Meta-Knowledge in systems design: panacea...or undelivered promise?

Yannis Kalfoglou*
yannisk@dai.ed.ac.uk

Tim Menzies†
tim@menzies.com

Klaus-Dieter Althoff‡
klaus-dieter.althoff@iese.fhg.de

Enrico Motta§
e.motta@open.ac.uk

Abstract

In this study we present a review of the emerging field of meta-knowledge components as practised over the past decade among a variety of practitioners. We use the artificially-defined term 'meta-knowledge' to encompass all those different but overlapping notions used by the Artificial Intelligence and Software Engineering communities to represent reusable modelling frameworks: ontologies, problem-solving methods, experience factories and experience bases, patterns, to name a few. We then elaborate on how meta-knowledge is deployed in the context of system's design to improve its reliability by consistency checking, enhance its reuse potential, and manage its knowledge sharing. We speculate on its usefulness and explore technologies for supporting deployment of meta-knowledge. We argue that, despite the different approaches being followed in systems design by divergent communities, meta-knowledge is present in all cases, in a tacit or explicit form, and its utilisation depends on pragmatic aspects which we try to identify and critically review on criteria of effectiveness.

keywords: Ontologies, Problem-Solving Methods, Experienceware, Patterns, Design Types, Cost-Effective Analysis.

1 Introduction

As knowledge engineers we are trying to find ways of building intelligent systems better, faster, and cheaper. A way of achieving this is by deploying meta-knowledge: we use this term to refer to knowledge that has been acquired and stored in prior system development activities and that is being applied to the

*School of Artificial Intelligence, 80 South Bridge, University of Edinburgh, Edinburgh EH1 1HN, Scotland - now associated with: Knowledge Media Institute(KMi), The Open University, Walton Hall, Milton Keynes MK7 6AA, England

†NASA/WVU Software Research Lab, 100 University Drive, Fairmont, WV, USA - now associated with: Department of Electrical and Computer Engineering, 2356 Main Mall, Vancouver V6T1Z4 B.C., Canada

‡Fraunhofer Institute for Experimental Software Engineering, Kaiserslauten, Germany

§Knowledge Media Institute(KMi), The Open University, Walton Hall, Milton Keynes MK7 6AA, England

current software design project to improve the quality of the end product and to reduce its cost. Meta-knowledge is seen as a productivity tool by engineers and we found evidence for this in the cognitive psychology literature: as Anderson reports in (Anderson 1990), human experts rarely solve problems from first principles. Such basic reasoning can be very slow. Experts are faster than novices since experts draw on their experiences. As much as possible, experts adapt their prior experiences to the new situation. An evidence for this, Anderson continues, is that the dominant influence on expertise is lengthy practice with the domain and not initial training.

In modern knowledge acquisition(hereafter, KA) we tacitly accept this theory and extend it as follows:

- Symbolic descriptions of past experience aid human experts and seeks to build libraries of such past experience. These are intended to support reuse and components in these libraries can be either application-specific or generic. In the former case we reuse by analogy whereas in the latter we reuse by instantiating generic compoments. This is supported by tools that are built to apply this experiential knowledge to new situations;

- We identify at least two forms of knowledge: focused knowledge about the current application or meta-knowledge that transcends the current application and applies to the domain in question.

In this article we investigate meta-knowledge. In the literature, we identify at least the meta-knowledge types shown in figure 1.

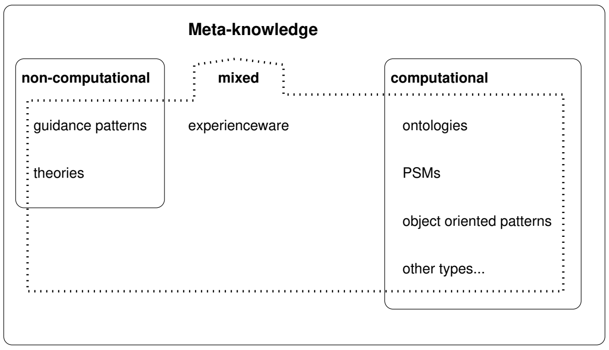


Figure 1: Meta-knowledge types reported in the literature.

The typology shown in the figure may not be a complete picture of the field but reveals how broad and intricate meta-knowledge is: Guidance patterns direct novice analysts to a set of issues that experienced analysts have found insightful. Theoretical investigations are trying to find ways of transforming theories to computer-readable formats in order to enable mechanised-reasoning. Experienceware is used in the software engineering community as a means to promote reuse of all-kinds of artefacts in a software organisation. Ontologies are explicit representations of a shared understanding of the important concepts in some domain of interest. PSMs are application-independent descriptions of problem-solving behaviour. Object-oriented patterns are abstractions of common parts in many designs and are mainly structurals. Many researchers have contributed to these fields of meta-knowledge. Although we describe these in the sequel, for an in-depth analysis we point the interested reader to review articles and special issues devoted to some of the fields included in the figure(for example, (Uschold and Gruninger 1996) for ontologies, (Benjamins and Fensel 1998) for Problem-Solving Methods(hereafter, PSMs), and (Menzies 1999b) for knowledge maintenance). In this article, we merely review samples of meta-knowledge by asking the following questions:

- What are the benefits of meta-knowledge and how it improves systems design?
- What are its associated costs and how can those benefits be realised?

The former question has been studied by many people in this area whereas the latter is usually ignored. Here, we focus on the second question and summarise what is known about the relative costs of using meta-knowledge. However, we cannot provide a complete summary. The state-of-the-art in metrics collection for knowledge engineering is inadequate to perform cost-benefit analyses for meta-knowledge. Instead, we can only point to certain partial indicators we see in current work. We will use these partial indicators to generate lists of potential issues with meta-knowledge.

At the end, we will elaborate on metrics that we could collect to test its usefulness and directions which should be followed to improve its exercise. This will give the reader an insight of how meta-knowledge can be a panacea for our systems and under which circumstances it is an undelivered promise.

1.a Outline

Initially though, we review design paradigms as reported in the literature. Meta-knowledge plays a fundamental role in design since it is seen as the constructive blocks upon which the system will be built. In section 2 we present a brief review of four commonly observed design types along with pointers to meta-knowledge involvement in those types.

We continue in section 3, by investigating types of meta-knowledge and elaborating on its benefits and how it has been realised in various projects drawn from the artificial intelligence(hereafter, AI) and software engineering(hereafter, SE) communities. We will also give pointers to emerging technologies for supporting organisation and deployment of meta-knowledge.

In the sequel, we discuss in section 4 the pragmatics of using meta-knowledge by focusing, mainly, on cost-related factors. We sample various cases to illustrate our points and identify future research directions and potential issues to be tackled which are summarised in section 5.

2 Design paradigms

In ((Moran and Carroll 1996)-page 3) the authors argue that four broad descriptions of design paradigms exist in the literature. In figure 2 we illustrate these along with a simple example of how they are conceived and deployed.

In the sequel we briefly describe the paradigms along with the types of meta-knowledge that can be classified under each of these. It is intended to show the involvement of meta-knowledge in design rather than acting as a directive for software design. For the latter we refer the interested reader to collections such as (Winograd 1996) and (Moran and Carroll 1996).

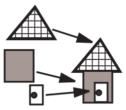
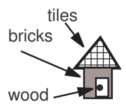
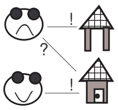
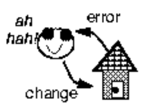
Design papadigm:	**Decomposition and Synthesis**	**Search**	**Negotiation and deliberation**	**Situated**
Driver:	Reuse libraries	Heuristics	Conflicts	Errors
Early work:	(Alexander, 1964), (Alexander et.al., 1977)	(Simon, 1969)	(Rittel et.al., 1973), (Finkelstein et.al., 1994)	(Schon, 1983)
Current work:	OO design patterns, KADS, etc.	(Rosenbloom et.al., 1993), (Menzies, 1998), (Josephson et.al., 1998)	(Finkelstein et.al., 1994) (Nissen et.al., 1996)	(Compton et.al., 1990)
Meta-knowledge types:	ontologies, PSMs, OO-pattens experienceware	non-computational, ontologies	*undefined*	computational: other types non-comp.: guidance patterns experienceware
Example:	Build the house using parts found when the last house was built. 	Build the house via a search of the space of what is known of bricks, tiles, and wood. 	Let each stakeholder design their preferred house, then lets discuss it 	Build the house, find out that the living room gets too hot in the summer, plant fruit trees for shade, make and sell the jam from the fruit. 

Figure 2: Building a house using four different paradigms of design.

2.a Design as Decomposition and Synthesis

This paradigm dates back at least to 1964 with Alexander's work on architecture(Alexander 1964). Design is taken to be the re-shuffling of components developed previously, then abstracted into reusable meta-knowledge. Modern expressions of this approach include object-oriented design patterns(Menzies 1997), ontologies(Uschold and Gruninger 1996), and PSMs(Benjamins and Fensel 1998). These are the dominant paradigms in the contemporary knowledge and software engineering. For example, the software engineering community is working on the contributions of 'experienceware'[1] which will be described in detail in section 3.

2.b Design as Search

Early work in this paradigm dates back to 1969 with Simon's work on AI(Simon 1969). Design is taken to be the traversal of a space of possibilities, looking for pathways to goals. Modern expressions of this approach include most of the AI search literature(Pearl and Korf 1987), *knowledge-level* search(Newell 1993), the SOAR papers(Rosenbloom et al. 1993), the SVF project(Josephson et al. 1998), certain KA approaches(Menzies 1998), etc. In a *knowledge-level* search, intelligence is modelled as a search for appropriate operators that convert some current

[1] This term was introduced in (von Wangenheim et al. 1998).

state to a goal state. Domain-specific knowledge is used to select the operators according to *the principle of rationality*: an intelligent agent will select an operator which its knowledge tells it will lead the achievement of some of its goals. In this paradigm we see preliminary work on modelling the domain-specific knowledge in computational forms of meta-knowledge, like domain-specific ontologies.

2.c Design as Negotiation and Deliberation

This paradigm dates back to at least 1973 with Ritell's work on *wicked problems*(Rittel and Webber 1973). Wicked problems have many features, the most important being that no objective measure of success exists. Designing solutions for wicked problems cannot aim to produce some perfectly correct answer since no such definition of *correct* exists. Hence, this approach to design tries to support effective debates by a community over a range of possible answers. Modern expressions of this approach include the requirements engineering community. Requirements engineering is usually complicated by the incompleteness of the specification being developed: while a specification should be consistent, requirements are often inconsistent. Requirements engineering researchers such as Easterbrook(Easterbrook 1991), and Finkelstein *et.al.*(Finkelstein et al. 1994), argue that we should routinely expect specifications to reflect different and inconsistent viewpoints. Traditionally, this paradigm has relied much on manual methods. In recent years more automatic techniques have been used. For example, the work of (Nissen et al. 1996) on conceptual modelling and in particular the use of declarative meta-models in requirements engineering and the work on negotiation in multiagent systems(Davis and Smith 1998). We cannot really identify meta-knowledge types in this paradigm although the design process that each stakeholder will follow to produce his/her own artefact may involve some kind of meta-knowledge.

2.d Situated Design

Schon's work on the *reflective practitioner*(Schon 1983) is among the first in this paradigm. In that approach, design mostly happens when some concrete artefact *talks back* to the designer - typically by failing in some important situation. That is, reflective design is less concerned with the creation of some initial artefact than the on-going re-interpretation and adjustment of that artefact. Modern expressions of this approach include the situated cognition community(Clancey 1989), certain approaches to design rationale(Casady 1996), and knowledge engineering techniques that focus on maintenance rather than initial design(Menzies 1998). In this paradigm we see types of meta-knowledge other than ontologies and PSMs. These will be described in section 3.

2.e Combinations

The design paradigms surveyed by Moran and Carroll do not explicitly mention possible combinations. For example, when designing new components we infer that their design is a combination of search, negotiation and deliberation, and situated design. Experienceware is also regarded as a combination of decomposition and synthesis and situated design paradigms.

3 Managing meta-knowledge

In this section we revisit the meta-knowledge types identified in figure 1. We briefly describe each type in section 3.a and then we proceed to analyse the benefits of meta-knowledge in system's design(section 3.b) and review representative applications where meta-knowledge has been deployed(section 3.c). In section 3.d we elaborate on emerging technologies for supporting organisation of meta-knowledge.

3.a Meta-knowledge types

As we can see from figure 1 we classify meta-knowledge in three clusters: non-computational, computational, and mixed. These are analysed below:

3.a.1 Non-computational

Non-computational meta-knowledge refers to knowledge that is not expressed in a machine-readable format. The level of abstraction at which we formalise meta-knowledge could be learnt via extensive experience with that particular topic(Althoff et al. 1999c). That is, it is not necessarily true that meta-knowledge should always be expressed in some computer-readable form. Sometimes, simply rendering it on paper will suffice. For example, object-oriented "guidance patterns" serve to direct novice analysts to a set of issues that experienced analysts have found insightful. Such patterns include CHECKS(Cunningham 1995), Caterpillar's Fate(Kerth 1995), etc. This type of meta-knowledge is stored as simple checklists of English text.

In this cluster we also place theoretical investigations in fields such as formal ontologies as studied in the philosophy discipline. Examples of this are the *conceptual realism* investigations(Cocchiarella 1996), the study of mereotopology, etc. Note that, this is an open area of research which tries to convert non-computational meta-knowledge to computational form, that is, to find ways of tranforming theories to computer-readable formats in order to enable mechanised reasoning(see, for example, (Kowalski and Sadri 1997)).

3.a.2 Computational

The majority of meta-knowledge research is focused on computational forms. These include ontologies, PSMs, object-oriented patterns and other types which are analysed in the sequel.

Ontologies: These are explicit representations of a shared understanding of the important concepts in some domain of interest. We see different levels of genericity in various ontologies. For instance, very broad ontologies, like CYC (Lenat and Guha 1990) which model generic notions that form the foundation for knowledge representation across various domains, are often regarded as a separate top-level, in comparison with domain-specific ontologies, like the `Enterprise` ontology(Uschold et al. 1998b), which capture domain-related knowledge. For an extensive discussion on different types of ontologies we point the interested reader to the survey articles of Uschold in (Uschold 1998) and Fridman-Noy's and Hafner's in (Fridman-Noy and Hafner 1997), and to Guarino's review in (Guarino 1998a).

The machine-readable format of this meta-knowledge type can be an implementation of a generic ontology, like mereology(Simons 1987) or a foundational theory like situation calculus(Pinto and Reiter 1993) or event calculus(Kowalski and Sadri 1997), or it can be specific such as an ontology for engineering mathematics(Gruber and Olsen 1994) or a representation of warplane types in the air-campaign planning domain(Valente et al. 1999).

The authoring of ontologies can be neutral or emerge from the application to be developed. In the former case we tend to see more generic forms(see, for example `Ontolingua`[2] ontologies) while in the latter more specific. The last distinction also dictates a different way of construction: the majority of generic ontologies are vast collections of formalised terminology and usually follow a top-down construction fashion, while the less generic ones are more compact and follow an opposite direction of construction: bottom-up (van der Vet and Mars 1998) or even middle-out.

The main focus of ontologies is to deliver knowledge sharing and reuse. It has been reported(Uschold and Gruninger 1996) that the use of ontologies is beneficial for the design of our systems in areas such as: communication between designers with different needs, interoperability among different systems, guidance for exploring a new domain, browsing and searching for domain-specific terminology and systems engineering. While most of these areas have been studied extensively in the literature[3] the systems engineering benefit is rarely discussed in the community. In section 3.c we sample applications where ontologies are deployed in order to realise, directly or indirectly, this benefit.

PSMs: These are reusable, application-independent descriptions of problem-solving behaviour. In the view of mainstream knowledge engineering(i.e.: KADS (Breuker and de Velde eds)), system development becomes a structured search for an appropriate PSM. Once a PSM is found or developed, then knowledge acquisition becomes a process of filling in the details required to implement that problem-solving method. It is argued that libraries of PSMs are a productivity tool for building a wide-variety of expert systems: integrated search/task PSMs(Motta and Zdrahal 1998); the SPARK/BURN/FIREFIGHTER(SBF) study(Marques et al. 1992); generic tasks(Chandrasekaran et al. 1992); configurable role-limiting methods(Gil and Melz 1996); CommonKADS(Schreiber et al. 1994); the PROTEGE approach(Eriksson et al. 1995); just to name a few.

However, researchers have found that the libraries built according to the 'PSM-solves-task' organisation criterion are difficult to reuse(Orsvarn 1996). This led to an increased interest to use strong epistemological foundations for structuring PSM libraries. Examples of these are described in (Motta 1999).

The standard architecture for a PSM is at least a two-layered system. In the bottom layer we see the domain-dependent facts in the language of the users. The upper layer contains the domain-independent PSMs which are written in a more general language. Some mapping function is defined to connect the domain-dependent language to a more general, domain-independent language. The domain-specific theory is assumed to be changeable and the same fact can take on different roles in different problem solving contexts via different map-

[2] Electronically accessible from http://www-ksl.stanford.edu/sns.html

[3] See (Uschold and Gruninger 1996) for a detail categorisation of these benefits along with examples and the volume edited by (Guarino 1998b) for extensive reports on applications of ontologies.

ping functions(Shadbolt and O'Hara 1997).

Object-oriented patterns: These are patterns that can generalise object-oriented implementations. Consider the left browser in figure 3: the class-hierarchy browser.

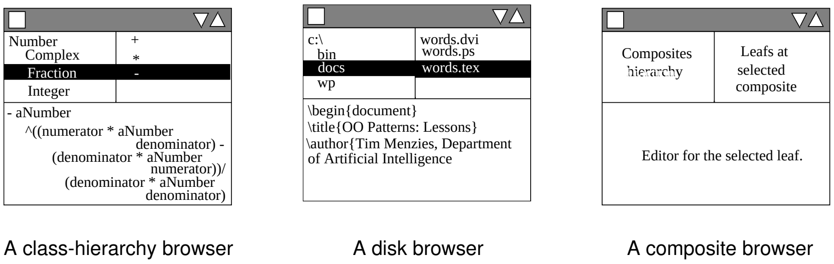


Figure 3: Various browsers.

When a class name is selected in the upper-left list box, the methods of that class are displayed in the upper-right list box. If one of these methods is selected, then the source code for that method is displayed in the bottom text pane. Now compare this class hierarchy browser with the disk browser shown in the middle of figure 3. When a directory name is selected in the upper-left list box, the files in that directory are displayed in the upper-right list box. If one of these files is selected, then the contents of that file are displayed in the bottom text pane. Obviously, there is some similarity in the two browsers. Container(i.e.: classes or directories) are shown top-left. The things in the containers that are not themselves containers(i.e.: methods and files) are shown top-right. The contents of these non-container things are shown in the bottom pane. If we rename containers to *composites* and the non-containers to *leaves* then we can design one *composite browser* class that handles both class hierarchies and directory trees as shown in the right browser of figure 3. That is, our disk browser and class hierarchy browser are both presentations of nested composites.

In figure 4 we illustrate the inner structure of the composite browser. Composites contain either other composites or leaves. Leaves compile the contents of the lower text-pane. Note the generality of figure 4: it can be used to (i) browse a disk; (ii) browse a class hierarchy; or, (iii) more generally, browse any 1-to-many nested aggregation (i.e.: players in teams, persons in companies, stock on shelves). This composite pattern is one of the 23 object-oriented reuse design patterns listed by (Gamma et al. 1995).

Patterns can be at different layers of abstraction. For example, in (Buschmann et al. 1996) three layers of abstraction are described: (i) low-level language-dependent *idiom* patterns; middle-layer language independent *design* patterns describing a programmer's key mechanisms; (iii) and top-level *architectural patterns* that spread across the entire application. In (Menzies 1997) the non-trivial overlap between KA research and object-oriented patterns is discussed. Menzies concludes that object-oriented patterns, ontologies, and PSMs are abstract descriptions of common parts of many designs; and that object-oriented patterns

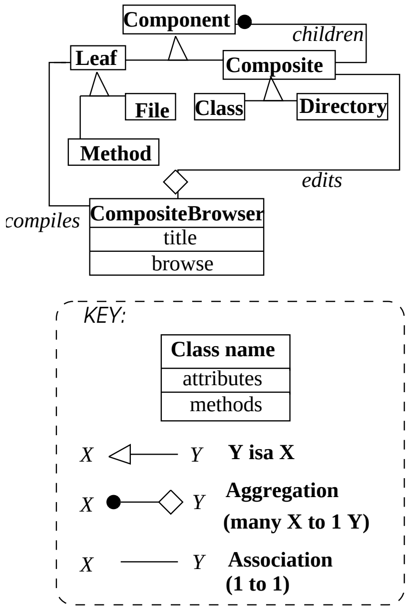


Figure 4: Object model for the composite browser.

are typically structural whereas PSMs are typically functional. The relation of object-oriented technologies, such as the modelling language UML(Booch et al. 1998) with ontologies is explored in (Cranefield and Purvis 1999).

Other types: In a recent review of knowledge maintenance strategies(Menzies 1999b), several knowledge types seen in the current KA literature were identified. Apart from ontologies and PSMs described above, other computational meta-knowledge types include *social*,*quality*, and *fix* knowledge.

Social knowledge refers to the social context of a system. Some expert systems practitioners argue that knowledge cannot be understood without understanding its social context. Paper documents are often collected which informally describe the organisational context of a system(e.g.: the organisational model of KADS(Wielinga et al. 1992)). An example of operationalising *social* knowledge is the REMAP system(Ramesh and Dhar 1992).

Quality knowledge stores assessment knowledge about a system and includes the work on non-functional requirements. Functional requirements can be measured via executing and measuring a program. Non-functional requirements such as portability, evolvability, development affordability, security, privacy or reusability cannot be assessed with respect to the current version of the working program. For example, consider the non-functional requirement of maintainability. Maintainability can only be definitively assessed in retrospect; i.e.: only after delivery has occurred and we have some track record of the system's performance in the field. Nevertheless, during initial construction, we may still want to assure ourselves as to the potential maintainability of the system. The QARCC tool(Boehm 1996) allows for different stakeholders in a system to represent their quality concerns.

Finally, *fix* knowledge specifies how to correct errors. Since fixing is normally a slow process, *fix* knowledge is usually expressed algorithmically for reasons of efficiency. An example of *fix* knowledge is given in the context of the SEEK systems(Politakis 1985). One general method for fixing a knowledge base is to handle transactions properly. In traditional databases, a transactions manager ensures the completion of all updates and/or deletions. Further, if some update and/or deletion is aborted half-way, the transaction manager reverses all the half-finished changes. Similarly, the EXPECT transactions manager(ETM) manages updates to methods in an object-oriented knowledge base(Gil and Tallis 1997). Errors are detected using partial evaluation. Given a particular example, the ETM performs a single forward propagation of types and reports an error when a method needs to fire, but it can't since the types of the input parameters to that method are not available.

3.a.3 Mixed

Recently, we have seen the emergence of another kind of meta-knowledge which combines computational and non-computational characteristics. These are the

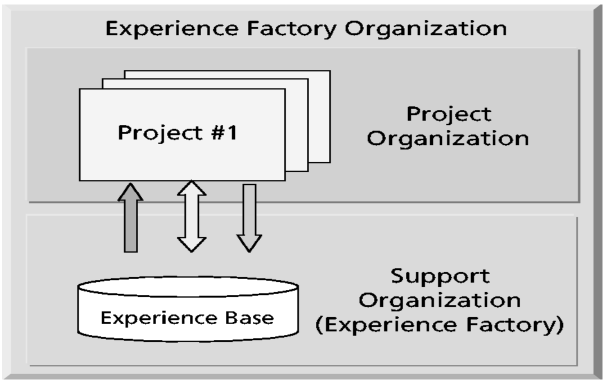


Figure 5: The *Experience Factory Organisation*.

experience factories(hereafter, EF) and their constituents, experience bases (hereafter, EB). In figure 5 we illustrate the organisation of an EF. EFs which were first investigated in the context of the TAME project(Basili and Rombach 1988), further generalised in the early nineties(Basili et al. 1994) as a means to promote reuse of all-kinds of artefacts in an organisation. The core of an EF is the EB which acts as the organisational memory. The key idea is to install an organisational memory to support exchange of all kinds of experiences in the life cycle of a software project. While the structure of an EB might differ from that of an ontology, the purpose is the same: to support reuse. The main focus of an EF is to support 'learning from experience' on a technology-independent organisational level. An example of an EB on improvement ideas and problem-solution statements for supporting continuous improvement processes in hospitals is given in (Althoff et al. 1999b).

3.b Meta-knowledge Benefits

With meta-knowledge we can achieve certain system engineering benefits. In (Uschold and Gruninger 1996) those benefits have been identified and classified in the context of ontologies. Here we elaborate on that classification and explore two major areas in the context of the meta-knowledge types discussed above:

1. reuse and knowledge sharing: the shared understanding of a domain of interest in the form of formal encoding of the important concepts makes it possible to reuse them in a software system. Furthermore, an implication of adhering to a shared understanding is to facilitate knowledge sharing among applications and across different development groups;

2. reliability: tacit knowledge and assumptions concerning a domain of interest are made explicit in a formal encoding which enforces automated consistency checking resulting in more reliable software. In the case where the explicit representation is not encoded formally, meta-knowledge serves as a basis for manually checking the design against the specification.

3.c Meta-knowledge applications

There is a large volume of applications of meta-knowledge reported in the literature. Although a complete listing is not feasible[4] we selected and briefly describe here representative applications that realise, directly or indirectly, the two benefits listed above.

The **Boeing** experiment, **AIRCRAFT**, **PIF**, **SBF**, **Coad et.al's** patterns, **PEBs**, **IBR0W** and **HPKB** focus on the reuse and knowledge sharing benefit whereas the **Comet**, **Cosmos**, **DISCOVER**, **TOVE**, **Repertory Grids** and a generic **Multi-layer** mechanism focus on the reliability benefit.

3.c.1 Reuse and knowledge sharing benefit

In an experiment of ontology reuse(Uschold et al. 1998a), researchers working at **Boeing** were investigating the potential of using an existing ontology for the purpose of specifying and formally developing software for aircraft design. The ontology used was the `EngMath` ontology(Gruber and Olsen 1994) and the application problem addressed was to enhance the functionality of a software component used to design the layout of an aircraft stiffened panel. Their conclusions were that despite the effort involved, knowledge reuse was cost-effective and that it would have taken significantly longer to design the knowledge content of the ontology used from scratch. However, the lack of automated support was an issue and the authors elaborate on the effort required from the knowledge engineer: "the process of applying an ontology requires converting the knowledge-level specification which the ontology provides into an implementation. This is time-consuming, and requires careful consideration of the context, intended usage, and idioms of both the source ontology representation language, and the target implementation language as well as the specific task of the current application."

[4] For ontologies resources we point the interested reader to the URL: http://www.dai.ed.ac.uk/daidb/people/homes/yannisk/seke99panelhtml.html for a classified list and to http://www.cs.utexas.edu/users/mfkb/related.html for a list updated by Peter Clark.

In (Valente et al. 1999) the authors discuss how they achieved reuse among ontologies themselves. The resulting ontology, **AIRCRAFT**[5], contains knowledge about types of US military aircraft, including data about the engines, pods, and fuel tanks that these aircraft can carry. Despite the fact that the authors had to face problems similar to those encountered in the Boeing experiment, their conclusions are indicative of the impact that this approach had to the system's design: "Having a well-structured ontology of a domain provides the basis on which to build, and thus helps enormously to develop new systems in that domain." A proof of this conclusion was the case study performed by Kalfoglou and Robertson, described in (Kalfoglou and Robertson 1999a), where the AIRCRAFT ontology was the basis for conceptual error checking in a prototype software system designed to defend an allied vessel from aircraft attack.

In the SPARK/BURN/FIREFIGHTER(**SBF**) study(Marques et al. 1992), 9 applications of intelligent computer hardware configuration were built with and without the SBF toolkit. The SBF auto-configured an expert system case tool via an automatic exploration of a library of PSMs. The ontologies of each method were then translated into data collection screens. Development times dropped from 63 to 250 days(without SBF) to 1 to 17 days(with SBF).

In (Bergmann and Althoff 1998) the authors describe an EF based method for developing software systems that use the Case-Based Reasoning(CBR) approach. A publicly available result is the CBR Product Experience Base(**PEB**) accessible from the URL: http://demolab.iese.fhg.de:8080. The **PEB** shows how the EF paradigm and in particular the EBs can be deployed to characterise and organise a range of products, like for example CBR tools.

In the **IBROW** project, the means of supporting comprehensive reuse are further investigated. The approach taken is an holistic one and proposes the development of various technologies required to achieve reuse: software architectures, a modelling language, brokering services and methodologies[6]. The idea behind this project is to provide a brokering service that plays the role of a mediator between customers and PSM providers to support the configuration of customised knowledge systems that solve customers' problems. Meta-knowledge models the different worlds of customers, brokers and PSMs providers. In (Motta et al. 1999) the authors investigated a particular issue in IBROW: how to construct a library of reusable components. The starting point was a pre-existing library of reusable components for parametric design(Motta and Zdrahal 1998) which was reformulated in terms of the IBROW constructs. The resulted library emphasizes the role of adaptive reusable components and makes explicit the adaptation process. In addition, the pre-existing library was applied to 5 application domains and the authors reported an average improvement in the design process by a factor of 10(Motta and Zdrahal 1998).

The **HPKB** project[7] aims at fostering the development of technologies that can increase the rate at which we can write knowledge bases(Cohen et al. 1998). An important issue in the context of this project is to find out to what extent reuse of prior knowledge is a productivity gain for the development of a new system. The metrics and findings of the study presented in (Cohen et al. 1999) will be discussed in section 4 along with their implication in systems engineering.

[5]An electronic version is accessible from http://www.isi.edu/isd/ontosaurus.html

[6]More on these technologies can be found in the project's home page: http://www.swi.psy.uva.nl/projects/IBROW3/home-ibrow.html

[7]Electronically accessible from http://projects.teknowledge.com/HPKB/

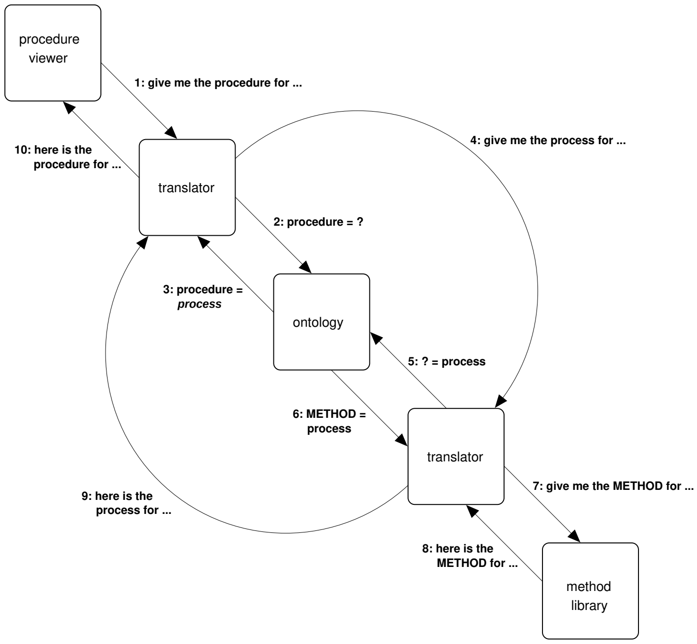


Figure 6: Interchange format example: the *procedure*, used by one tool is translated into the term, *method* used by the other via the ontology, whose term for the same underlying concept is *process*.

The use of a formal ontology, Process Interchange Format(**PIF**), in a knowledge sharing effort to facilitate the business process reengineering of supply chain activities is illustrated in (Polyak et al. 1998). PIF(Lee et al. 1998) acts as an interlingua between two separate tools used in the modelling and simulation of the proposed processes. The benefit of using the underlying process ontology was to capture domain knowledge in a generic way so that it can be reused across applications and shared among groups. In figure 6 we illustrate an interchange format example taken directly from (Uschold and Gruninger 1996).

In (Coad et al. 1997) several object-oriented patterns were surveyed. Their role was to guide novice analysts to a set of issues given by experienced analysts. For example, in figure 7 we show one of **Coad et.al's** patterns, a financial transaction pattern. As we can see from figure 7, this pattern includes the term 'subsequent transaction'. When browsing this pattern, a requirements engineer might then be prompted to ask the question: "are the sold items ever returned to the store?". This can assist in auditing and implementing the current version of a system description.

3.c.2 Reliability benefit

The **Comet**(Mark et al. 1992) system, developed in the Lockheed AI Center, supports the design of software systems specialised in the area of radar trackers by providing feedback for its users: when a user makes a change to a software module, `Comet` alerts the user about which other modules are affected and will

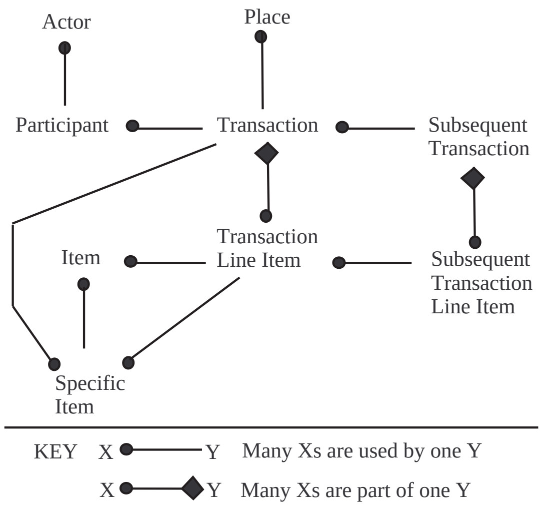


Figure 7: Part of *(Coad et.al. 1997)* definition of a financial transaction

require modification. The **Cosmos**(Mark et al. 1994) system, developed in the same site, supports engineering negotiation specialised in the area of actively controlled gimbal(i.e.: spacecraft components) and provides hardware designers with analyses that indicate the impact of a proposed design change. Both systems were aiming at developing knowledge bases by capturing the set of ontological commitments that define the interdependencies among key terms in the underlying ontology. Their role is to assess the impact of changes in their world and provide context-specific guidance to their users on what modules may be relevant to include in the design, and what design modifications will be required in order to include them(Mark et al. 1995). The key idea behind this work was to make use of the ontological commitment expressed by the underlying ontology in the system's development process. Ontologies must clearly express ontological commitments if the terms they specify are to be used without misinterpretation.

Repertory Grids were used in (Gaines and Shaw 1989) for detecting conflicts in terminology by comparing grids from different experts. The detection mechanism can reveal four classes of inconsistencies in terminology: consensus, correspondence, true conflict, and contrast. Experts are asked to identify dimensions along which items from their domain can be distinguished. The two extreme ends of these dimensions are recorded left and right of a grid. New items from the domain are categorized along these dimensions. This may lead to the discovery of new dimensions of comparisons from the expert which, in turn, will cause the grid to grow. For example, based on how an expert scaled some example houses, we can see from the repertory grid of figure 8 that the `ideal home` is closest to `1, Abraham Point, NW`. Once the dimensions stabilize, and a representative sample of items from the domain have been categorized, then the major distinctions and terminology of a domain have been defined. Inconsistencies are reported if the categorisations are significantly difference.

Ontological commitments were also the main topic in the **TOVE** project(EIL

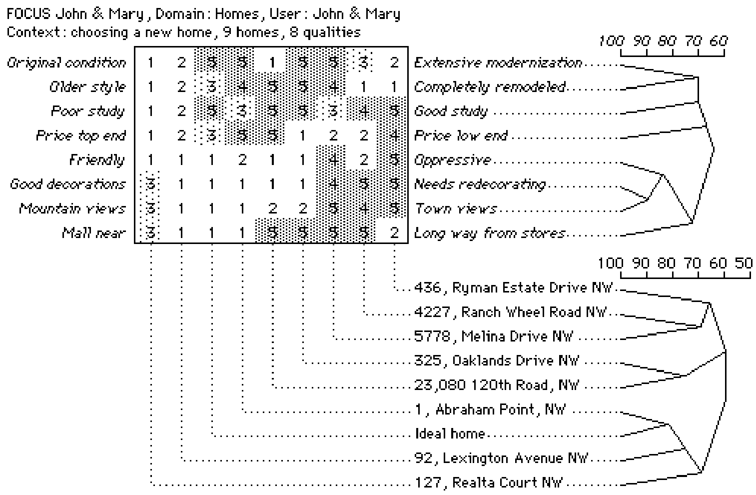


Figure 8: Repertory Grids generated from the WebGrid WWW server at: http://gigi.cpsc.ucalgary.ca

1995) but studied from a different angle: use them to define an ontology's competence. That is, a set of queries that the ontology can answer. These questions, called competency questions, were used to evaluate the expressiveness of the ontology that is required to represent them and characterise their solutions(Gruninger and Fox 1995). They do not generate ontological commitments but are used to evaluate them. The TOVE ontologies are built on top of foundational theories such as situation calculus. Foundational theories provide the semantics for the ontology and their axioms serve as a basis for the implementation of competency questions.

In the **DISCOVER** project(Waterson and Preece 1999), the role of ontological commitment was further analysed and operationalised. The authors state that ontological commitment is a key issue for knowledge sharing and reuse and they applied existing verification techniques from the KBSs literature to check the commitment of a knowledge base to an ontology. In that project the role of the ontology was to act as a background body of knowledge against which a knowledge base can be validated.

The role that ontological commitment plays in the engineering of a system that adopts an ontology has motivated the work described in (Kalfoglou et al. 2000). The authors point out that the role anticipated by ontological axioms is rarely delivered: to restrict the possible interpretations ontological constructs could have. To operationalise this role and enforce it in an integrated development environment they invented a **multi-layered** architecture in which ontological axioms are separated from other ontological constructs included in a system that uses the underlying ontology. These are enforced to comply to the axiomatisation in order to verify the consistency of the system with respect to domain knowledge as explicitly represented in the underlying ontology(Kalfoglou and Robertson 1999b). Ultimately, this layered metaphor

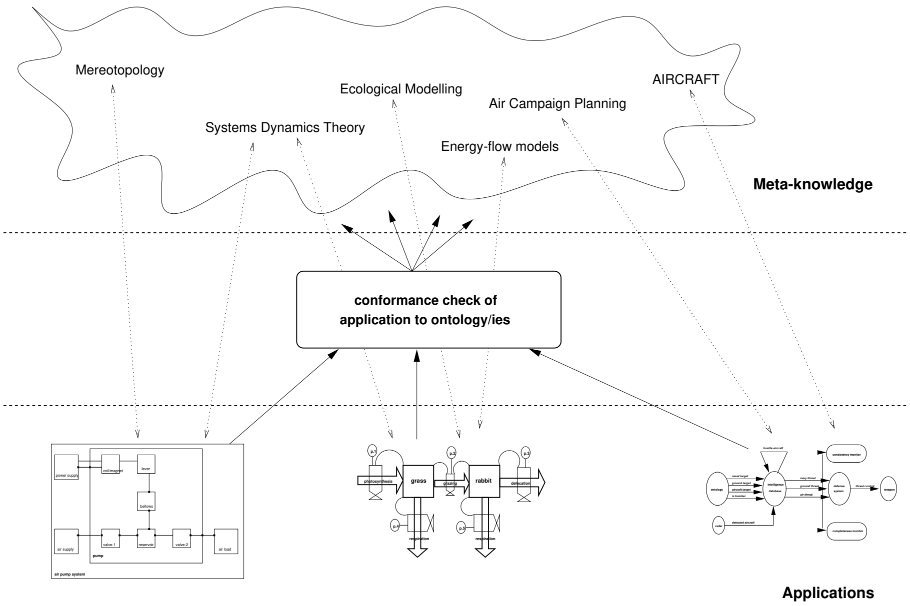


Figure 9: The multi-layer approach: it enforces the conformance check of an application to ontology/ies. Applications are using meta-knowledge constructs from various ontologies(mereotopology, system dynamics theory, etc.) in an integrated environment which enables checks on the use of those constructs against their axiomatised definitions.

can be extended to check the ontological axioms themselves against another set of axioms, meta-axioms, which could come from another ontology. This facilitates the conformance check of an application to ontology and can be extended to check dependencies among ontologies themselves. Moreover, it supports the integration of ontologies in applications while preserving their identity as being a separate layer in the multi-layer architecture. The approach is illustrated in figure 9.

3.d Organising meta-knowledge

In this section we sample supporting technologies and elaborate on issues that deal with organisational aspects of meta-knowledge. For example, in (Althoff et al. 1999a) an architecture which supports reuse of all kinds of experience from the software engineering domain is presented. The underlying representation formalism, called *Representation Formalism for Software Engineering Ontologies*(**REFSENO**(Tautz and von Wangenheim 1999)), supports the construction and manages the reuse of EBs in the life-cycle of a software project. In figure 10 an EB architecture based on **RESFENO** is presented. It is designed as a three-tiered architecture to accommodate the user access level(general purpose browser, data management(EB server)), and data storage. While *structure* is the main organisational task to develop a conceptualisation(e.g.: developing concepts like project), *record* is the main task to fill-in the EB with characterisations(instances, cases) like the project shown in figure 10 named Vesuv. In addition, *reuse* is the main task to apply knowledge stored in an EB within projects of the project organisation. This includes, for example, the provision of knowledge maps for identifying sources of tacit knowledge in terms of authorship, which at a later stage may be consulted for further guidance on the reuse process.

This approach(see figure 10) also provides knowledge maps for identifying sources of tacit knowledge in terms of authorship which later on may be consulted for further guidance on the reuse process.

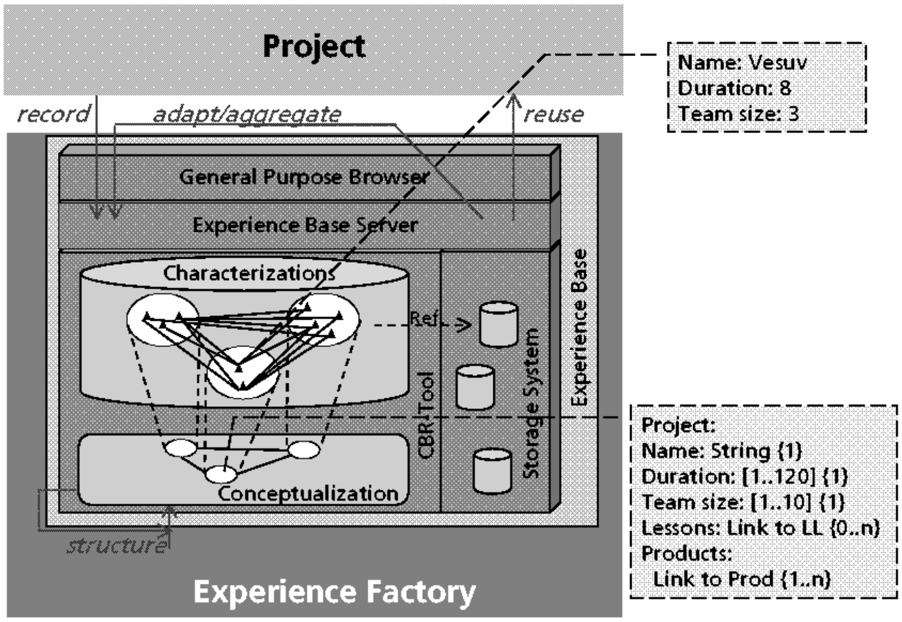


Figure 10: Main EF tasks and EB architecture.

Such knowledge can also be described on different levels of abstraction. With ongoing experience collection(see, for example, (Althoff et al. 1999c), (Basili et al. 1999)), these different abstraction levels can be compared with one another. In this sense the EF approach contributes to the question: "how meta-knowledge should be?" for domains where there are no rich background sources of prior knowledge.

Another important aspect in organising meta-knowledge is to handle changes over time. The majority of research in this area is focused on construction issues, like methodologies to support design(see, for example, (Fridman-Noy and Hafner 1997) and (Fernandez 1999) for ontology-design guidelines). However, coping with change and providing maintenance facilities requires a different approach and we cannot say that there exist commonly-agreed methodologies and guidelines for meta-knowledge maintenance. But we identify various research efforts that produced prototypes capable of serving, crudely speaking, maintenance purposes. For example, in (Domingue 1998) two systems were described: `Tadzebao` and `WebOnto`. The former aims to support a dialectical approach in ontology design and maintenance while the latter provides editing and browsing facilities. The goal of `Tadzebao` was to provide guidance for knowledge engineers around ongoing dialogues for designing ontologies. This can be used as a negotiation tool for proposed changes in a knowledge component with the additional flexibility that `Tadzebao` offers: the integration of discussion about an artefact and its representation in the same visual metaphor. An elaboration of this work is the 'discussion spaces' used in (Summer and Buckingham-Shum 1998) to support the negotiation of collaborative model construction.

The role that multiple agents play in ontology construction was investigated in (Swartout et al. 1996). The authors discuss the benefits of collaborative ontology construction, an important feature of this approach is the ability to support changes as the ontology evolves. To quote the authors:

> "[...]rather than regarding the ontology as a separate resource that is updated periodically, the development and extension of the ontology should occur as part of system development. In this way, the ontology becomes a 'living document' that is developed collaboratively and, at any given time, reflects the terminology that is shared by developers within an effort"

The environment which supports this evolution is the `Ontosaurus` browser[8] and an application of its use is presented in (Valente et al. 1999).

4 Pragmatic aspects in using meta-knowledge

In section 3 we pointed out benefits of using meta-knowledge and we listed applications that those benefits were realised. Here, we shift our focus to the costs of achieving these benefits. However, as we mentioned in the introduction, we currently lack extensive cost-benefit studies from the knowledge engineering literature. Therefore, we will use two different resources in our study: we will scrutinise the applications reported in the literature and critically review their benefits with respect to costs using empirical evidence. Also, we will consult the SE literature and borrow their metrics which we will apply to assess meta-knowledge benefits. Since experiences with procedural systems(mostly cited in the SE literature) may not apply to the construction of declarative systems(mostly cited in the AI literature) we will be very careful to make our case precise. We do not intend to conclusively discount the benefits of using meta-knowledge but we aim to provide certain points upon which we can forge research directions to solve problems with meta-knowledge. These will be discussed in section 5. This section also includes a brief discussion on reasons other-than-costs that could justify our investment in meta-knowledge.

4.a Construction cost

A meta-knowledge component is knowledge, and knowledge comes at a cost. Whether we build the meta-knowledge component from scratch or adopt it from others there is a 'cost of construction' following that investment. In the former case, that cost refers to pure construction issues such as, for example, choice and adoption or creation of a design methodology. In the latter case, there is

[8]Electronically accessible from the URL: http://www.isi.edu/isd/ontosaurus.html

an adoption of a pre-existent meta-knowledge component(for example, residing in a public library), and the cost is related to familiarisation and installation issues. Here we review the former case: building it from scratch.

Empirical evidence: We observe that this cost is often a drawback and those who had to afford it were skeptical for the effectiveness of the approach. For example, the developers of the ATOS system(Jones et al. 1995) wrote: "it was overambitious and unnecessary to develop a complete ontology of spacecraft operations[...]". Although their abandonment of developing a complete ontology was driven by architectural decisions the word 'overambitious' hints at the problem: complete ontologies are costly to build and require a substantial, time consuming and well coordinated effort.

This is apparent in broad meta-knowledge types like generic ontologies. For example, the CYC project where the developers had to devote more than 10 years of effort to build the CYC ontology(i.e.: initial CYC references date back to 1983:(Lenat et al. 1983)). However, as we look at other meta-knowledge types the situation changes: we have seen that domain-specific components are easier to build and are often developed incrementally as new knowledge regarding the domain of interest is acquired and pushed into the system. Examples of this are the **Ontosaurus** environment described in section 3.d and the reusable components described in (Motta 1999). In the same line are the EBs whose construction is incremental and they evolve along with the organisation that they belong to.

SE metrics: The COCOMO-II consortium(Abts et al. 1998) studies on 161 projects(Chulani et al. 1999) reveal that developing widely reusable components adds little to the overall project's efforts. For example, as we see from figure 11, the RUSE rating for *Extra High(XH)* reuse[9] across multiple product lines(i.e.: meta knowledge spans across multiple products), will cause an increase in effort by a factor of 1.56.

Develop for Reuse(RUSE)	Low(L)	Nominal(N)	High(H)	Very High(H)	Extra High(H)
Definition	None	Across project	Across program	Across product line	Across multiple product lines
1997 A-priori Values	0.89	1.00	1.16	1.34	1.56

Figure 11: COCOMO-II RUSE("Develop for Reuse") effort multiplier: Experts determined the RUSE a priori rating scale. Adopted from (Chulani et al. 1999)

In (Chulani et al. 1999) the authors presented a Bayesian analysis of these results and elaborated on a prediction model that merges sample data along with the experts' predictions. In that model, the value of RUSE rate was even lower, dropped to 24%. These results are encouraging in the sense that, at least for the procedural systems studied by the COCOMO-II consortium, building reusable components does not add vast amounts to the project's effort. However, we should be cautious in our interpretations because similar studies suggest that the cost of building these highly reusable components is considerably higher:

[9] In the COCOMO-II study several ratings are defined. While here we use the RUSE(Reuse) rating we point the interested reader to (Boehm 1995) for a comprehensive analysis.

one quarter to one half of the project's overall cost. This is recognised in the SE community as, literally speaking, the issue of *present-cost,future-reward.* In other words, we should expect to invest large amounts of cost to build meta-knowledge components that we anticipate to be beneficial in the foreseeable future.

4.b Reuse cost

We now turn to cost related with the reuse of meta-knowledge components. An alternative to building meta-knowledge from scratch is to reuse pre-existenting components. This is actually the mainstream in knowledge engineering and most of the applications reported in the literature follow this approach.

Empirical evidence: Recent experiments in ontology reuse show that particular types of ontologies are more useful than others. In the context of the HPKB(Cohen et al. 1999) project, it was found that very generic ontologies provide less support and are less useful than domain-specific ones. The latter scored a constant 60% rate of reuse in the HPKB study in contrast with the poor 22% rate of reuse scored by generic ontologies. However these results should not undermine their role in structuring meta-knowledge: "Although the rate of reuse of terms from very general ontologies may be significantly lower, the real advantage of these ontologies probably comes from helping knowledge engineers organise their knowledge bases along sound ontological lines"(Cohen et al. 1999).

Prior to reusing a meta-knowledge component an engineer has to locate the right component first and then familiarise herself with it. We have seen some efforts to facilitate the selection task, as for example the reusable libraries of ontologies(like `Ontolingua`) and in the same context ongoing work to produce frameworks which characterise and classify ontologies(Uschold and Jasper 1999). However, the familiarisation task remains a problem. Systems like **OntoSaurus** and **WebOnto**(described in section 3.d) aim to help the engineer find herself around.

One practical approach here is the EF where its EBs contain characterized and classified experienceware described in different levels of abstraction. The approach supports a goal-oriented, similarity-based retrieval that helps to find the right piece of knowledge in the right time and within a justifiable effort. Based on a continuous evaluation of the EB(Nick et al. 1999) the user herself decides whether or not more abstract knowledge is provided.

SE metrics: But, evidence from the SE literature shows that familiarisation involves a learning curve which must be traversed before a module can be adapted. The COCOMO-II study gives us some clues on the cost of customising a component: by the time you know enough to change a little of a component, you may as well have re-written 60% of it scratch(see figure 12). As mentioned above, the COCOMO-II results relate to the adaption cost of procedural systems. Hence, they may not apply to the declarative descriptions of system terminology. However, at the very least, these results caution us that just because we can access meta-knowledge, this does not necessarily mean that we can use it as a productivity tool. Meta-knowledge must be learnt prior to use and this learning curve may have a non-trivial impact on the overall cost.

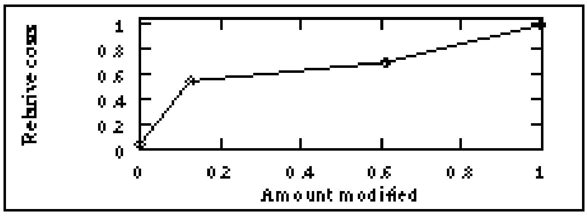


Figure 12: COCOMO-II: The cost of reuse with X% changes. The Y axis shows the costs of reuse divided by the cost of rebuilding from scratch.

4.c Maintenance cost

Empirical evidence: In the long run this cost might hinder further deployment of meta-knowledge. However, it is not easily predictable and quantifiable since there are various angles of viewing this problem. For instance, if we accept that meta-knowledge rarely stabilises then we should expect to include in our budget along with the cost of constructing, costs for maintaining the meta-knowledge component we use as well as the system which uses it. How common is meta-knowledge instability? We don't know since we have very little experience with the long-term use of large libraries of meta-knowledge. However, this is a debatable point(Menzies et al. 2000) and we find projects where meta-knowledge was deployed on the rationale that is was stable(i.e.: in parametric design ontology (Motta 1999)), and projects where this is not taken for granted as meta-knowledge is expected to change over time(i.e.: Aitken's HPKB experiment(Aitken 1998)). Another angle is the type of meta-knowledge we are dealing with. For example, if it is ontologies, despite the arguments made for instability, maintenance cost should be accounted for: to quote (Robertson 1998):

> "[...] the cost of producing an ontology is not just in inventing the domain-specific formal language but in maintaining it once the system is deployed, since perfect ontologies cannot be guaranteed. Over-commitment to perfecting an ontology causes failure either during development(through irreconcilable arguments over what the ontology should be) or after deployment(through inappropriate human interpretation of inference system inputs or outputs)"

On the other hand, other types of meta-knowledge tend to be easier to maintain: EFs and EBs are relatively easy to maintain since every time you add a new experience the EF is updated and reflects the current status in the field of work. Moreover, the supporting technology behind EF, the CBR technique, facilitates changes and retrieval/storage tasks. At the end, we can say that we have little evidence of the maintenance cost since there are no studies of long-term usage of meta-knowledge available. Further, we encountered constantly diverge opinions regarding the costs and ease of maintenance from various developers and users of meta-knowledge. This arises, mainly, from the high variety of meta-knowledge types(i.e.: maintenance cost of an ontology versus that of an EB). Hence, we rely on results drawn from SE and knowledge engineering studies:

SE/KE metrics: In (Hatton 1998) a study on the ease of fixing errors in hierarchies was presented. The author argues that inheritance confuses rather than clarifies maintenance. He continues by relating the increased maintenance times to the distributed nature of properties in an inheritance hierarchy. If Hatton's hypothesis is correct, then this SE result applies equally to any object-oriented hierarchy. If we accept that inheritance hierarchies are present in most of the meta-knowledge types identified in this paper, then we can equally accept the validity of this result in the context of meta-knowledge hierarchies. However, we should be very careful in making this assumption because, as the author acknowledges, the particular results do not distinguish between the effects of object-oriented hierarchies and the particulars of the implementation environments(Hatton 1998). For example, all the experiments were made by comparing C++ and C programs, whereas in the declarative systems used in meta-knowledge we found built-in features for maintaining hierarchies(i.e.: the Description Logics(see, for example, (Borgida 1995)) formalisms[10]).

In this area we also have results from studies on the behaviour of experts. We are interested in these results from the collaborative construction of meta-knowledge point of view rather than interested in emulating the experts themselves. Many researchers argue that experts disagree about even well-established features of their domain(see, for example, (Finkelstein et al. 1994), (Menzies and Clancey 1998)). Others studied the behaviour of experts(Shaw 1988) and found that they often held different views about a supposedly standard terminology in their field. Furthermore, in (Agnew et al. 1993) the authors state: "expert knowledge is comprised of context-dependent, personally-constructed, highly-functional but fallible abstractions". This suggests that we should routinely expect evolution of experts' views, especially in domains where there is disagreement on used terms.

However, we have to point here that in situations where there is a lack of consensus among the experts regarding the domain of interest then the principle of meta-knowledge does not apply by definition: meta-knowledge represents consensual and common-agreed terminology about a domain of interest. O'Leary uses this argument to point out that: "ontologies are chosen after a political decision has been made, therefore it is impossible to choose an ontology that maximises the utility of all agents in the process and the group."(O'Leary 1997). We agree with O'Leary's thesis and as a rule of thumb we can say that in domains where there is literally no consensus among the domain experts then building a meta-knowledge component is pointless. This, however, should not be interpreted as a guideline to build meta-knowledge components only when experts agree: this will rarely happen, as the studies described above suggest, therefore we have seen the most successful meta-knowledge stories coming from domains where the 'majority' of experts agree on used terms. The issue here is to find the right balance between commonly agreed terminology and usability of meta-knowledge. Guidelines from the ontology-building literature suggest this as one of the design principles: minimal ontological commitment(Gruber 1995). We should also mention that experience with task models in the KBS community indicates a broad degree of consensus with respect to the structure of KBS tasks, like diagnosis, parametric design, scheduling, etc. The key fac-

[10]Though, we do not have a comprehensive analysis of their maintenance cost benefits to cite.

tor here is the effective support for KBS development rather than achieving community-wide consensus, a goal of generic meta-knowledge components like broad ontologies.

4.d Purpose

Apart from the cost-related factors we discussed above, other issues emerge when we decide to use meta-knowledge. These are the level of formality, often related to the purpose of use, level of support, technical obstacles to overcome, etc.

In particular the level of formality is a traditional point of contention in many fields(see, for example, (Cleland and MacKenzie 1995) and (Shipman and Marshall 1999)). Despite the strong claims made by opponents of formality the sample systems we reviewed in section 3.c.2 shows that formal meta-knowledge can be operationalised which makes it suitable for enforcing automated consistency checking. On the other hand, this sort of use is not the mainstream and as the comparative review of (Uschold et al. 1999) points out, not a cost-effective approach: "[...]if it is application data without operational semantics, then it is a cost effective approach as the results with STEP schemata suggest; but is currently not yet mature enough for exchanging data with operational semantics and building fully automating translators is in general beyond the state of the art."

This may justify the wide usage of semi-formal or even informal meta-knowledge we see in the literature, which is mostly directed to deliver reuse and knowledge sharing rather than improving reliability[11]. The level of support for using meta-knowledge and technical obstacles which should be overcomed raised in experiments where the meta-knowledge components used were outsourced. For example, as the authors of the **Boeing**(section 4.b) experiment pointed out the translation activity involved was an intensive one and lack of automatic support is an important disadvantage. However, the situation changes when we look at other similar efforts. For instance, in the **AIRCRAFT** project, where most of the meta-knowledge used was provided by the developers themselves, no such claims for lack of support or translation problems were made.

5 Directions

We conclude the article by summarising issues that emerge from current research in meta-knowledge and speculate on prospective solutions to potential problems. A notable change in meta-knowledge research is that meta-knowledge is no longer a core theme of a single community(i.e.: AI). There is an increasing interest in meta-knowledge research and products from a wide variety of communities and industrial partners. If someone looks at the proceedings of recent KA workshops[12], major international AI conferences(IJCAI,AAAI,ECAI), CBR events(ICCBR,EWCBR), interdisciplinary-oriented conferences(see, for example, the SEKE series), and a plethora of journals, he will see a wide and versa-

[11]As an evidence of this claim we found that only 4 out of the 19 papers included in the volume edited by Guarino(Guarino 1998b) use formal meta-knowledge.

[12]Note that these AI-dominated events(Banff, European, and Pacific series) tend to be the barometer in ontology and PSM research.

tile range of meta-knowledge research issues to be tackled. While the expansion of meta-knowledge has been praised by many it also causes problems with the classification of meta-knowledge types and comprehension of its usage. For example, one of the problems we faced in this study was the diverse angles of viewing and tackling the same problems by different communities. However, we should highlight the emergence of 'classification' efforts such as the framework proposed in (Uschold and Jasper 1999) and of supporting technologies for organising meta-knowledge, such as the EFs, which can alleviate the situation.

A consequence of the versatile and intricate nature of meta-knowledge, along with its relatively young age, is the lack of metrics. For example, in our cost-effectiveness analysis in section 4 we had to rely on empirical evidence and 'borrow' SE metrics to draw our conclusions. We would like to have more knowledge engineering-specific metrics at our disposal and we anticipate to see more work on this in the near future. For example, recently we saw the first publicly available metric for reuse of terminologies in the context of ontologies(Cohen et al. 1999). Another example is the deployment of a goal-oriented measurement and evaluation approach, the GQM in knowledge bases(Nick et al. 1999).

Further to these, we also need tools that support the whole infrastructure for meta-knowledge deployment. Apart from guidelines and methodologies to aid initial construction, more attention should be paid to maintenance, ease of reuse and cost-effectiveness benefits in the long term. This is an open issue in meta-knowledge research and possible solutions are explored in (Uschold et al. 1999). In addition, the results of the two major projects, **HPKB** and **IBROW** are expected to contribute in this area. We also expect to see more EF-based approaches. One example here is the European project INRECA that bases its methodology for developing and maintaining CBR-based software systems on EFs(Bergmann et al. 1999).

In this study we were also able to identify types of meta-knowledge that have become popular during the years. These are domain-specific components that, empirically, have been proved cheaper to construct, easier to apply, and provide means for maintenance. This justifies, somehow, the shift of interest from very generic, broad meta-knowledge components, to domain-related components tailored to serve particular applications.

So far, in this closing stage we didn't mention the question posed in the title: whether meta-knowledge is a panacea or undelivered promise for systems design. In our view, meta-knowledge is neither a panacea nor an undelivered promise. These two characterisations represent, probably, the two extremes in defining meta-knowledge. We place meta-knowledge, literally speaking, in the middle of these two extremes and towards the panacea end. This is because, we saw evidence in the systems reported in sections 3.b to 3.d, that meta-knowledge can improve systems design in such areas as knowledge sharing and reuse and contribute to enhance their reliability by consistency checking. In particular, we saw that meta-knowledge had an impact in system design by reducing production costs, shortening development times and communicating context among applications and across organisations. It also improved the quality of the resulted systems with respect to verification of their correctness against domain knowledge.

On the other hand, in the 'pragmatics' section we identified potential drawbacks that might undermine the benefits that meta-knowledge claims to deliver, thus becoming an undelivered promise. The most important being, the considerably high cost of constructing a meta-knowledge component from scratch, the lengthy learning curve which has to be traversed in order to become familiar with meta-knowledge before integrating it in the system, the lack of rigid maintenance strategies, and the dearth of metrics for assessing meta-knowledge. However, as we highlighted in this section, potential solutions to these problems have begun to emerge which will place meta-knowledge more towards the panacea end rather than the undelivered promise one.

Acknowledgements

The idea for this study emerged from a panel debate among the authors that took place in the SEKE'99 conference (Menzies 1999a). Thanks to Daniela Carbogim, Virginia Brilhante and David Robertson for their comments on an early draft. The research described in this paper is supported by a European Union Marie Curie Fellowship (programme: Training and Mobility of Researchers) for the first author and partially supported by NASA through cooperative agreement NCC 2-979 for the second author.

References

C. Abts, B. Clark, S. Chulani, E. Horowitz, R. Madachy, D. Reifer, R. Selby, and B. Steece. *COCOMO-II: Model Definition Manual.* Center for Software Engineering, USC, CA, USA, 1998.

N.M. Agnew, K.M. Ford, and P.J. Hayes. Expertise in Context: Personally Constructed, Socially Elected, and Reality-Relevant? *International Journal of Expert Systems*, 7(1), 1993.

S. Aitken. Extending the HPKB-Upper-Level Ontology: Experiences and Observations. In A. Gomez-Perez and R. Benjamins, editors, *Proceedings of Workshop on Applications of Ontologies and Problem Solving Methods, ECAI'98, Brighton, England*, August 1998.

C. Alexander. *Notes on Synthesis of Form.* Harvard University Press, 1964.

K-D. Althoff, A. Birk, S. Hartkopf, W. Muller, M. Nick, D. Surmann, and C. Tautz. Managing Software Engineering Experience for Comprehensive Reuse. In *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering, SEKE'99, Kaiserslauten, Germany*, pages 10–19, June 1999a.

K-D. Althoff, F. Bomarius, W. Muller, and M. Nick. Using Case-Based Reasoning for Supporting Continuous Improvement Processes. In P. Perner, editor, *Proceedings of German Workshop on Machine Learning(FGML'99), Leipzig, Germany*, IBal Report, pages 54–61. Institute for Image Processing and Applied Informatics, 1999b. ISSN: 1431-2360.

K.-D. Althoff, M. Nick, and C. Tautz. Improving organizational memories through user feedback. In F. Bomarius, editor, *Proceedings of the Learning Software Organizations(LSO'99) workshop, Kaiserslauten, Germany*, pages 27–44, June 1999c.

J.R. Anderson. *Cognitive Psychology and its Implications.* W.H. Freeman and Company, 1990. 3rd edition.

R.V. Basili, F. Shull, and F. Lanubile. Building Knowledge through Families of Experiments. *IEEE Transactions on Software Engineering*, 25(4):456–473, August 1999.

V. Basili, G. Caldiera, and D. Rombach. Experience Factory. In J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 469–476. John Wiley & Sons, 1994.

V.R. Basili and H.D. Rombach. The TAME Project: Towards Improvement Oriented Software Environments. *Transactions on Software Engineering*, SE-14(6):758–773, June 1988.

R. Benjamins and D. Fensel. Special issue on problem solving methods. *International Journal of Human Computer Studies*, 49(4), 1998.

R. Bergmann and K-D. Althoff. Methodology for Building Case-Based Reasoning Applications. In M. Lenz, B. Bartsch-Sporl, H-D. Burkhard, and S. Wess, editors, *Case-Based Reasoning Technology - From Foundations to Applications*, number 1400 in LNAI, pages 299–326. Springer Verlag, 1998.

R. Bergmann, S. Breen, M. Goker, M. Manago, and S. Wess. Developing Industrial Case-Based Reasoning Applications - the INRECA methodology. Springer Verlag, 1999.

B. Boehm. Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. In J.D. Arthur and S.M. Henry, editors, *Annals of Software Engineering. Special Volume on Software Process and Product Management*, volume 1, pages 45–60. J.C. Baltzer AG, Science, Amsterdam, The Netherlands, 1995.

B. Boehm. Aims for Identifying Conflicts Among Quality Requirements. *IEEE Software*, 13(2), March 1996.

G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide.* Addison-Welsey, 1998. ISBN: 0-207-57168-4.

A. Borgida. Description Logics in data management. *IEEE Transactions on Knowledge and Data Engineering*, 7:671–682, 1995.

J. Breuker and W. Van de Velde (eds). *The CommonKADS Library for Expertise Modelling.* IOS Press, Netherlands, 1994.

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *A System of Patterns: Pattern-Oriented Software Architecture.* John Wiley & Sons, 1996.

G. Casady. Rationale in Practice: templates for Capturing and Applying Design Expertise. In T.P. Moran and J.M. Carroll, editors, *Design Rationale: Concepts, Techniques, and Use*, pages 351–372. Lawerence Erlbaum Associates, 1996.

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

S. Chulani, B. Boehm, and B. Steece. Bayesian Analysis of Empirical Software Engineering Cost Models. *IEEE Transactions on Software Engineering*, 25 (4):573–583, August 1999.

W. Clancey. The knowledge level reinterpreted: Modeling how systems interact. *Machine Learning*, 4(3/4):285–293, 1989.

G. Cleland and D. MacKenzie. Inhibiting Factors, Market Structure and the Industrial Uptake of Formal Methods. In *Workshop on Industrial-Strength Formal Specification Techniques, Boca Raton, FL, USA*, pages 46–60, Orlando(Florida) USA, April 1995.

P. Coad, D. North, and M. Mayfield. *Object Models: Strategies, Patterns, and Applications.* Prentice Hall, 1997.

B.N. Cocchiarella. *Conceptual Realism as a Formal Ontology*, pages 27–60. Formal Ontology. Kluwer Academic Publishers, August 1996. ISBN: 079234104X.

P. Cohen, V. Chaudhri, A. Pease, and R. Schrag. Does prior knowledge facilitate the development of knowledge-based systems? In *Proceedings of the Sixteenth National Conference on Artificial Intelligence, AAAI'99, Orlando, FL, USA*, pages 221–226, July 1999.

P. Cohen, R. Schrag, E. Jones, A. Pease, A. Lin, B. Starr, D. Gunning, and M. Burke. The DARPA High Performance Knowledge Bases project. *AI Magazine*, 19(4):25–49, 1998.

S. Cranefield and M. Purvis. UML as an Ontology Modelling Language. In *Proceedings of the IJCAI-99 Workshop on Intelligent Information Integration, Stockholm, Sweden*, August 1999.

W. Cunningham. The checks pattern language of information integrity. In J. Coplien and D. Schmidt, editors, *Pattern Languages of Program Design.* Addison-Wesley, 1995.

R. Davis and R. Smith. Negotiation as a metaphor for distributed problem solving. In Morgan Kaufmann, editor, *Readings in Distributed Artificial Intelligence*, 1998.

J. Domingue. Tadzebao and WebOnto: Discussing, Browsing, and Editing Ontologies on the Web. In *Proceedings of the 11th Knowledge Acquisition, Modelling and Management Workshop, KAW'98, Banff, Canada*, April 1998.

S. Easterbrook. Handling conflicts between domain descriptions with computer-supported negotiation. *Knowledge Acquisition*, 3:255–289, 1991.

Enterprise Integration Laboratory. EIL. TOVE Project, University of Toronto, Canada. available from http://www.ie.utoronto.ca/EIL/tove/ontoTOC.html, July 1995.

H. Eriksson, Y. Shahar, S.W. Tu, A. Puerta, and M. Musen. Task Modeling with Reusable Problem-Solving Methods. *Artificial Intelligence*, 79(2):293–326, 1995.

M. Fernandez. Overview for Methodologies for Building Ontologies. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods(KRR5), Stockholm, Sweden*, August 1999. Available from: http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-18/.

A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multi-perspective specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578, 1994. also as a Research Report DoC 93/2.

N. Fridman-Noy and C.D. Hafner. The State of the Art in Ontology Design: A Survey and Comparative Review. *AI Magazine*, 18(3):53–74, 1997.

B. Gaines and M. Shaw. Comparing the Conceptual Systems of Experts. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'89, Detroit, USA*, pages 633–638, 1989.

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

Y. Gil and E. Melz. Explicit Representations of Problem-Solving Strategies to Support Knowledge Acquisition. In *Proceedings of the 13th National Conference on Artificial Intelligence, AAAI' 96, Portland, OR, USA*, 1996.

Y. Gil and M. Tallis. A script-based approach to modifying knowledge bases. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97, Rhode Island*, 1997.

T. Gruber and G. Olsen. An ontology for engineering mathematics. In J. Doyle, P. Torasso, and E. Sandewall, editors, *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning, San Mateo, CA, USA.*, pages 258–269, 1994.

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

M. Gruninger and M.S. Fox. Methodology for the Design and Evaluation of Ontologies. In *Proceedings of the IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing, Montreal, Quebec,Canada*, August 1995.

N. Guarino. Formal Ontology and Information Systems. In N. Guarino, editor, *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems, FOIS'98, Trento, Italy*, pages 3–15. IOS Press, June 1998a.

N. Guarino, editor. *Formal Ontology In Information Systems*, Frontiers in Artificial Intelligence and Applications, June 1998b. IOS Press. ISBN: 90-5199-399-4.

L. Hatton. Does OO Sync with How we Think? *IEEE Software*, 15(3):46–54, June 1998.

M. Jones, J. Wheadon, D. Whitgift, M. Niezatte, M. Timmermans, R. Rodriquez, and R. Romero. An agent-based approach to spacecraft mission operations. In N.J.I. Mars, editor, *Proceedings of 2nd International Conference on Knowledge Building and Knowledge Sharing(KB&KS'95), Twente, The Netherlands*, pages 259–269. IOS Press, April 1995.

J. Josephson, B. Chandrasekaran, M. Carroll, N. Iyer, B. Wasacz, and G. Rizzoni. Exploration of Large Design Spaces: an Architecture and Preliminary Results. In *Proceedings of the 15th National Conference on Artificial Intelligence, AAAI'98, Madison, Wisconsin, USA*, July 1998.

Y. Kalfoglou and D. Robertson. A Case Study in Applying Ontologies to Augment and Reason about the Correctness of Specifications. In *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering, SEKE'99, Kaiserslauten, Germany*, pages 64–71, June 1999a. Also as: Research Paper No.927, Dept. of AI, University of Edinburgh.

Y. Kalfoglou and D. Robertson. Managing Ontological Constraints. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods(KRR5), Stockholm, Sweden*, http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-18/, August 1999b. Also as: Research Paper No.948, Dept. of AI, University of Edinburgh.

Y. Kalfoglou, D. Robertson, and A. Tate. Using Meta-Knowledge at the Application Level. *Journal of Artificial Intelligence Research, submitted*, 2000. Also as: Research Paper No.956, Dept. of AI, University of Edinburgh.

N. Kerth. *Caterpillar's Fate: A Pattern Language for Transformation from Analysis to Design.* Pattern Languages of Program Design. Addison-Welsey, 1995.

R.A. Kowalski and F. Sadri. Reconciling the situation calculus and event calculus. *Journal of Logic Programming*, 31:39–58, 1997.

J. Lee, M. Gruninger, Y. Jin, T. Malone, A. Tate, G Yost, and other members of the PIF working group. The PIF Process Interchange Format and framework. *Knowledge Engineering Review*, 13(1):91–120, February 1998.

B.D. Lenat, A. Borning, D. McDonald, C. Taylor, and S. Weyer. Knoesphere: Building Expert Systems With Encyclopedic Knowledge. In *proceedings of the IJCAI'83*, pages 167–169, 1983.

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

W. Mark, J. Dukes-Schlossberg, and R. Kerber. Ontological Commitment and Domain-Specific Architectures: Experience with Comet and Cosmos. In N.J.I.Mars, editor, *Proceedings of the 2nd International Conference on Knowledge Building and Knowledge Sharing(KB & KS'95), Twente, The Netherlands*, pages 33–45, April 1995.

W. Mark, J. Schlossberg, S. Tyler, and J. McGuire. Cosmos: A System for Supporting Engineering Negotiation. *Concurrent Engineering: Research and Applications*, 2:173–182, 1994.

W. Mark, S. Tyler, J. McGuire, and J. Schossberg. Commitment-Based Software Development. *IEEE Transactions on Software Engineering*, 18(10):870–884, October 1992.

D. Marques, G. Dallemagne, G. Kliner, J. McDermott, and D. Tung. Easy programming: empowering people to build their own applications. *IEEE Expert*, pages 16–29, June 1992.

T. Menzies. OO Patterns: Lessons from Expert Systems. *Software Practice and Experience*, 27(12):1457–1478, December 1997.

T. Menzies. Towards situated knowledge acquisition. *International Journal of Human-Computer Studies*, 49:867–893, 1998.

T. Menzies. Knowledge Maintenance Heresies: Meta-Knowledge Complicates KM. In *Proceedings of the 11th International Conference in Software Engineering and Knowledge Engineering,SEKE'99,Kaiserslauten, Germany*, June 1999a.

T. Menzies. Knowledge Maintenance: the state of the art. *The Knowledge Engineering Review*, 14(1):1–46, February 1999b.

T. Menzies, K-D. Althoff, Y. Kalfoglou, and E. Motta. Issues with Meta-Knowledge. *International Journal of Software Engineering and Knowledge Engineering(to appear)*, 10(4), August 2000.

T. Menzies and B. Clancey. Special Issue on Situated Cognition. *International Journal of Human-Computer Studies*, 49, 1998. Editorial.

T.P. Moran and J.M. Carroll. *Design Rationale: Concepts, Techniques, and Use.* Lawerence Erlbaum Associates, 1996. ISBN: 0-8058-1567-8.

E. Motta. *Reusable Components for Knowledge Models: Case Studies in Parametric Design Problem Solving*, volume 53 of *Frontiers in Artificial Intelligence and Applications.* IOS Press, 1999. ISBN: 1-58603-003-5.

E. Motta, D. Fensel, M. Gaspari, and R. Benjamins. Specifications of Knowledge Components for Reuse. In *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering, SEKE'99, Kaiserslauten, Germany*, pages 36–43, June 1999.

E. Motta and Z. Zdrahal. An approach to the organization of a library of problem solving methods which integrates the search paradigm with task and method ontologies. *International Journal of Human-Computer Studies*, 49 (4):437–470, 1998.

A. Newell. Reflections on the Knowledge Level. *Artificial Intelligence*, 59:31–38, February 1993.

M. Nick, K-D. Althoff, and C. Tautz. Facilitating the Practical Evaluation of Knowledge-Based Systems and Organizational Memories Using the Goal-Question-Metric Technique. In *Proceedings of the 12th Knowledge Acquisition, Modelling and Management Workshop(KAW'99), Banff, Canada*, pages 16–21, October 1999.

H. Nissen, A. Jeusfeld, M. Jarke, G. Zemanek, and H. Huber. Requirements Analysis from Multiple Perspectives: Experiences with Conceptual Modelling Technology. *IEEE Software*, 13(2), March 1996.

D. O'Leary. Impediments in the use of explicit ontologies for KBS development. *International Journal of Human-Computer Studies*, 46(2):327–337, 1997.

K. Orsvarn. Principles for Libraries of Task Decomposition Methods - Conclusions from a Case-study. In N. Shadbolt, K. O'Hara, and G. Schreiber, editors, *Proceedings of the 9th European Workshop on Knowledge Acquisition, Modelling and Management(EKAW'96)*, volume 1076 of *Lecture Notes in Artificial Intelligence*, pages 48–65. Springer-Verlag, 1996.

J. Pearl and R. Korf. Search Techniques. *Annual Review of Computer Science*, 2:451–467, 1987.

J. Pinto and R. Reiter. Temporal reasoning in logic programming: A case for the situation calculus. In *Proceedings of the 10th International Conference on Logic Programming, Budapest, Hungary*, June 1993.

P. Politakis. *Empirical Analsis for Expert Systems.* Pitman, 1985.

S. Polyak, J. Lee, M. Gruninger, and C. Menzel. Applying the Process Interchange Format(PIF) to a Supply Chain Process Interoperability Scenario. In A. Gomez-Perez and R. Benjamins, editors, *Proceedings of Workshop on Applications of Ontologies and Problem Solving Methods, ECAI'98, Brighton, England*, August 1998.

B. Ramesh and V. Dhar. Supporing Systems Development by Capturing Deliberations During Requirements Engineering. *IEEE Transactions on Software Engineering*, 18(6):498–510, June 1992.

H. Rittel and M. Webber. Dilemmas in a general theory of planning. *Policy Sciences*, 4:155–169, 1973.

D. Robertson. Pitfalls of Formality in Early System Design. In *Proceedings of the 1998 ARO/NSF Monterey Workshop on Increasing the Practical Impact of Formal Methods for Computer-Aided Software Development, Carmal, California*, 1998.

P. Rosenbloom, J. Laird, and A. Newell. *The SOAR Papers.* The MIT Press, 1993.

D.A. Schon. *The Reflective Practitioner.* Harper Collins, 1983.

A. Schreiber, B. Wielinga, H. Akkermans, W. VanDeVelde, and deHoog.H. CommonKADS: A Comprehensive Methodology for KBS Development. *IEEE Expert*, 9(6):28–37, 1994.

N. Shadbolt and K. O'Hara. Model-based Expert Systems and the Explanations of Expertise. In P.J. Feltovich, K.M. Ford, and R.R. Hoffman, editors, *Expertise in Context*, chapter 13, pages 315–337. MIT PRess, 1997.

M. Shaw. Validation in a Knowledge Acquisition System with Multiple Experts. In *proceedings of the International Conference on 5th Generation Computer Systems*, pages 1259–1266, 1988.

F.M. Shipman and C.C. Marshall. Formality Considered Harmful: Experiences, Emerging Themes, and Directions on the Use of Formal Representations in Interactive Systems. *Computer Supported Cooperative Work*, 8:333–352, 1999.

H. Simon. *The Science of the Artificial.* MIT Press, 1969.

P. Simons. *Parts: A Study in Ontology*, pages 5–128. Oxford: Clarendon Press, 1987.

T. Summer and S. Buckingham-Shum. From Documents to Discourse: Shifting Conceptions of Scholarly Publishing. In *proceedings of the CHI'98: Human Factors in Computing Systems, Los Angeles, CA, USA*, pages 95–102. ACM Press, 1998.

B. Swartout, R. Patil, K. Knight, and T. Russ. Toward Distributed Use of Large-Scale Ontologies. In *Proceedings of the 10th Knowledge Acquisition, Modeling and Management Workshop(KAW'96),Banff,Canada*, November 1996.

C. Tautz and C.G. von Wangenheim. REFSENO: A Representation Formalism for Software Engineering Ontologies. In *Proceedings of the 5th German Conference on Knowledge-Based Systems(XPS99), Wurzburg, Germany*, pages 61–71, 1999.

M. Uschold. Knowledge level modelling: concepts and terminology. *The Knowledge Engineering Review*, 13(1):5–29, February 1998.

M. Uschold, P. Clark, M. Healy, K. Williamson, and S. Woods. An Experiment in Ontology Reuse. In *Proceedings of the 11th Knowledge Acquisition Workshop, KAW98, Banff, Canada*, April 1998a.

M. Uschold and M. Gruninger. Ontologies: principles, methods and applications. *The Knowledge Engineering Review*, 11(2):93–136, November 1996.

M. Uschold and R. Jasper. A Framework for Understanding and Classifying Ontology Applications. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods(KRR5), Stockholm, Sweden*, August 1999. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-18/.

M. Uschold, R. Jasper, and P. Clark. Three Approaches for Knowledge Sharing: A Comparative analysis. In *Proceedings of the 12th Knowledge Acquisition, Modelling and Management Workshop, KAW'99, Banff, Canada*, October 1999.

M. Uschold, M. King, S. Moralee, and Y. Zorgios. The enterprise ontology. *The Knowledge Engineering Review*, 13(1), February 1998b. Also available as AIAI-TR-195 from AIAI, University of Edinburgh.

A. Valente, T. Russ, R. MacGrecor, and W. Swartout. Building and (Re)Using an Ontology for Air Campaign Planning. *IEEE Intelligent Systems*, 14(1): 27–36, January 1999.

P. van der Vet and N. Mars. Bottom-Up Construction of Ontologies. *IEEE Transactions on Knowledge and Data Engineering*, 10(4):513–526, 1998.

C. von Wangenheim, A. von Wangenheim, and R.M. Barcia. Case-Based Reused of Software Engineering Measurement Plans. In *proceedings of the 10th International Conference on Software Engineering and Knowledge Engineering(SEKE'98), San Fransisco, CA, USA*, pages 193–200, June 1998.

A. Waterson and A. Preece. Verifying Ontological Commitment in Knowledge-based Systems. *Knowledge-Based Systems*, 12:45–54, April 1999.

B. Wielinga, A. Schreiber, and J. Breuker. KADS: a Modeling Approach to Knowledge Engineering. *Knowledge Acquisition*, 4:1–162, January 1992.

T. Winograd. *Bringing Design to Software.* Addison-Wesley, 1996. ISBN: 0-201-85491-0.