Applications of Ontologies in Software Engineering

Hans-Jörg Happel[*)] and Stefan Seedorf[†)]

*) FZI Forschungszentrum Informatik
Forschungsgruppe Information Process Engineering (IPE)
Haid-und-Neu-Str. 10-14, 76137 Karlsruhe, Germany
happel@fzi.de

†) Universität Mannheim
Lehrstuhl für Wirtschaftinformatik III, Schloss, 68131 Mannheim, Germany
seedorf@uni-mannheim.de

Abstract. The emerging field of semantic web technologies promises new stimulus for Software Engineering research. However, since the underlying concepts of the semantic web have a long tradition in the knowledge engineering field, it is sometimes hard for software engineers to overlook the variety of ontology-enabled approaches to Software Engineering. In this paper we therefore present some examples of ontology applications throughout the Software Engineering lifecycle. We discuss the advantages of ontologies in each case and provide a framework for classifying the usage of ontologies in Software Engineering.

1 Introduction

The communities of Software Engineering and Knowledge Engineering share a number of common topics [1]. While Software Engineering research has been continuously striving towards a higher degree of abstraction and emphasizing software modeling during the last decade, the Knowledge Engineering community has been eager to promote several modeling approaches in order to realize the vision of the semantic web [2].

With the advent of web-based software and especially web services, the overlap becomes even more evident. However, both communities mostly live in their own worlds. The number of forums for discussing synergies is still relatively small (e.g. SWESE[1], SEKE[2] and W3C[3]) although growing steadily.

The discussion on integrating Software and Knowledge Engineering approaches tends to be academic, focusing on aspects like meta-modeling, thereby neglecting important aspects such as applicability and providing little guidance for software engineers. Further, both are cultivating their own understanding of central concepts, making it difficult for members of each community to grasp the concepts of the other one. To overcome this gap, we review potential benefits the Software Engineering

[1] http://www.mel.nist.gov/msid/conferences/SWESE/
[2] http://www.ksi.edu/seke/
[3] http://www.w3.org/2001/sw/BestPractices/

community can achieve by applying ontologies in various stages of the development lifecycle in this paper.

The intended contribution of this paper is threefold. We first provide a concise description of various ontology-based approaches in Software Engineering, ordered by their position in the Software Engineering lifecycle (chapter 2). Second, we propose a framework which allows us to classify the different approaches (chapter 3). Finally, we try to derive some generic advantages of ontologies in the context of Software Engineering.

1.1 Software Engineering

Software engineering is the "application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" [3]. Although the claim of software development being an engineering discipline is subject to ongoing discussions, there is no doubt that it has undergone fundamental changes during the last three decades. This assertion holds true both for emergence of new technology and sophistication of methodology.

In order to cope with the complexity inherent to software, there has been a constant drive to raise the level of abstraction through modeling and higher-level programming languages. For example, the paradigm of model-driven development proposes that the modeling artifacts are "executable", i.e. through automated validation and code generation as being addressed by the OMG Model Driven Architecture (MDA) [4]. However, many problems have only partially been solved including component reuse, composition, validation, information and application integration, software testing and quality. Such fundamental issues are the motivation for new approaches affecting every single aspect in Software Engineering. Within this paper, we will thus restrain the scope of interesting applications and techniques to ontologies.

1.2 Knowledge Engineering

The engineering of knowledge-based systems is a discipline which is closely related with Software Engineering. The term Knowledge Engineering is often associated with the development of expert-systems, involving methodologies as well as knowledge representation techniques. Since its early days the notion of "ontology" in computer science has emerged from that discipline, giving rise to Ontology Engineering [5], which we focus on in this paper.

Despite sharing the same roots, ontologies emphasize aspects such as inter-agent communication and interoperability [6]. In computer science, the concept "ontology" is interpreted in many different ways and concrete ontologies can vary in several dimensions, such as degree of formality, authoritativeness or quality. As proposed by Oberle [7], different kinds of ontologies can be classified according to purpose, specificity and expressiveness. The first dimension ranges from application ontologies to reference ontologies that are primarily used to reduce terminological ambiguity among members of a community. In the specificity dimension, Oberle distinguishes generic (upper level), core and domain ontologies. Domain ontologies are specific to a universe of discourse, whereas generic and core ontologies meet a higher level of generality. According to the expressiveness of the formalism used, one can further distinguish lightweight and heavyweight ontologies.

Due to the emergence of the “semantic web” vision ontologies have been attracting much attention recently. Along with this vision, new technologies and tools have been developed for ontology representation, machine-processing, and ontology sharing. This makes their adoption in real-world applications much easier. While ontologies are about to enter mainstream Software Engineering practices, their applications in Software Engineering are manifold, which increases terminological confusion. We therefore try to alleviate some of the confusion by providing a framework for categorizing potential uses of ontologies in Software Engineering.

2 Ontologies in the Software Engineering Lifecycle

In this chapter, we will present concrete approaches for using ontologies in the context of Software Engineering. The presentation will be in the order of appearance in the Software Engineering lifecycle. Each approach will be described concerning the general problem it tries to solve. It is followed by a short description of the approach and the assumed advantages of ontologies.

2.1 Analysis and Design

Within software analysis and design, two main areas of application are identified: First, requirements engineering can benefit from ontologies in terms of knowledge representation and process support. Second, component reuse is chosen as a potential application area during design.

2.1.1 Requirements engineering

Problem addressed: The phase of requirements engineering deals with gathering the desired system functionality from the customers. Since the involved software engineers are often no domain experts, they must learn about the problem domain from the customers. A different understanding of the concepts involved may lead to an ambiguous, incomplete specification and major rework after system implementation. Therefore it is important to assure that all participants in the requirements engineering phase have a shared understanding of the problem domain. Moreover, change of requirements needs to be considered because of changing customer’s objectives.

Description of approach: An ontology can be used for both, to describe requirements specification documents [8, 9] and formally represent requirements knowledge [10, 11]. In most cases, natural language is used to describe requirements, e.g. in the form of use cases. However, it is possible to use normative language or formal specification languages which are generally more precise and pave the way towards the formal system specification. Because the degree of expressiveness can be adapted to the actual needs, ontologies can cover semi-formal and structured as well as formal representation [11].

Further, the “domain model”[4] represents the understanding of the domain under consideration, i.e. in the form of concepts, their relations and business rules. In its simplest form, a glossary may serve as a basis for a domain model. However, it can be formalized using a conceptual modelling language such as the UML. Moreover, the problem domain can be described using an ontology language, with varying degrees formalization and expressiveness.

Advantages of ontologies: In contrast to traditional knowledge-based approaches, e.g. formal specification languages, ontologies seem to be well suited for an evolutionary approach to the specification of requirements and domain knowledge [11]. Moreover, ontologies can be used to support requirements management and traceability [8, 10]. Automated validation and consistency checking are considered as a potential benefit compared to semi-formal or informal approaches providing no logical formalism or model theory. Finally, formal specification may be a prerequisite to realize model-driven approaches in the design and implementation phase.

2.1.2 Component reuse

Problem addressed: Modern Software Engineering practices advise developers to look for components that already exist when implementing functionality, since reuse can avoid rework, save money and improve the overall system quality. Usually, this search for reusable components takes place after the analysis phase, when the functional requirements are settled [12]. Since most reuse repositories are limited to a plain syntactical key-word based search, they are suffering from low precision (due to homonyms) and low recall (due to synonyms) [13].

Description of approach: Ontologies can help here to describe the functionality of components using a knowledge representation formalism that allows more convenient and powerful querying [14]. One approach implementing this is the KOntoR system, that allows to store semantic descriptions of components in a knowledge base and run semantic queries on it (using the SPARQL language).

Advantages of ontologies: Compared to traditional approaches, ontologies provide two advantages in this scenario. First, they help to join information that normally resides isolated in several separate component descriptions. Second, it provides background knowledge (e.g. about the properties of a certain software license) that allows non-experts to query from their point of view (ask for a license that allows to modify source code).

2.2 Implementation

A critical step in the development process is moving from analysis and design to implementation. To this end, the way in which the problem domain is mapped to code has always been playing a pivotal role. The question arises how ontologies can be leveraged to narrow the gap between design and implementation. Two areas of interest are the overlaps of software modelling with ontology languages and the run-

[4] In Software Engineering, the terms domain model, ontology and CIM (Computation Independent Model) are sometimes used to describe the same thing [cf. 20].

time usage of ontologies in applications. We also look at techniques where ontologies support coding and code documentation.

2.2.1 Integration with Software Modelling Languages

Problems addressed: The current MDA-based infrastructure provides an architecture for creating models and metamodels, define transformations between those models, and managing metadata. Though the semantics of a model is structurally defined by its metamodel, the mechanisms to describe the semantics of the domain are rather limited compared to knowledge representation languages [cf. 15]. MDA–based languages do not have a knowledge-based foundation to enable reasoning. Other possible shortcomings include validation and automated consistency checking. However, this is addressed by the Object Constraint Language (OCL).

Description of approach: There are several alternatives for integrating MDA-based information representation languages and ontology languages, which are exemplified in [16]. Whereas some regard the UML as ontology representation language by defining direct mappings between language constructs [17], others employ the UML as modelling syntax for ontology development [18]. In most cases, MDA-compliant languages and RDF/OWL are regarded as two distinct technological spaces sharing a “semantic overlap” where synergies can be realized by defining bridges between them [19]. The Ontology Definition Metamodel (ODM) [20] is an effort to standardize the mappings between knowledge representation and conceptual modelling languages. It specifies a set of MOF metamodels for RDF Schema and OWL among others, informative mappings between those languages, and profiles for a UML-based notation.

Advantages of ontologies: Software modelling languages and methodologies can benefit from the integration with ontology languages such as RDF and OWL in various ways, e.g. by reducing language ambiguity, enabling validation and automated consistency checking [cf. 15]. Ontology languages provide better support for logical inference, integration and interoperability than MOF-based languages. UML-based tools can be extended more easily to support the creation of domain vocabularies and ontologies. Since ontologies promote the notion of identity, ODM and related approaches simplify the sharing and mediation of domain models.

2.2.2 Ontology as Domain Object Model

Problems addressed: Since a domain model is initially unknown and changes over time, a single abstraction and separation of concerns is considered feasible if not necessary [cf. 21]. Therefore a single representation of the domain model should be shared by all participants throughout the lifecycle to increase quality and reduce costs [22]. The mapping of a domain model to code should therefore be automatized to enable the dynamic use by other components and applications.

Description of approach: The programmatic access of ontologies and manipulation of knowledge bases using ontology APIs requires special knowledge by the developers. Therefore an intuitive approach for object-oriented developers is desirable [cf. 23]. This can be achieved by ontology tools that generate an API from the ontology, e.g. by mapping concepts of the ontology to classes in an object-oriented language. The generated domain object model can then be used managing models, inferencing, and querying. Tools supporting those features are already available today, e.g. [23] and [24].

Advantages of ontologies: The end-to-end use of ontologies in analysis and design as well as implementation is highly suitable for rapid application development [22]. Not only is this an intuitive way for object-oriented developers for managing ontologies and knowledge models, interoperability with other components or applications is improved as well. The use of a web-based knowledge representation format enables developers to discover sharable domain models and knowledge bases from internal and external repositories.

2.2.3 Coding Support

Problem addressed: In object-oriented software development, the concept of encapsulation demands the decoupling of the interface specification from its implementation in order to make requesting applications independent from internal modifications [25]. Nowadays, developers face a large number of frameworks and libraries they have to access through application programming interfaces (APIs). Thus, the documentation of APIs has become an important issue. Some IDEs like Eclipse use this information to enhance developer productivity by providing auto-completion of method calls. However, many operations (such as the connection to a database) require several calls to an API. While developers could benefit from formalized knowledge about the interrelations of method calls in the API in a similar way to auto-completion, there is currently no support for this.

Description of approach: The SmartAPI approach [26] suggests enriching APIs with semantic information. Since the semantics of string parameters like "username" or "password" is only clear for users, but not for machines, they must be annotated with the concept "database user name". The authors propose to store those annotations via a public web service to enable a collaborative knowledge acquisition effort. Besides the easier location of API interfaces and methods, the authors present how a suitable sequence of method calls can be automatically generated, given a desired goal state (like getting a database result set).

Advantages of ontologies: In the SmartAPI scenario, the main advantage of ontologies is that they provide a globally unique identifier for concepts. While at the programming level it is convenient to have a limited set of data "types" like strings, that can be used for multiple purposes, an ontology enables developers to annotate API elements with an unambiguous concept. A potential drawback is the extra-effort for modelling the semantic layer. In the case of APIs, this is partially eased since an initial modelling effort scales well with the estimated reuse. However, the question of incentives for someone to semantically describe an API still remains.

2.2.4 Code Documentation

Problem addressed: The maintenance of software systems is one of the most dominant activities in Software Engineering. However, programming languages as the default representation of knowledge in Software Engineering are badly suited for maintenance tasks. They describe knowledge in a procedural way and are rather geared towards the execution of code than towards the querying of knowledge [27].

Description of approach: So called "Software Information Systems" (SIS) [28, 29] were among the first approaches that applied description logics to Software Engineering problems. Their main goal was to improve the maintainability of large software systems by providing powerful query mechnisms. The LaSSIE system [28] for example consists of programming-language independent descriptions of software structures and an ontology that describes the problem domain of the software. Both can be manually connected to allow e.g. querying for all functions dealing with a certain domain object.

Advantages of ontologies: Here, ontologies provide a unified representation for both problem domain and source code, thus enabling easier cross-references among both information spheres. Moreover, it is easy to create arbitrary views on the source code (e.g. concerning a variable). Reasoning is applied to create those views, e.g. to find all places where a variable is accessed either directly or indirectly.

2.3 Deployment and Run-time

2.3.1 Semantic Middleware

Problem addressed: In modern three-tier architectures, the middleware layer lies in the focus of attention. Sophisticated middleware infrastructure like application servers shield a lot of complexity from the application developer, but creates challenging tasks for the administrator. Issues like interdependencies between modules or legal constraints make the management of middleware systems a cumbersome task.

Description of approach: In the context of his work in the area of semantic management of middleware [7], Oberle developed a number of ontologies for the formal description of concepts from component-based- and service-oriented development[5]. His goal is to support system administrators in managing server applications, e.g. by making knowledge about library dependencies explicit. The conceptualization of the ontology was driven by two objectives: to provide a precise, formal definition of some ambiguous terms from Software Engineering (like "component" or "service") as well as structures supporting the formalization of middleware knowledge (i.e. by modelling the dependencies of libraries, licenses etc.).

Advantages of ontologies: In this case, ontologies provide a mechanism to capture knowledge about the problem domain. So the semantic tools in this approach create an information space where knowledge, e.g. about library dependencies, can be stored. Reasoning can then be applied to reuse this knowledge for various purposes. Oberle provides a detailed qualitative analysis on the modelling effort for a number of use-cases [7].

2.3.2 Business Rules

Problem addressed: In most software systems, the "business logic" - i.e. the mechanisms implemented in software systems to comply with the business policies of

[5] http://cos.ontoware.org

a company - are hard-coded in programming languages. Thus, changes to the business logic of a software system require modifications to the source code, triggering the normal compilation and deployment cycle. Since many companies are facing flexible, frequently changing business environments nowadays, technologies are sought, that support a quick propagation of new business rules into the core software systems [30].
Description of approach: (Business) rule engines are a possible solution approach for this problem. The core idea is to untangle business logic and processing logic. The business logic is modelled declaratively with logical statements and processed by a rule engine. Similar to a reasoner, it applies inference algorithms to derive new facts on a knowledge base. Most rule engines forward-chain rules in the knowledge base to infer actions the system should take [31]. While business rule engines are available for quite some time, they can be regarded as "ontology-based" approaches towards Software Engineering since they run declarative knowledge on a special middleware. Also there are standardization efforts in place to enable interoperability between rule formalisms used by the industrial vendors and those proposed in the semantic web community[6]. A better integration of rules with available description logics formalisms [32] could also help to establish a "knowledge layer" in system architectures which is served by a special kind of middleware (i.e. inference engines).
Advantages of ontologies: The main advantage in this approach is the declarative specification of knowledge which tends to change frequently. Business rules that would be hard-coded in most current systems, can be changed more easily, because they are not buried implicitly in some source code, but explicitly stated in a formal language that can be presented in a user friendly way for editing.

2.3.3 Semantic Web Services

Problem addressed: Offering data and services via well-defined interface descriptions in the web is the core idea of "web services" [33]. While web services enable developers to combine information from different sources to new services in the first place[7], the actual composition process remains troublesome. First, it is rather difficult to find appropriate services, since most industry standards (e.g. WSDL) are purely syntactical. Thus, also the wiring of services has to be done manually, since an algorithm can not find out, whether a string output "credcardNo" of some service is appropriate as a string input value for "ccNumber" for another service.
Description of approach: The basic idea of the semantic web services effort is to add a semantic layer on top of the existing web service infrastructure [34]. Input parameters, functionality and return values are annotated semantically, such that - at least in theory - automatic discovery, matching and composition of service-based workflows. Several standards and frameworks like OWL-S [35] or WSMX [36] are currently under development.
Advantages of ontologies: In the case of semantic web services, ontologies provide the flexibility that is sought in dynamic scenarios. They can ensure discovery and interoperability in cases that were not anticipated by the initial developer, since semantic descriptions can be extended in the course of time. Even mediation among

[6] http://www.w3.org/2005/rules/
[7] e.g. „mash-ups“ - http://www.programmableweb.com/

services that have been developed independently and annotated with different ontologies could interoperate by defining mappings that are supported by most ontology languages.

2.4 Maintenance

2.4.1 Project Support

Problem addressed: In software maintenance workflows, several kinds of related information exists without an explicit connection. This is problematic, since a unified view could avoid redundant work and speed up problem solving. A bug resolution process for example usually involves the discovery and reporting of a bug (often into a bug tracking system), subsequent discussion inside a developer group, and finally changes in the code that hopefully resolve the bug. While the discussion on the mailing list and the code changes are clearly triggered by the bug report, their relation is not necessarily explicit and often kept separately. Since it is difficult to manage larger amounts of bugs without all existing context information, the lack of tool support may lead to delays in bug fixing and duplicate work or discussions.

Description of approach: Dhruv [37, 38] is a semantic-web enabled prototype to support problem-solving processes in web communities. The application scenario is how open source communities deal with bugs in their software under development. Ontologies help to connect the electronic communication (via forums and mailing lists) of the developers with bug-reports and the affected areas in the source code. Central concepts are the community (e.g. developers), their interactions and content (e.g. emails). The knowledge is codified in three kinds of ontologies: two “content” ontologies describe the structure of artefacts, i.e. a software ontology based on Welty’s work and a taxonomy of software bugs. Second, an ontology of interactions describes the communication flow among the developers. Third, a community ontology defines several roles that are involved in the problem solving process.

Advantages of ontologies: In the Dhruv system, ontologies primarily provide a layer to integrate data from different source into a unified semantic model. The combined data can then be used to derive additional information that was not stated explicitly in one of the single sources before. The author gives the example of classifying people into roles like “bug-fixer” or “core developer” [37, p. 173].

2.4.2 Updating

Problem addressed: Agile development practices like rapid prototyping have led to an acceleration of release cycles for software products. So, keeping one's application zoo up-to-date is a time consuming tasks that involves checking for new versions, downloading and installing them. Although lots of modern software programs come with auto-update functions, there is no general mechanism to cope with such problems in a platform independent way.

Description of approach: Dameron describes a framework for automatically updating the Protege ontology editor and its plug-ins [39]. Therefore he uses an extension of the DOAP ontology[8] and a python script that retrieves the most recent version number and a download URL by calling a web service that does reasoning.

Advantages of ontologies: The basic advantage of an RDF-based solution in contrast to e.g. describing the download information in XML is extensibility. Using an XML schema, all plug-in providers must provide their data in the specified format. In order to stay compatible to the update script, changes would have to be done centrally and distributed to all plug-in providers. Using an RDF ontology, every provider is free to add or subclass concepts from the initial version without being at risk to become incompatible.

2.4.3 Testing

Problem addressed: Software tests are an important part of quality assurance [3]. However, the writing of test cases is an expensive endeavour that does no directly yield business value. It is also not a trivial task, since the derivation of suitable test cases demands a certain amount of domain knowledge.

Description of idea: Ontologies could help to generate basic test cases since they encode domain knowledge in a machine processable format. A simple example for this would be regarding cardinality constraints. Since those constraints define restrictions on the association of certain classes, they can be used to derive equivalency classes for testing (see also [23]).

Advantages of ontologies: Ontologies may not be the first candidate for such a scenario, since there are formalisms like OCL that are specialized for such tasks. However, once domain knowledge is available in an ontology format anyway (e.g. due to one of the various other scenarios described in this paper), it might be feasible to reuse that knowledge.

3 Categorizing Ontologies in Software Engineering

The preceding section has presented a number of different approaches for using ontologies in the context of Software Engineering. In this chapter, we propose a simple classification scheme that allows a better differentiation among the various ideas. While the ordering according to their position in the Software Engineering lifecycle was suitable to provide a first roundtrip in the world of ontologies and Software Engineering, we think there should be more meaningful distinctions regarding their application. Common categorizations of ontologies rank them by their level of abstraction and their expressivity (see sec. 1). However, when trying to understand how ontologies can be applied in Software Engineering and what the benefits are in each case, this distinction does not help much.

The Ontology Driven Architecture (ODA) note at W3C merely served as a starting point to elaborate a systematic categorization of the approaches and to derive more clearly defined acronyms [cf. 15]. Rethinking the approaches described in section 2, and bearing in mind the basic properties of Software Engineering, we propose two dimensions of comparison to achieve a more precise classification. First, we distinguish the role of ontologies in the context of Software Engineering between

[8] http://usefulinc.com/doap/

usage at run-time and development time. Second, we look at the kind of knowledge the ontology actually compromises. Here, we distinguish between the problem domain that the software system tries to tackle itself, and infrastructure aspects to make the software or its development more convenient. Putting these two dimensions together, we end up with the matrix in figure 1. We see four basic areas there:

Ontology-driven development (ODD) subsumes the usage of ontologies at development time that describe the problem domain itself. Prime example are the approaches in the context of MDD, presented in sec. 2.2.1.

Ontology-enabled development (OED) also uses ontologies at development time, but for supporting developers with their tasks. For example, component search (sec. 2.1.2) or problem-solving support (sec. 2.4.1) can be put in here.

Ontology-based architectures (OBA) use an ontology as a primary artifact at run-time. The ontology makes up a central part of the application logic. Business rule approaches are an example for this kind of application.

Ontology-enabled architectures (OEA) finally, leverage ontologies to provide infrastructure support at the run-time of a software system. An example are semantic web services, where ontologies add a semantic layer on top of the existing web service descriptions, adding functionality for the automatic discovery, matching and composition of service-based workflows.

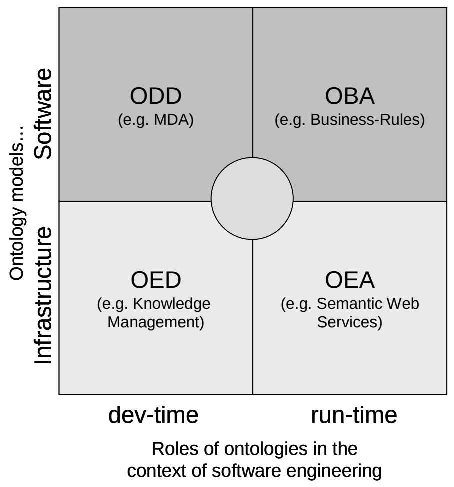
Ontology models...
Software
Infrastructure
ODD
(e.g. MDA)
OBA
(e.g. Business-Rules)
OED
(e.g. Knowledge Management)
OEA
(e.g. Semantic Web Services)
dev-time
run-time
Roles of ontologies in the context of software engineering

Figure 1: Usage categories for ontologies in Software Engineering

Although the four clusters seem to be quite distinct on first glance, there may be overlaps in some application areas. In particular, the classification scheme does not make any statement about clustering within or between the categorization groups. Indeed, in order to make the case for the large-scale reusability of ontologies, it is crucial to provide evidence for a broad range of applications. So one specific ontology might be useful in several of the described dimensions in parallel.

4 Advantages of Ontologies in Software Engineering

Since modeling ontologies is a tedious and costly task, it is always important to demonstrate the advantages one can gain by applying ontologies in Software Engineering. This is underlined by the fact that most of the formal foundations of ontologies have been in place for a long time, without enjoying a wide-spread adoption by software engineers.

So clearly the current advent of logic-based formalisms in the context of the semantic web effort is an important factor. Activities by the W3C and others have helped to flesh out standards like RDF or OWL that receive increasing attention by tool builders and users. In a certain sense, the importance of standardization here can be compared to the situation of visual modeling in Software Engineering before UML.

Another important factor is the flexibility of ontologies. With information integration as a major use case, ontologies are well-suited to combine information from various sources and infer new facts based on this. Also, the flexibility allows to extend existing ontologies very easy, thus fostering the reuse of existing work.

This is further promoted by the "web"-focus of current ontology approaches. Due to the fact that software systems also get increasingly web-enabled and must thus cope with data from heterogeneous sources that may not be known at development time, software engineers seek technologies that can help in this situation. Thus, experts in the field like Grady Booch are expecting semantic web technology to be one of the next big things in the architecture of web-based applications [40]. Also, the web makes it easier to share knowledge. Having URIs as globally unique identifiers, it is easy to relate one's ontology to someone else's conceptualization. This in turn encourages interoperability and reuse.

Regarding more Software Engineering-specific advantages, ontologies make domain models first order citizens. While domain models are clearly driving the core of every software system, their importance in current Software Engineering processes decreases after the analysis phase. The core purpose of ontologies is by definition the formal descriptions of a domain and thus encourages a broader usage throughout the whole Software Engineering lifecycle.

5 Conclusion

There is some discussion about how ontologies and Software Engineering fit together, and how both communities can learn from each other. As a contribution to this process, we presented a couple of approaches that use ontologies in a Software Engineering context in this paper. Therefore we selected examples from the entire Software Engineering lifecycle.

While studying the ontology applications, we found that the purpose of ontologies as well as the real benefits are hard to grasp without a proper framework for analysis. Thus, in section 3, we came up with an initial proposal for a better categorization, refining the notion of "Ontology Driven Architecture" (ODA) into four categories, that describe the usage of ontologies in different contexts. However, we think that this is just a preliminary step towards a better understanding of possible benefits of ontologies in Software Engineering.

Those benefits are a core part of a better understanding of ontologies in Software Engineering. Like Oberle [7] pointed out, ontologies demand additional modeling effort, that must pay off by savings at other places. Thus, we think that one key to promoting the advantages of ontologies is in a higher reuse of ontological knowledge across the Software Engineering lifecycle. While this may be partially conflicting with the presented approaches in detail, it would be interesting to perform a case study to what extent a single domain ontology could be leveraged across some of the presented works.

Acknowledgement

This work was supported by the BMBF-funded project WAVES under (01ISF04A) and the Landesstiftung Baden-Würrtemberg foundation. The authors are responsible for the content of this publication.

References

1. Rech, J and Althoff, K.-D.: Artificial Intelligence and Software Engineering: Status and Future Trends. 18(3) (2004), , 5-11.
2. Berners-Lee, T., Hendler J. and Lassila, O.: The Semantic Web. Scientific American 284(5) (2001)
3. Abran, A., and Moore, J.W. (Exec. Eds.), Bourque, P. and Dupuis, R. (Eds.) Guide to the Software Engineering Body of Knowledge (2004)
4. OMG: MDA Guide. http://www.omg.org/docs/omg/03-06-01.pdf (2003)
5. Gomez-Perez, A., Fernández-Lopez, M. and Corcho, O.: Ontological Engineering. Springer (2004)
6. Uschold, M., Gruninger, M.: Ontologies: Principles, Methods, and Applications. Knowledge Engineering Review 11 (1996) 93-155
7. Oberle, D.: Semantic Management of Middleware, Volume I of The Semantic Web and Beyond Springer, New York (2006)
8. Mayank, V., Kositsyna, N., Austin, M.: Requirements Engineering and the Semantic Web, Part II. Representation, Management, and Validation of Requirements and System-Level Architectures. Technical Report. TR 2004-14, University of Maryland (2004)
9. Decker, B., Rech, J., Ras, E., Klein, B., Hoecht, C.: Selforganized Reuse of Software Engineering Knowledge supported by Semantic Wikis. In: Proc. of Workshop on Semantic Web Enabled Software Engineering (SWESE). November (2005)
10. Lin, J., Fox, M. S.; Bilgic, T.: A Requirement Ontology for Engineering Design. Enterprise Integration Laboratory,, University of Toronto, Manuscript, September (1996)
11. Wouters, B., Deridder, D., Van Paesschen, E.: The Use of Ontologies as a Backbone for Use Case Management. In: "European Conference on Object-Oriented Programming (ECOOP 2000), Workshop : Objects and Classifications, a natural convergence" (2000)
12. Cheesman, J. and Daniels, J.: UML Components: A Simple Process for Specifying Component-Based Software. Addison-Wesley, 2000.
13. Mili, A., Milli, R.., Mittermeir, R.T.: A Survey of Software Reuse Libraries. In: Annals of Software Engineering, vol. 5, (1998) 349-414
14. Happel, H.-J., Korthaus, A., Seedorf, S., Tomczyk, P.: KOntoR: An Ontology-enabled Approach to Software Reuse. In: Proc. of the 18th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE), San Francisco, July (2006)
15. Tetlow, P., Pan, J., Oberle, D., Wallace E., Uschold, M., Kendall, E.: Ontology Driven Architectures and Potential Uses of the Semantic Web in Software Engineering. W3C, Semantic Web Best Practices and Deployment Working Group, Draft (2006)

16. Kiko, K. and Atkinson, C.: Integrating Enterprise Information Representation Languages. In: Proc. of Int. Workshop on Vocabularies, Ontologies and Rules for The Enterprise (VORTE 2005), Enschede, The Netherlands (2005)
17. Cranefield, S.: UML and the Semantic Web. In: Proceedings of the International Semantic Web Working Symposium (SWWS), Stanford (2001)
18. Baclawski K., Kokar, M. K., Kogut, P., Hart, L.; Smith J. E., Letkowski, J., and Emery, P.: Extending the Unified Modeling Language for Ontology Development. Int. Journal Software and Systems Modeling (SoSyM) 1(2) (2002) 142-156
19. Gaševic, D., Djuric, D., Devedzic, V., Damjanovic, V.: Approaching OWL and MDA Through Technological Spaces. Workshop WS5 at the 7th International Conference on the UML, Lisbon, Portugal (2004)
20. OMG: Ontology Definition Metamodel RFP. http://www.omg.org/dontology/, 6th Revised Submission (2006)
21. Evans, E.: Domain-Driven Design - Tackling Complexity in the Heart of Software. Addison-Wesley (2004)
22. Knublauch, K.: Ramblings on Agile Methodologies and Ontology-Driven Software Development. In: Proc. of the Workshop SWESE, ISWC, Galway, Ireland (2005)
23. Knublauch, H., Oberle, D., Tetlow, P., Wallace, E.: A Semantic Web Primer for Object-Oriented Software Developers. W3C Working Group Note, http://www.w3.org/TR/sw-oosd-primer/, 9 March (2006)
24. Völkel, M.: RDFReactor - From Ontologies to Programatic Data Access. In: Proc. of the Jena User Conference 2006. HP Bristol, May (2006)
25. Schach, S. R.: Object-Oriented and Classical Software Engineering. 6. McGraw-Hill (2004)
26. Eberhart, A. and Argawal, S.: SmartAPI - Associating Ontologies and APIs for RAD. In: Proceedings of Modellierung (2004)
27. Welty, C.A.: An Integrated Representation for Software Development and Discovery. Ph.D. Thesis, Rensselaer Polytechnic Institute (1995)
28. Devanbu, R,. Brachman, J., Selfridge, P. G., Ballard, B. W.: LaSSIE? - A Knowledge-Based Software Information System, ACM Comm., 34(5), (1991) 34-49.
29. Welty, C.A.: Software Engineering. In: Description Logic Handbook (2003) 373-387
30. von Halle, B.: Business Rules Applied Building Better Systems Using the Business Rules Approach. John Wiley & Sons (2001)
31. McClintock, C. and de Sainte Marie, C.: ILOG's position on Rule Languages for Interoperability. In: Proceedings of the W3C Workshop on Rule Languages for Interoperability. Washington, DC, USA. (2005)
32. Motik, B., Sattler, U. and Studer, R..: Query Answering for OWL-DL with Rules, Proc. of the 3rd Int. Semantic Web Conference (ISWC 2004), Hiroshima, Japan (2004)
33. Huhns, M.H., and Singh, M.P., Service-Oriented Computing: Key Concepts and Principles, IEEE Internet Computing, 9(1), (2005). 75-81
34. McIlraith, S. A., Son, T. C., Zeng, H.: Semantic Web Services. IEEE Intelligent Systems, 16:2, (2001) 46-53
35. OWL Services Coalition: OWL-S Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.0/owl-s.html (2004)
36. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX - A Semantic Service-Oriented Architecture. In: Proc. of the Int. Conference on Web Service. Orlando (2005)
37. Ankolekar, A.: Towards a Semantic Web of Community, Content and Interactions. Ph.D. Thesis September, CMU-HCII-05-103 (2005)
38. Ankolekar, A., Sycara, K., Herbsleb, J., Kraut, R., Welty, C.: Supporting Online Problem-solving Communities with the Semantic Web. In Proc. of the 15th Int. Conference on World Wide Web, Edinburgh, Scotland, (2006) 575-584.
39. Dameron, O.: Keeping Modular and Platform-Independent Software Up-To-Date: Benefits from the Semantic Web. 8th International Protege conference, Madrid, Spain (2005)
40. Booch, G.: Generations of Software Architecture. http://www-03.ibm.com/developerworks/ blogs/page/gradybooch?entry=generations_of_software_architecture. September (2005)