distinguishes between "diagnosis", "classification" and "configuration". Chandrasekaran [7] has described a typology of "design tasks" (including configuration). McDermott [24] describes a taxonomy of problem types. Table 25.1 shows the typology of task types distinguished by Schreiber et al. [37].

In this section we describe two problem-solving methods in more detail: one method for configuration design and one method for assessment. In general, there does not need to be a one-to-one correspondence between methods and tasks, although in practice there often is.

25.3.1 Two Sample Problem-Solving Methods

Propose-and-revise. The propose-and-revise (P&R) method was described by Marcus and McDermott [23]. The method was used to solve a configuration-design task, namely elevator design (the so-called VT case study [22]). The data for this case

Table 25.1. Task types in CommonKADS (adapted from [37])

Task type	Input	Output	Knowledge	Features
		ANALYTIC TASK TYPES		
Classification	Object features	Object class	Feature-class associations	Set of classes is predefined.
Diagnosis	Symptoms/ complaints	Fault category	Model of system behavior	Form output varies (causal chain, state, label) and depends on use made of it (troubleshooting).
Assessment	Case description	Decision class	Criteria, norms	Assessment is performed at one particular point in time (cf. monitoring).
Monitoring	System data	Discrepancy class	Normal system behavior	System changes over time. Task is carried out repeatedly.
Prediction	System data	System state	Model of system behavior	Output state is a system description at some future point in time.
		SYNTHETIC TASK TYPES		
Design	Requirements	Artifact description	Functions, components, skeletal design, constraints, preferences	May include creative design of components.
Configuration design	Requirements	Artifact description	Functions, components, constraints, preferences	Subtype of design in which all components are predefined.
Assignment	Two object sets, requirements	Mapping set 1 $\rightarrow$ set 2	Constraints, preferences	Mapping need not be one-to-one.
Planning	Goals, requirements	Action plan	Actions, constraints, preferences	Actions are (partially) ordered in time.
Scheduling	Job activities, resources, time slots, requirements	Schedule = mapping activities $\rightarrow$ time slots of resources	Constraints, preferences	Time-oriented character distinguishes it from assignment.

study was subsequently used in a comparative study in which different knowledge-engineering approaches were used to solve this problem. The results of the study were published in special issue of Human-Computer Studies [38]. This issue provides a wealth of information for readers interested in details of modern knowledge engineering. We will come back to this study in the next section, as reuse of the pre-existing ontology was a prime focus of this study.

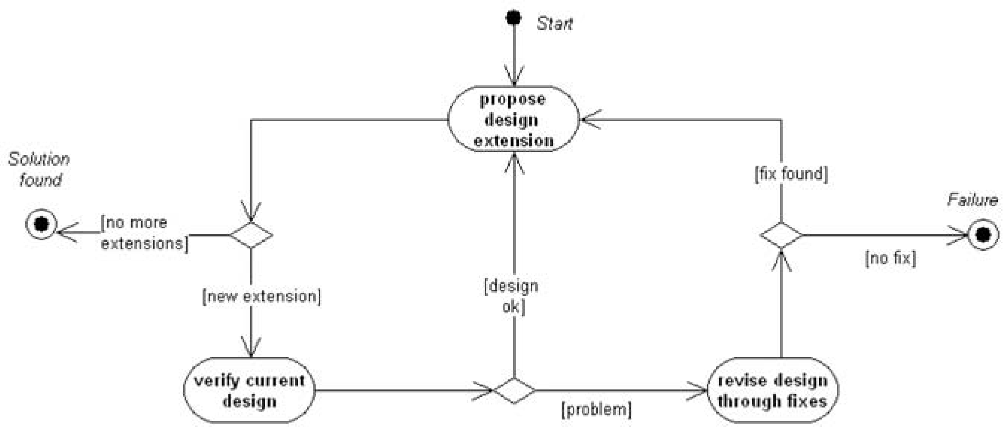

Figure 25.1: Top-level reasoning strategy of the P&R method in the form of a UML activity diagram.

Fig. 25.1 shows the top-level reasoning strategy of the P&R method. The method decomposes the configuration-design task into three subtasks:

1. Propose an extension to the existing design, e.g., add a component.
2. Verify the current design to see whether any of the constraints are violated. For example, adding a particular hoist cable might violate constraints involving the strength of the cable in comparison to other elevator components.
3. If a constraint is violated, use domain-specific revision strategies to remedy the problem, for example, upgrading the model of the hoist cable.

Unlike some other methods, which undo previous design decisions, P&R fixes them. P&R does not require an explicit description of components and their connections. Basically, the method operates on one large bag of parameters. Invocation of the propose task produces one new parameter assignment, the smallest possible extension of an existing design. Domain-specific, search-control knowledge guides the order of parameter selection, based on the components they belong to. The verification task in P&R applies a simple form of constraint evaluation. The method performs domain-specific calculations provided by the constraints. In P&R, a verification constraint has a restricted meaning, namely a formula that delivers a Boolean value. Whenever a constraint violation occurs, P&R's revision task uses a specific strategy for modifying the current design. To this end, the task requires knowledge about fixes, a second form of domain-specific, search-control knowledge. Fixes represent heuristic strategies for repairing the design and incorporate design preferences. The revision task tries to make the current design consistent with the violated constraint. It applies combinations of fix operations that change parameter values, and then propagates these changes through the network formed by the computational dependencies between parameters.

Applying a fix might introduce new violations. P&R tries to reduce the complexity of configuration design by disallowing recursive fixes. Instead, if applying a fix introduces a new constraint violation, P&R discards the fix and tries a new combination. Motta and colleagues [27] have pointed out that, in terms of the flow of control, P&R offers two possibilities. One can perform verification and revision directly after every

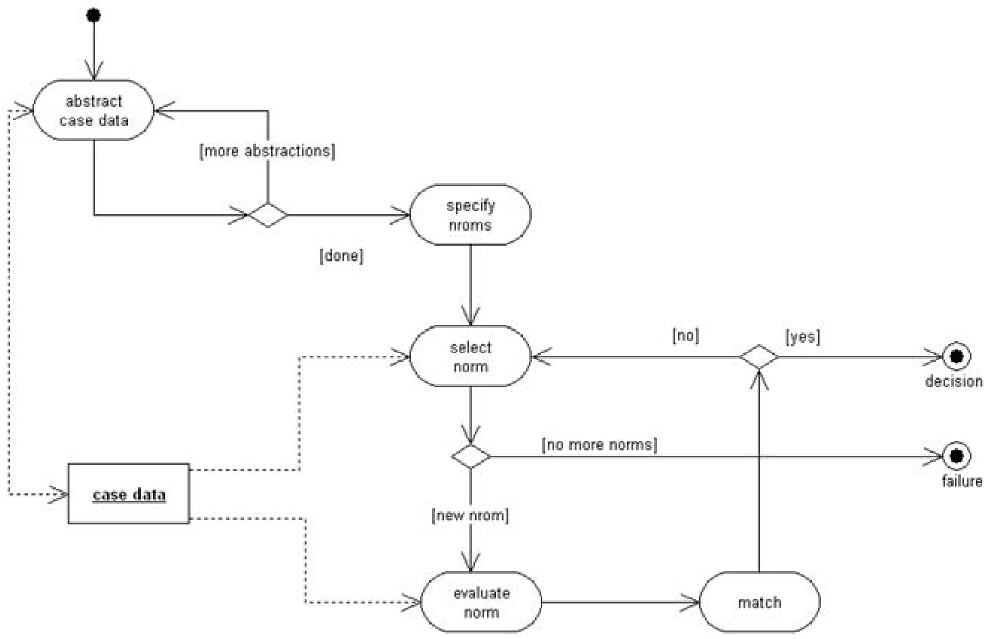

Figure 25.2: Top-level reasoning strategy of the basic assessment method in the form of a UML activity diagram.

new design or after all parameter values have been proposed. The original P&R system used the first strategy, but Motta argues that the second strategy is more efficient and also comes up with a different set of solutions. Although this method has worked in practice, it has inherent limitations. Using fix knowledge implies that heuristic strategies guide the design revisions. Fix knowledge implicitly incorporates preferences for certain designs. This makes it difficult to assess the quality of the method's final solution.

Assessment. Assessment is a task not often described in the AI literature, but of great practical importance. Many assessment application have been developed over the years, typically for tasks in financial domains, such as assessing a loan for mortgage application, or in the civil-service area, such as assessing whether a permit can be given. The task is often confused with diagnosis, but where diagnosis is always considered with some faulty state of the system, assessment is aimed at producing a *decision*: e.g., yes/no to accept a mortgage application. During the Internet hype at the start of this decade every bank was developing such applications to be able to offer automated services on the Web.

A basic method for assessment is shown in Fig. 25.2. Assessment starts off with case data (e.g., customer data about a mortgage application). As a first step the raw case data is abstracted into more general data categories (e.g., income into income class). Subsequently, domain-specific norms/criteria are retrieved (e.g., "minimal income") and evaluated against the case data. The method then checks whether a decision can be taken or whether more norms need to be evaluated. This basic method is typically enhanced with domain-specific knowledge, e.g., select inexpensive (e.g., in terms of data acquisition) first. The resulting decision category are also domain-specific; for example, for a mortgage application this could be “accepted”, “declined”, or “flag for manual assessment”.[1]

A detailed example of the use of this method can be found in the CommonKADS book [37, Ch. 10]. Valente and Löckenhoff [43] have published a library of different assessment methods.

25.3.2 The Notion of “Knowledge Role”

Above we showed two examples of methods for different tasks. These methods cannot be applied directly to a domain; typically, the knowledge engineer has to link the components of the method to elements of the application domain. Problem-solving methods can best be viewed as *patterns*: they provide template structures for solving a problem of a particular type. Designing systems with the help of patterns is in fact a major trend in software engineering at large, see, for example, the work of Gamma and colleagues [13] on design patterns.[2]

The knowledge-engineering literature provides a number of proposals for specification frameworks and/or languages of problem-solving methods. These include the “Generic Task” approach [6], “Role-Limiting Methods” [24], “Components of Expertise” [40], Protégé [32], KADS [48, 49] and CommonKADS [39]. Although there differences at a detailed level between these approaches, the one important commonality is: all rely on the notion of “knowledge role”:

> *A* knowledge role *specifies in what way particular domain knowledge is being used in the problem solving process.*

Typical knowledge role in the assessment method are “case data”, “norm” and “decision”. These are method-specific names for the role that pieces of domain knowledge play during reasoning. From a computational perspective, they limit the role that these domain-knowledge elements can play, and therefore make problem solving more feasible, when compared to old “old” expert-systems idea of one large knowledge-vase with a uniform reasoning strategy. In fact, the assumption behind PSM research is that the epistemological adequacy of the method gives one a handle on the computational tractability of the system implementation based on it. This issue is of course a long-standing debate in knowledge representation at large (see, e.g., [4]).

Another issue that frequently comes up in discussions about problem-solving methods is their correspondence with human reasoning. Early work on KADS used problem-solving methods as a coding scheme for expertise data [47]. Over the years the growing consensus has become that, while human reasoning can form an important inspirational source for problem-solving method and while it is use to use role cognitively-plausible terms for knowledge role, the problem-solving strategy may well be different. Machines have different qualities than humans. For example, a method that requires a large memory space cannot be carried out by a human expert, but presents no problem to a computer program. In particular methods for synthetic tasks,

[1]Many of these assessment systems are aimed at reducing administrative workload and are not designed to solve the standard cases and leave atypical ones for manual assessment.

[2]Problem-solving methods would be called “strategy patterns” in the terminology of Gamma et al. [13].

where the solution space is usually large, problem-solving methods often have no counterpart in human problem solving.

25.3.3 Specification Languages

In order to put the notions of "problem-solving method" and "knowledge role" on a more formal footing, the mid-1990s saw the development of a number of formal languages that were specifically designed to capture these notions.

The goal of such languages was often twofold. First of all, to provide a formal and unambiguous framework for *specifying* knowledge models. This can be seen as analogous to the role of formal specification languages in Software Engineering, which aim to use logic to describe properties and structure of software in order to enable the formal verification of properties. Secondly, and again analogous to Software Engineering, some of these formal languages could be made executable (or contained executable fragments), which could be used to simulate the behavior of the knowledge models on specific input data. Most of the languages that were developed followed the maxim of *structure preserving specification* [44]: if the structure of the formal specification closely follows the structure of the informal knowledge model, any problems found during verification activities performed on the formal model can be easily translated in terms of possible repairs on the original knowledge model.

In particular the Common KADS framework was the subject of a number of formalization attempts, see [12] for an extensive survey. Such languages would follow the structure of Common KADS model into (1) a *domain layer*, where an ontology is specified describing the categories of the domain knowledge and the relationships between these categories (i.e., the boxes in Fig. 25.5); (2) *knowledge roles* link the components of the method to elements of the application domain; (3) *inference steps* that are the atomic elements of a problem solving method (i.e., the ovals in Fig. 25.2), and (4) a task definition which emposes a control structure over the inference steps to complete the definition of the problem solving method.

A simplified example is shown in Fig. 25.3, using a simplification of the syntax of (ML)^2 [45]:

- the *domain layer* specifies a number of declarative facts in the domain. These facts are already organized in three different modules.
- the *knowledge roles* empose a problem-solving interpretation on these neutral domain facts: any statement from the patient-data module is interpreted as data, any implication from the symptom-definition module is interpreted as an abstraction rule, and any implication from the symptomatology module is interpreted as a causal rule.
- the *inference steps* then specify how these knowledge roles can be used in a problem solving method: an abstraction step consists of a deductive (modus ponens) step over an abstraction rule, whereas a hypothesize step consists of an abductive step over a causation rule.
- finally, the *task model* specifies how these atomic inference steps must be strung together procedurally to form a problem solving method: in this a sequence of a deductive abstraction step followed by an abductive hypothesize step.

DOMAIN

`patient-data:` $temp(patient1) = 38$
`symptom-definitions:` $temp(P) > 37 \rightarrow fever(P)$
`sympotomatology:` $hepatitis(P) \rightarrow fever(P)$

KNOWLEDGE ROLES

from `patient-data`: $A \mapsto data(A)$
from `symptom-definition`: $A \rightarrow B \mapsto abstraction(A, B)$
from `sympotomatology`: $A \rightarrow B \mapsto causation(A, B)$

INFERENCE

$abstract(A_1, A_2)$: $data(A_1) \wedge abstraction(A_1, A_2) \rightarrow observation(A_2)$
$hypothesise(B_1, B_2)$: $observation(B_2) \wedge causation(B_1, B_2) \rightarrow hypothesis(B_1)$

TASK

begin abstract(A,B) **;** hypothesize(B,C) **end**

Figure 25.3: A simple problem-solving method specification in the style of (ML)^2.

The impact of the languages such (ML)^2 [45], KARL [11] and many others (see [12]) was in one sense very limited: although the knowledge modeling methods are in widespread use, the corresponding formal languages have not received widespread adaptation. Rather than direct adoption, their influence is perhaps mostly seen through the fact that they forced a much more precise formulation of the principles behind the knowledge modeling methods.

There is renewed activity in the area of formal languages for problem solving methods at the time of writing. This is causes by an interest from web services. Web-services are composed into work-flows, and these workflows often exhibit typical patterns (e.g., browse-order-pay-ship, or search-retrieve-process-report). Problem solving methods are essentially reusable workflows of reasoning-patterns, and the established lessons from problem solving methods may well be applicable to this new area.

25.4 Ontologies

During the nineties ontologies become popular in computer science. Gruber [16] defines an ontology as an "explicit specification of a conceptualization". Several authors have made small adaptations to this. A common definition nowadays is:

> *An* ontology *is an explicit specification of a shared conceptualization that holds in a particular context.*

The addition of the adjective "shared" is important, as the primary goal of ontologies in computer science was to enable knowledge sharing. Up till the end of the nineties "ontology" was a niche term, used by a few researchers in the knowledge engineering and representation field.[3] The term is now in widespread use, mainly due

[3] At a preparation meeting for a DARPA program in this area in 1995, the rumors were that DARPA management talked about the O-word.

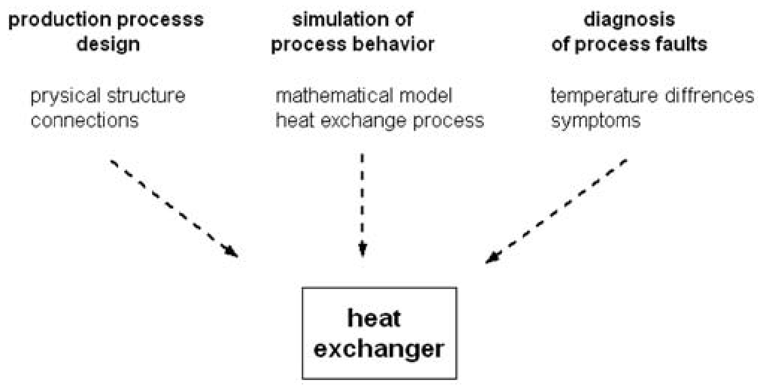
production processs design
prysical structure connections
simulation of process behavior
mathematical model heat exchange process
diagnosis of process faults
temperature diffrences symptoms
heat exchanger

Figure 25.4: Three different viewpoints on a heat exchanger.

to enormous need for shared concepts in the distributed world of the web. People and programs need to share at least some minimal common vocabulary. Ontologies have become in particular popular in the context of the Semantic Web effort, see Chapter 21.

In practice, we are confronted with many different conceptualizations, i.e., ways of viewing the world. Even is in a single domain there can be multiple viewpoints. Take, for example, the concept of a heat exchanger as shown in Fig. 25.4. The conceptualization of a heat exchanger is can be very different, depending on whether we take the viewpoint of the physical structure, the internals of the process, or the operational management.

"Context" is therefore an important notion when reusing an ontology. We cannot expect other people or programs to understand our conceptualization, if we do not explicate what the context of the ontology is. Lenat [21] has made an attempt to define a theory of context spaces. In practice, we see most often that context is being defined though typing the ontology. We discuss ontology types in Section 25.4.2 and/or reusing an ontology.

The plural form used in the title of this section is revealing. The notion of ontology has been a subject of debate in philosophy for many ages. The study of ontology, or the theory of "that what is" (from the Greek "ontos" = being), has been a discipline in its own right since the days of Aristotle, who can be seen as founder and inspirator. The plural form signifies the pragmatic use made of the notion in modern computer science. We talk now about "ontologies" as the state of the art does not provide us with a single theory of what exists.

25.4.1 Ontology Specification Languages

Many of the formalisms can be said to be useful for specifying an ontology. An insightful article into the ontological aspects of KR languages is the paper by Davis and colleagues [10]. They define five roles for a knowledge representation, which we can briefly summarize as follows:

1. A surrogate for the things in the real world.

2. A set of ontological commitments.

3. A theory of representational constructs plus inferences it sanctions/recommends.
4. A medium for efficient computation.
5. A medium for human expression.

One can characterize ontology-specification languages as KR languages that focus mainly on roles 1, 2 and 5. In other words, ontologies are not specified with a particular reasoning paradigm in mind.

There have been several efforts to define tailor-made ontology-specification languages. In the context of the DARPA Knowledge Sharing Effort Gruber defined Ontolingua [16]. Ontolingua was developed as an ontology layer on top of KIF [15], which allowed frame style definition of ontologies (classes, slots, subclasses, . . .). Additional software was provided to be able the use of Ontolingua as a mediator between different knowledge-representation languages, such as KIF and LOOM. Ontolingua provided a library service where users share their ontologies.[4]

Other languages, in particular conceptual graphs (see Chapter 5) have been popular for specifying ontologies. Recently, OWL has gained wide popularity. OWL is the W3C Web Ontology Language [46]. Its syntax is XML based. Things defined in OWL get a URI, which simplifies reuse. OWL sails between Scylla of expressiveness and the Charybdis of computability by defining a subset of OWL (OWL DL) that is equivalent to a well-understood fragment of description logic (see Chapter 3). User who limit themselves to this fragment of OWL get some guarantees with respect to computability. The OWL user is free to step outside the bounds of OWL DL, if s/he requires additional expressive power. An overview of OWL is given in Chapter 21.

One might ask, whether the use of description logic as a basis for an ontology language does to contradict the statement of the start of this section, namely that ontologies are not specified with a reasoning mechanism in mind. It is undoubtedly true that the DL reasoning paradigm biases the way one models the world with OWL. However, subclass modeling appears to be an intrinsic feature of modeling domain knowledge. The use of a DL-style modeling in knowledge of domains has been popular since the early days of KL-ONE [5]. Also, DL reasoning is often mainly used to validate the ontology; typically, additional reasoning knowledge is needed in applications. The fact that Web community is defining a separate rule language to complement OWL is also evidence for this. Still, one could take the view that a more general first-order language would be better for ontology specification, as it introduces less bias and provides the possibility of specifying reasoning within the same language. If one takes this position, a language like KIF [15] is a prime candidate as ontology language.

25.4.2 Types of Ontologies

Ontologies exist in many forms. Roughly, ontologies can be divided into three types: (i) foundational ontologies, (ii) domain-specific ontologies, and (iii) task-specific ontologies.

[4]http://ontolingua.stanford.ed.

Foundational ontologies. Foundational ontologies stay closest to the original philosophical idea of "ontology". These ontologies aim to provide conceptualizations of general notions, such as time, space, events and processes. Some groups have published integrated collections of foundational ontologies. Two noteworthy examples are the SUMO (Suggested Upper Merged Ontology)[5] and DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering).[6] An ontology of time has been published by Hobbs and Pan [20], which includes Allen's set of time relations [1]. Chapter 12 of this Handbook also addresses time representation.

Ontologies for part–whole relations have been an important area of study. Unlike the subsumption relation, part–whole relations are usually not part of the basic expressivity of the representation language. In domains dealing with large structures, such as biomedicine, part–whole relations are often of prime importance. A simple baseline representation of part–whole relations is given by Rector and Welty [35]. Winston et al. published a taxonomy of part–whole relations, distinguishing, for example, assembly–component relations from portion–mass relations. Such typologies are of practical importance as transitivity of the part–whole relation does not hold when different part–whole relations are mixed ("I'm part of a club, my hand is part of me, but this does not imply my hand is part of the club"). Several revised versions of this taxonomy have been published [30, 2].

Lexical resources such as WordNet[7] [26], can also be seen as foundational ontologies, although with a weaker semantic structure. WordNet defines a semantic network with 17 different relation types between concepts used in natural language. Researchers in this area are proposing richer semantic structuring for WordNet (e.g., [31]). The original Princeton WordNet targets the English–American language; WordNets now exist or are being developed for almost all major languages.

Domain-specific ontologies. Although foundational ontologies are receiving a lot of attention, the majority of ontologies are domain-specific: they are intended for sharing concepts and relations in a particular area of interest. One domain in which a wide range of ontologies has been published is biomedicine. A typical example is the Foundational Model of Anatomy (FMA) [36] which describes some 75,00 anatomical entities. Other well-known biomedical ontologies are the Unified Medical Language System[8] (UMLS), the Simple Bio Upper Ontology,[9] and the Gene Ontology.[10]

Domain ontologies vary considerably in terms of the level of formalization. Communities of practice in many domains have published shared sets of concepts in the form of vocabularies and thesauri. Such concept schemes typically have a relatively weak semantic structure, indicating many hierarchical (broader/narrower) relations, which most of the time loosely correspond to subsumption relations. This has triggered a distinction in the ontology literature between weak versus strong ontologies. The SKOS model,[11] which is part of the W3C Semantic Web effort, is targeted at

[5]http://ontology.teknowledge.com/.

[6]http://www.loa-cnr.it/dolce.html.

[7]http://wordnet.princeton.edu/.

[8]http://www.nlm.nih.gov/pubs/factsheets/umls.html.

[9]http://www.cs.man.ac.uk/~rector/ontologies/simple-top-bio/.

[10]http://www.geneontology.org/.

[11]http://www.w3.org/2004/02/skos/.

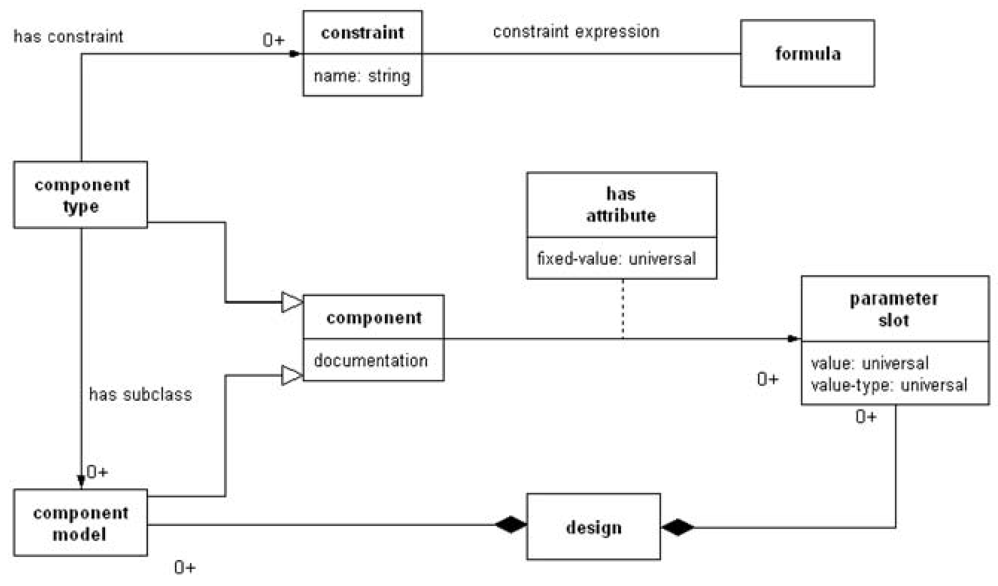

Figure 25.5: Configuration-design ontology in the VT experiment [18] (in the form of a UML class diagram).

allowing thesaurus owners to publish their concept schemes in an interoperable way, such that sharing of these concepts on the Web becomes easier. In practice, thesauri are important sources for information sharing (the main goal of ontologies in computer science). For example, in the cultural-heritage domain thesauri such as the Getty vocabularies[12] (Art & Architecture Thesaurus, Union List of Artist Names, Thesaurus of Geographic Names) and IconClass (concepts for describing image content) are important resources. Current efforts focus therefore on making such vocabularies available in ontology-representation formats and enriching ("ontologizing") them.

Task-specific ontologies. A third class of ontologies specifies the conceptualizations that are needed for carrying out a particular task. For each of the task types listed in Table 25.1 one can specify domain conceptualizations needed for accomplishing this task. An example of a task-specific ontology for the configuration-design task can be found in Fig. 25.5. Data of configuration-design of an elevator system were used in the first ontology-reuse experiment in the nineties [38].

In general, conceptualizations of domain information needed for reasoning algorithms typically takes the form of a task-specific ontology. For example, search algorithms typically operate on an ontology of states and state transitions. Tate's plan ontology [42] is another example of a task-specific ontology.

25.4.3 Ontology Engineering

Ontology engineering is the discipline concerned with building and maintaining ontologies. It provides guidelines for building domain conceptualizations, such as the construction of subsumption hierarchies.

[12]http://www.getty.edu/research/conducting_research/vocabularies/.

An important notion in ontology engineering is *ontological commitment*. Each statement in an ontology commits the user of this ontology to a particular view of the domain. If a definition in an ontology is stronger than needed, than we say that the ontology is over-committed. For example, if we state that the name of a person must have a first name and a last name we are introducing a western bias into the ontology and may not be able to use the ontology in all intended cases. Ontology engineers usually try to define an ontology with a minimal set of ontological commitments. One can translate this into an (oversimplified) slogan: “smaller ontologies are better!”. Gruber [17] gives some principles for minimal commitments.

Construction of subsumption hierarchies is seen as a central activity in ontology engineering. The OntoClean method of Guarino and Welty [19] defines a number of principles for this activity, based on three meta-properties of classes, namely rigidity, unity and identity. Central in the OntoClean method is the identification of so-called “backbone” classes of the ontology. Rector [33] defines also a method for backbone identification.

In addition, design patterns have been specified for frequently occurring ontology-engineering issues. We mention here the work of Noy on patterns for defining N-ary relations [29] (to be used with an ontology language that supports only binary relations, such as OWL) and the work of Rector on patterns for defining value sets [34]. Gangemi has published a set of design patterns for a wide range of modeling situations [14].

25.4.4 Ontologies and Data Models

The difference between ontologies and data models does not lie in the language being used: you can define an ontology in a basic ER language (although you will be hampered in what you can say); similarly, you can write a data model with OWL. Writing something in OWL does not make it an ontology! The key difference is not the language the intended use. A data model is a model of the information in some restricted well-delimited application domain, whereas an ontology is intended to provide a set of shared concepts for multiple users and applications. To put it simply: data models live in a relatively small closed world; ontologies are meant for an open, distributed world (hence their importance for the Web). So, defining a name as consisting of a first name and a last name might be perfectly OK in a data model, but may be viewed as incorrect in an ontology. It must be added that there is a tendency to extend the scope of data models, e.g., in large companies, and thus there is an increasing tendency to “ontologize” data models.

25.5 Knowledge Elicitation Techniques[13]

Although this entire Handbook is devoted to the formal and symbolic representation of knowledge, very few if any of its chapters are concerned with how such representations are actually obtained. Many techniques have been developed to help elicit knowledge from an expert. These are referred to as knowledge elicitation or

[13]Material in this section has been taken from the CommonKads book [37], the CommonKADS website at http://www.commonkads.uva.nl and the website of Epistemics, http://www.epistemics.co.uk.

knowledge acquisition (KA) techniques. The term “KA techniques” is commonly used.

The following list gives a brief introduction to the types of techniques used for acquiring, analyzing and modeling knowledge:

- Protocol-generation techniques include various types of interviews (unstructured, semi-structured and structured), reporting techniques (such as self-report and shadowing) and observational techniques.
- Protocol analysis techniques are used with transcripts of interviews or other text-based information to identify various types of knowledge, such as goals, decisions, relationships and attributes. This acts as a bridge between the use of protocol-based techniques and knowledge modeling techniques.
- Hierarchy-generation techniques, such as laddering, are used to build taxonomies or other hierarchical structures such as goal trees and decision networks.
- Matrix-based techniques involve the construction of grids indicating such things as problems encountered against possible solutions. Important types include the use of frames for representing the properties of concepts and the repertory grid technique used to elicit, rate, analyze and categorize the properties of concepts.
- Sorting techniques are used for capturing the way people compare and order concepts, and can lead to the revelation of knowledge about classes, properties and priorities.
- Limited-information and constrained-processing tasks are techniques that limit the time and/or information available to the expert when performing tasks. For instance, the twenty-questions technique provides an efficient way of accessing the key information in a domain in a prioritized order.
- Diagram-based techniques include the generation and use of concept maps, state transition networks, event diagrams and process maps. The use of these is particularly important in capturing the “what, how, when, who and why” of tasks and events.

Specialized tool support has been developed for each of these techniques. Table 25.2 briefly describes some of these techniques, and correlates them with the appropriate tool support.

This wide variety of techniques is required to access the many different types of knowledge possessed by experts. This is referred to as the Differential Access Hypothesis, and has been shown experimentally to have supporting evidence.

Fig. 25.6 presents the various techniques described above and shows the types of knowledge they are mainly aimed at eliciting. The vertical axis on the figure represents the dimension from object knowledge to process knowledge, and the horizontal axis represents the dimension from explicit knowledge to tacit knowledge. The details of these techniques are described in a number of survey articles and textbooks, such as [3], [37, Ch. 8], and [25].

Table 25.2. Summary of elication techniques

Technique	Used for	Tool support
Unstructured interview	Familiarization with organization and application domain	Markup tools; text analysis
Structured interview	Knowledge-identification activities; initial knowledge specification; completing the knowledge bases	Markup tools; rule editor (when used for completing the knowledge base)
Protocol analysis	Checking a task template Generating an inference/task specification (in case of unfamiliar application domains, for which no models exist yet)	Marking up a transcript with inference and/or task markers
Laddering	Preparatory work for domain-schema specification with respect to useful hierarchies and concept attributes	Graphical support for constructing multiple hierarchies
Concept sorting	Domain-schema specification in unfamiliar domains	Graphical support tool for creating piles and new features
Repertory grid	Domain-schema specification in unfamiliar domains	Graphical grid presentation/editing plus cluster analysis software

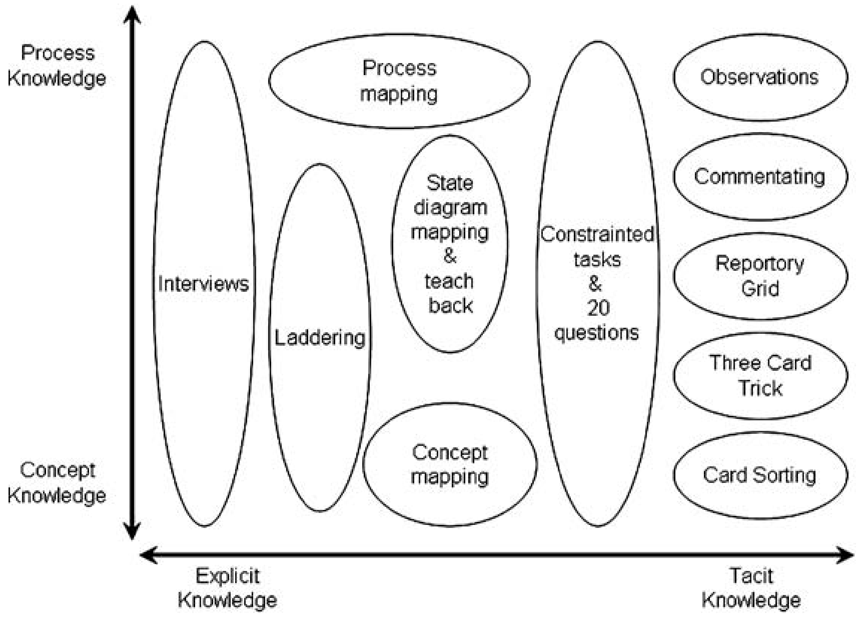

Figure 25.6: Applicability of Knowledge Acquisition techniques.

Bibliography

[1] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.

[2] A. Artale, E. Franconi, and L. Pazzi. Part-whole relations in object-centered systems: An overview. *Data and Knowledge Engineering*, 20:347–383, 1996.

[3] J.H. Boose. A survey of knowledge acquisition techniques and tools. *Knowledge Acquisition*, 1(1):3–37, 1989.

[4] R.J. Brachman and H.J. Levesque. The tractability of subsumption in frame-based description languages. In *AAAI 84*, 1984.

[5] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, 1985.

[6] B. Chandrasekaran. Generic tasks in knowledge based reasoning: High level building blocks for expert system design. *IEEE Expert*, 1(3):23–30, 1986.

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

[8] W.J. Clancey. The epistemology of a rule based system—a framework for explanation. *Artificial Intelligence*, 20:215–251, 1983.

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

[10] R. Davis, H. Shrobe, and P. Szolovits. What is a knowledge representation? *AI Magazine*:17–33, Spring 1993.

[11] D. Fensel. *The Knowledge Acquisition and Representation Language KARL*. Kluwer, ISBN-13: 978-0792396017, 1995.

[12] D. Fensel and F. van Harmelen. A comparison of languages which operationalise and formalise KADS models of expertise. *The Knowledge Engineering Review*, 9:105–146, 1994.

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

[14] A. Gangemi. Ontology design patterns for Semantic Web content. In *International Semantic Web Conference ISWC'05, Galway, Ireland*, *LNCS*, pages 262–276. Springer-Verlag, 2005.

[15] M.L. Ginsberg. Knowledge interchange format: the KIF of death. *AI Magazine*, 12(33):57–63, 1991.

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

[17] T.R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli, editors. *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Kluwer, Boston, 1994.

[18] T.R. Gruber, G.R. Olsen, and J. Runkel. The configuration-design ontologies and the VT elevator domain theory. *Int. J. Human-Computer Studies*, 44(3–4):569–598, 1996.

[19] N. Guarino and C. Welty. Evaluating ontological decisions with OntoClean. *Comm. ACM*, 45(2):61–65, 2002.

[20] J.R. Hobbs and F. Pan. Time ontology in OWL. Technical report, Ontology Engineering Patterns Task Force of the Semantic Web Best Practices and Deployment Working Group, World Wide Web Consortium (W3C), http://www.w3.org/TR/owl-time/, 2006.

[21] D. Lenat. The dimensions of context space. Technical report, CYCORP. URL: http://www.cyc.com/doc/context-space.pdf, 28 October 1998.

[22] S. Marcus, editor. *Automatic Knowledge Acquisition for Expert Systems*. Kluwer, Boston, 1988.

[23] S. Marcus and J. McDermott. SALT: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence*, 39(1):1–38, 1989.

[24] J. McDermott. Preliminary steps towards a taxonomy of problem-solving methods. In S. Marcus, editor. *Automating Knowledge Acquisition for Expert Systems*, pages 225–255. Kluwer, Boston, 1988.

[25] M.A. Meyer and J.M. Booker. *Eliciting and Analyzing Expert Judgement: A Practical Guide*. Academic Press, 1991.

[26] G. Miller. WordNet: A lexical database for English. *Comm. ACM*, 38(11), November 1995.

[27] E. Motta, A. Stutt, Z. Zdrahal, K. O'Hara, and N.R. Shadbolt. Solving VT in VITAL: a study in model construction and reuse. *Int. J. Human-Computer Studies*, 44(3–4):333–372, 1996.

[28] A. Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.

[29] N. Noy and A. Rector. Defining N-ary relations on the Semantic Web. Technical report, W3C Working Group Note, http://www.w3.org/TR/swbp-n-aryRelations, 2006.

[30] J. Odell. Six different kinds of composition. *Journal of Object Oriented Programming*, 5(8):10–15, 1994.

[31] A. Oltramari, A. Gangemi, N. Guarino, and C. Masolo. Restructuring WordNet's top-level: The OntoClean approach. In *Proc. LREC 2002*, 2002.

[32] 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.

[33] A. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In *Proc. K-CAP'03*, pages 121–128. AAAI, 2003.

[34] A. Rector. Representing specified values in OWL: "value partitions" and "value sets". Technical report, W3C Working Group Note, http://www.w3.org/TR/swbp-specified-values, 2005.

[35] A. Rector and C. Welty. Simple part–whole relations in OWL ontologies. Technical report, World-Wide Web Consortium (W3C), Working Group Note, http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/, 2005.

[36] C. Rosse and J.V.L. Mejino. A reference ontology for biomedical informatics: the foundational model of anatomy. *J. Biomedical Informatics*, 36:478–500, 2003.

[37] 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, Cambridge, MA, 1999.

[38] A.Th. Schreiber and W.P. Birmingham. The Sisyphus-VT initiative. *Int. J. Human-Computer Studies*, 43(3–4):275–280, 1996 (Editorial special issue).

[39] 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.

[40] L. Steels. Components of expertise. *AI Magazine*, Summer 1990.

[41] M. Stefik. *Introduction to Knowledge Systems*. Morgan Kaufmann, Los Altos, CA, 1993.

[42] A. Tate. Towards a plan ontology. *Journal of the Italian AI Association (AIIA)*, January 1996.

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

[44] F. van Harmelen and M. Aben. Structure preserving specification languages for knowledge-based systems. *International Journal of Human Computer Studies*, 44:187–212, 1996.

[45] F. van Harmelen and J.R. Balder. $(ML)^2$: a formal language for KADS models of expertise. *Knowledge Acquisition*, 4(1), 1992. Special issue: 'The KADS approach to knowledge engineering', reprinted in A.Th. Schreiber, et al., editors, *KADS: A Principled Approach to Knowledge-Based System Development*, 1993.

[46] Web Ontology Working Group. OWL web ontology language overview. W3C recommendation, World Wide Web Consortium, http://www.w3.org/TR/owl-features/, 2004.

[47] B.J. Wielinga and J.A. Breuker. Interpretation of verbal data for knowledge acquisition. In T. O'Shea, editor, *Advances in Artificial Intelligence*, pages 41–50, Amsterdam, The Netherlands, 1984. ECAI, Elsevier Science. Also as: Report 1.4, ESPRIT Project 12, University of Amsterdam.

[48] B.J. Wielinga and J.A. Breuker. Models of expertise. In *Proceedings ECAI-86*, pages 306–318, 1986.

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