RapidOWL - an Agile Knowledge Engineering Methodology

Sören Auer
University of Leipzig
Institute for Computer Science
Augustusplatz 10-11, 04109 Leipzig, Germany
auer@informatik.uni-leipzig.de

Abstract

Agile methodologies have recently gained growing success in many economic and technical spheres. This is due to the fact that flexibility, in particular fast and efficient reactions to changed prerequisites, is becoming increasingly important in the information society. To support adaptive, semantic collaboration between domain experts and knowledge engineers, a new, agile knowledge engineering methodology, called RapidOWL is proposed. This methodology is based on the idea of iterative refinement, annotation and structuring of a knowledge base. A central paradigm for the RapidOWL methodology is the concentration on smallest possible information chunks. The collaborative aspect comes into play, when those information chunks can be selectively added, removed, annotated with comments or ratings. Design rationales for the RapidOWL methodology are to be light-weight, easy-to-implement, and support of spatially distributed and highly collaborative scenarios.

1 Introduction

RapidOWL is an adaptive, light-weight methodology for collaborative Knowledge Engineering. The major aim of RapidOWL is to make the elicitation, structuring and processing of knowledge and thus the cooperation of domain experts and knowledge engineers more efficient. The RapidOWL methodology is based on the idea of iterative refinement, annotation and structuring of a knowledge base. Central to the paradigm for the RapidOWL methodology is the attention given to the smallest possible information chunks (i.e. RDF statements). The collaborative aspect comes into it's own by allowing those information chunks to be selectively added, removed, or annotated with comments and/or ratings. Design rationales for the RapidOWL methodology are to be light-weight, easy-to-implement, and supportive spatially distributed and highly collaborative scenarios.

RapidOWL is, on the one hand, inspired by the XP.K methodology (eXtreme Programming of Knowledge-based systems, [12]), which extends Extreme Programming to an agile methodology for the development of knowledge-based systems. On the other hand, RapidOWL is influenced by the Wiki idea [13], which established agile practices for collaborative text editing. However, contrary to XP.K the RapidOWL methodology stresses the generic nature of a knowledge base and thus focuses on development of knowledge bases, whose final usage scenario is either not a priori known or a single usage scenario is not easily definable. This is usually the case for conceptualizations targeting at information integration as well as for shared classification systems and vocabularies. Different from the Wiki idea on the other side RapidOWL's artifacts are structured information and knowledge represented in statements rather than the Wiki's unstructured text documents.

In this paper we will first clarify the notion of a knowledge engineering methodology (Section 2), than we present briefly approaches related to RapidOWL (Section 3). We collect some specific knowledge engineering characteristics which are, from our point of view, not adequately honored by existing approaches (Section 4). Thereafter, we present RapidOWL by exhibiting underlying paradigms (Section 5), its single process (Section 6) and finally discuss RapidOWL in the light of criteria for analyzing knowledge engineering methodologies (Section 7).

2 Knowledge Engineering Methodology

Many definitions of methodology can be found in the literature (see [12, Page 9]). In this document we will (consistent with [12]) adopt the view of Alistair Cockburn [6], who defines a methodology as "an agreement of how multiple people will work together. It spells out what roles they play, what decisions they must reach, how and what they will communicate".

Definition 1 *Knowledge Engineering Methodology A knowledge engineering methodology is an agreement*

of how multiple people will work together. It defines a process in which domain experts and knowledge engineers will build a knowledge base. This knowledge base is represented in a knowledge representation language with suitable tools. Processes, languages and tools are based on knowledge representation paradigms.

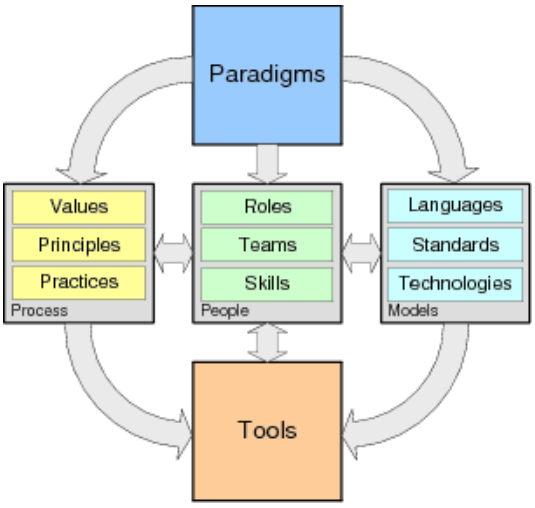

Figure 1. The way of portraying agile methodologies according to Alistair Cockburn.

The RapidOWL methodology is presented in this document following other agile methodologies. Figure 1 summarizes the important ingredients, i.e. people, paradigms, processes, models and tools. RapidOWL is grounded on paradigms (see Section 5). Paradigms influence the process (see Section 6), they lay the foundation for the models, and have to be internalized by people. Last but not least, tools implement the collaboration processes between people on the basis of the methodologies models. This document primarily aims at sketching the agile Knowledge Engineering process of RapidOWL (for a more detailed description see [3]). In addition, a framework supporting RapidOWL tool development exists [2] and it has also successfully applied in practice [4].

3 Existing Approaches

Related approaches can be roughly classified into two groups. Accompanied by the formation of knowledge engineering as an independent field of research several Knowledge Engineering methodologies were developed. Most of them are much inspired by Software Engineering methodologies. In the Software Engineering domain, in the 90's several Agile Software Engineering methodologies emerged. Triggered by the fact that flexibility, in particular fast and efficient reactions on changed prerequisites, becomes increasingly important, agile methodologies recently also appeared in other areas than Software Engineering.

Knowledge Engineering. The main goal of Knowledge Engineering is to structure the development and use of knowledge bases. For that purpose, the most widely known Knowledge Engineering approaches (such as CommonKADS [15]) are based on the ontology paradigm [11]. The development of both ontologies and adequate reasoning algorithms is supported by various methodologies, the phases and models of which resemble traditional Software Engineering approaches. These Knowledge Engineering methodologies now also reveal similar problems to traditional Software Engineering approaches. Significant initial efforts are needed to make the purpose of the final ontology explicit and to deduce an appropriate model. It is often hard to estimate the required level of detail for the knowledge structuring a priori. Changes to the knowledge structuring are difficult and costly. For these reasons, methods from Knowledge Engineering are often too expensive to apply and rarely used in practice (cf. also [12, page 59]). However, for ontology construction the need for methodologies supporting ontology evolution and the basis of interactive collaboration between actors has been tackled by few approaches (e.g. [14]).

Agile Methodologies. Agile methodologies have recently gained growing success in many economic and technical spheres. This is due to the fact that flexibility, in particular fast and efficient reactions to changed prerequisites, is becoming increasingly important in the information society. This development started in Software Engineering after the realization in the mid 1990's that the traditional "heavy" methodologies do not work well in settings where requirements are uncertain and change frequently. Several adaptive or agile Software Engineering methodologies subsequently evolved (e.g. [5, 7, 16]). Agile methodologies are especially suited for small co-located teams and for the development of non life-threatening applications. Since the problem of uncertain, changing requirements is not limited to the Software Engineering domain, the idea of establishing adaptive methodologies, which can react to changing prerequisites, was also adopted by other domains than Software Engineering. These include 'The Wiki Way' [13] for Content Management, Rapid Prototyping [10] for Industrial Engineering. Also, the Lean Management method was used to some extent in the business management domain.

4 Specific Characteristics of Knowledge Engineering

Most of the existing knowledge engineering methodologies adopt techniques and apply process models from software engineering. However, in many scenarios required knowledge engineering tasks reveal specific characteristics, which an knowledge engineering methodology should be aware of. In the following, we describe some specific characteristics of Knowledge Engineering important for RapidOWL.

Knowledge Engineering is not a Business in itself. There is no market for Knowledge Engineering as there is for Software Development. This is not because Knowledge Engineering is less important in the economic sphere, but due to the fact that the flow of knowledge in most cases accompanies the development of products and services, rather than being an economic asset itself. Hence, Knowledge Engineering services are often required when spatially distributed users and communities have to collaborate on a semantic level. For example, this is the case when a common terminology has to be established, dispersed information must be integrated, or when shared classification systems and taxonomies have to be developed. This type of semantic cooperation is for example often required for Virtual Organizations [1], Scientific communities or standardization boards, or Intra-organizational use. Thus, the actors within Knowledge Engineering processes are often not bound together by a legal contract, or the Knowledge Engineering processes are not part of such a contract.

Lack of a Unique Knowledge Serialization. Agile methodologies rely heavily on sophisticated versioning and evolution strategies due to their focus on small incremental changes. However, agile methodologies, as well as their respective versioning and evolutions strategies within software development, do not seem to be reasonably applicable to knowledge engineering. For example, contrary to software development paradigms, most knowledge representation paradigms do not provide unique serializations. In other words, the ordering of statements or axioms in a knowledge base is irrelevant, while the ordering of source-code lines in software is fixed. Consequently, the use of existing software versioning strategies (e.g. delta method) and their respective implementations (e.g. CVS, Subversion) would not be efficiently suitable.

Spatial Separation of Parties. Most agile Software Development methodologies assume a small team of programmers working closely (especially spatially) with domain experts. This is a reasonable assumption for commercial software development, where a client requests software developers to implement a certain functionality. But when the involved parties are spatially separated, the use of a formal, tool-supported Knowledge Engineering becomes particularly important. Furthermore, the knowledge engineering tasks of establishing common classification systems, shared vocabularies and conceptualizations are is especially important in distributed settings. When teams are co-located implicit knowledge representation in the form of text documents in conjunction with verbal communication turns out to be more efficient and for a long time established.

Involvement of a Large Number of Parties. The growing together of the world by Internet and Web technologies enabled completely new mechanisms of collaboration. Open source software projects as for example the Linux kernel or collaborative content authoring projects as Wikipedia demonstrate this power of scalable collaboration impressively. However, knowledge engineering is especially challenging when a large number of domain experts have to be integrated into the knowledge-engineering process. Agile software development methodologies claim to be best suited for small to medium sized development scenarios. This is mainly due to the accent on and need for instant communication. On the other hand, the interlinking of people and tools using internet technologies facilitates scaling of agile cooperation scenarios. Knowledge Engineering scenarios in most cases differ from software development scenarios: it is usually not optional, but crucial to integrate a large number of domain experts, knowledge engineers and finally users of the knowledge bases.

5 Paradigms

The basic paradigms of conventional knowledge engineering methodologies are the generic architecture of knowledge-based systems, ontologies and problem-solving methods. We argue that the paradigms of an agile knowledge engineering methodology, with a focus on semantic-web knowledge representation standards, must both reflect the distributed interlinked nature of the Web and recognize statements as being the smallest building blocks of semantic-web knowledge bases. Hence, the vague knowledge representation paradigm of ontologies is replaced by knowledge representation grounded on the *semantic-web data model*, i.e. RDF statement paradigm and the use of web technologies is established as an additional paradigm.

6 Process

Conventional methodologies distinguish different phases within the life-cycle of either software or knowledge. Agile methodologies give the importance of applying a change a much higher value than being located in a certain stage

of the life cycle. Consequently agile methodologies do not provide a phase model. Instead, they propose values from which (on the basis of paradigms) principles are derived for the engineering process in general, as well as practices for establishing those principles in daily life. We will describe RapidOWL along these dimensions in the following subsections, and then give an overview of how they can be combined and applied in practice.

Values. RapidOWL adopts the values of eXtreme Programming, namely Communication (to enable collaborative ontology development), Feedback (to enable evolution), Simplicity (to increase knowledge base maintainability) and Courage (to be able to escape modeling dead-ends). However, RapidOWL combines the values of Communication and Feedback in it's Community value. This includes the social constructs that underly the communication and subsumes feedback as a special form of communication. In addition to XP's values, RapidOWL includes the value of Transparency. These values are explained below.

Principles. Based on the four values which represent long-term goals, the RapidOWL development process is guided in the mid-term by various principles. They are partly inspired by Ward Cunningham's design goals for the first Wiki system [8]. The principles include an *open-word* assumption, promotion of *incremental changes*, *uniform authoring* methods for both modeling and instance acquisition, *observable development* and *rapid feedback* (see [3] for details). The principles describe the single RapidOWL process axiomatically in the sense that they define characteristics the RapidOWL process should possess. Concrete practices are derived from the principles aiming at achieving these desired characteristics in daily routine without prescribing a rigid process.

Practices. The practices of RapidOWL are inspired by the practices of eXtreme Programming (XP) and from Holger Knublauchs modifications for software / knowledge base co-design XP.K [12]. Due to the specific characteristics of knowledge engineering (cf. Section 3) not all practices from XP have an equivalent in RapidOWL and inversely. They include among others: *Joint Ontology Design* (to ease collaboration between knowledge engineers, domain experts and users), *Information Integration* (to ground the knowledge elicitation on existing information), *View Generation* (to provide domain specific views for human users and software systems) and *Ontology Evolution* (enabling the smooth adoption of modelings and corresponding instance data migration). In contrast to software development, where the team of full-time programmers can be easily instructed to put the values and principles of XP into daily routine, RapidOWL aims to turn domain experts into part-time knowledge engineers by keeping practices as simple as possible and by proposing strategies to support them with tools.

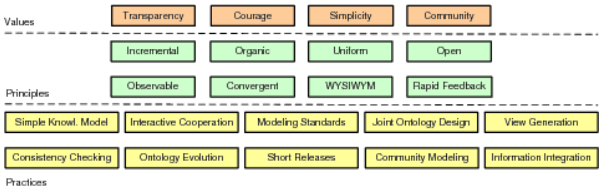

Figure 2. The building blocks of RapidOWL: Values, Principles, Practices.

Putting it all together. The values, principles, and practices outlined in the previous paragraph are the major ingredients of the RapidOWL methodology. They are also the main things that domain experts, knowledge engineers and especially tool developers for RapidOWL enabled knowledge engineering and management systems, should be aware of. In contrast to systematic engineering methodologies, RapidOWL does not prescribe a sequence of modeling activities that should be precisely followed. Furthermore, RapidOWL does not waste resources on comprehensive analysis and design activities. Instead, it follows the philosophy of agile methodologies, in which agility in the face of changing requirements and knowledge models is a major goal. The individual tasks and contributions of prospectively involved parties in building a knowledge base on the basis of the RapidOWL principles are presented in the next paragraphs.

RapidOWL encourages *domain experts* to initially express all facts they assume as true and worth being, represented by means of statements. This can be simply the adding of a statement which attaches an `rdfs:label` or `rdfs:documentation` to a URI reference, to give that URI reference an informal meaning. Or, instance data can be gathered by importing existing documents (spreadsheets, listings, text documents) into the knowledge base. A spreadsheet containing structured information in tabular form, for example, can be interpreted as a class, where columns indicate properties and rows usually represent instances. This activity promotes a shallow learning curve, since domain experts can instantly participate. As soon as an expert starts working on such a knowledge representation, other experts can observe every single step. They can add comments, vote about the usefulness of certain representations, add their own knowledge fragments and/or delete other ones.

More *experienced domain experts* assist in restructuring, interlinking and consolidating the gathered data. Importing

information from legacy documents often results in duplicates, because, for example, columns in different spreadsheet documents representing the same information are not labeled in a uniform manner. Such duplications have to be detected and eliminated (e.g. by merging the respective properties into one). To reduce the costs of such changes, RapidOWL will rely on implemented wizards assisting in the detection of frequent modeling errors and by providing (semi-)automatic resolution to support evolution and migration. It might also be necessary to convert literal data, as, for example, two properties to be merged can represent the same information differently (e.g. names like "Auer, Sören" or "Sören Auer"). This kind of consolidation activity also includes the establishing of relationships which are not yet represented.

Knowledge engineers can support such a community of domain experts with advice for reasonable representation methods and by providing ontology evolution and data migration strategies. Knowledge engineers can further enrich the knowledge base with logical ontology axioms. This includes property characteristics (e.g. transitivity) and restrictions (e.g. cardinality restrictions). Class descriptions are refined with set operators (`owl:intersectionOf`, `owl:unionOf`, `owl:complementOf`). The knowledge engineer can also extract distinct parts or 'slices' of the knowledge base adhering to the OWL species, DL and lite to perform consistency checks by means of Description Logic reasoner. Since RapidOWL does not restrict domain experts in their usage of RDF it is likely that the knowledge base does not fall into the OWL DL or OWL lite categories. The species validation of an OWL reasoner, however, can give hints about how the knowledge base has to be modified. Another task is the testing of existing queries and views on the knowledge base after changes have been incorporated.

RapidOWL focuses primarily on establishing conceptualizations for information integration as well as the establishing of shared classification systems and vocabularies. Hence, tools supporting RapidOWL will have a rather generic than domain specific nature. However, *software developers* participate in the collaboration by developing domain specific applications providing specific views onto the knowledge base, or by assisting domain experts and knowledge engineers in formulation more complex queries to the knowledge base.

RapidOWL does not enforce a distinct succession, neither does it require all the just mentioned tasks and activities to be accomplished by the respective parties. The quality of the knowledge base, however, is determined by carefully performing activities related to consolidation, restructuring and modeling as well as consistency checking. Tools on the other hand, can highly automatize and integrate these activities into the Gathering activity.

7 Conclusion

The purpose of RapidOWL is to bring about a stable state of the knowledge base through small incremental changes from a multiplicity of contributors. To achieve that, RapidOWL applies various techniques and practices with the explicit goal of reducing the cost of change. Much of the assumption that the cost of change can be reduced is based on the value of *Transparency*. The practice *Short Releases* for example promotes that ontologies are published quickly and frequently, so that expensive misunderstandings can be uncovered and eliminated early, when the costs of changing them are still low. *View Generation* furthermore enables domain experts to timely review the representations from different perspectives. *Joint Ontology Design* and *Community Modeling* promote communication between domain experts (and knowledge engineers) and thus help by detecting errors earlier and spreading the knowledge. *Ontology Evolution* enables to undo problematic changes and to estimate prospective effects on instance data. Early *Information Integration* helps that the ontology really captures needed conceptualizations adequately. The *Simple Knowledge Model* of the statement based approach of semantic-web knowledge representation standards is very easy to understand. In conjunction with *Modeling Standards* domain experts are thus enabled to efficiently contribute to a knowledge base, even in absence of knowledge engineers. Finally, *Consistency Checking* helps to build robust and terminologically correct knowledge bases.

Although each of these practices have weaknesses when applied individually, their benefits greatly outweigh their weaknesses when they are used as a combined approach. In other words, the practices of RapidOWL support one and other. This analogous to other agile methodologies (cf. [12, 5]) and due to the 'axiomatic' description of their single process. An example for individual weaknesses and mutual compensation of such is the interplay of the practices of *Short Releases* and *Ontology Evolution*. *Short Releases* of the ontologies may result in instabilities of the resulting knowledge-based systems. However, *Ontology Evolution* supports the early detection of prospective problems and enables the revoke of individual problematic changes in a simple way.

In [9] a number of criteria for analyzing methodologies was proposed. In the following we discuss RapidOWL in the light of these criteria.

Detail of the methodology. RapidOWL is a rather lightweight methodology. This is primarily due to the recognition that knowledge engineering is usually not a business in itself and thus significant resources for evaluating the methodology and later controlling the compliance of the processes with the methodology are not available. Rapi-

dOWL rather banks on tools supporting it than on exhaustive documentation.

Recommendations for knowledge formalization. RapidOWL bases on representation of all knowledge in the form of triples, i.e. RDF statements. A concrete degree of formalization is not prescribed. However, RapidOWL proposes to justify the degree of formalization according to the required querying and reasoning capabilities of the resulting knowledge base.

Strategy for building ontologies. Regarding this criteria it is questioned whether the strategy to develop ontologies is (a) application-dependent, (b) application-semidependent, or (c) application-independent. RapidOWL focuses on the development of rather application-independent ontologies. However, RapidOWL is primarily suited for information integration tasks and tasks related to the establishing of shared classification systems, vocabularies and conceptualizations.

Strategy for identifying concepts. RapidOWL here follows a middle-out strategy, i.e. from the most relevant to the most abstract and most concrete. By stressing the collecting of example or instance data RapidOWL tries to abolish knowledge elicitation by means of face-to-face communication between domain experts and knowledge engineers.

Recommended life cycle. Due to its adaptive nature RapidOWL does not explicitly propose a rigid life cycle. However, many aspects of stages in the life cycle of conventional methodologies can be discovered in RapidOWL's single process.

Differences between the methodology and IEEE 1074-1995. This criteria is related to the conviction that knowledge engineering processes should be similar to conventional software development processes. In this regard RapidOWL is different in two ways: Firstly it stresses the need to react on changed prerequisites, i.e. being agile. Secondly it assumes knowledge engineering to be fundamentally different from software engineering in certain scenarios.

Recommended techniques. RapidOWL stresses the importance of providing concrete techniques for performing the different practices of which the methodology is composed. However, in the description of RapidOWL's practices within this document only starting points on how to put them into effect are mentioned.

Usage and Application. Due to the fact that RapidOWL is rather new and significant resources had not been at our disposal for a broad evaluation the number of successfully realized RapidOWL projects is still small. However, ontologies and applications have been build on the basis of RapidOWL containing approximately 20,000 concepts and serving 3,000 parties (cf. the case study in [**?**]).

References

[1] W. P. Appel and R. Behr. Towards the theory of virtual organizations: A description of their formation and figure. Arbeitspapiere wirtschaftsinformatik, Justus-Liebig-Universitt Gieen Fachbereich Wirtschaftswissenschaften, 12 1996.

[2] S. Auer. Powl: A web based platform for collaborative semantic web development. In S. Auer, C. Bizer, and L. Miller, editors, *Proceedings of the Workshop Scripting for the Semantic Web*, number 135 in CEUR Workshop Proceedings, Heraklion, Greece, 05 2005.

[3] S. Auer. *Towards Agile Knowledge Engineering: Methodology, Concepts and Applications*. PhD thesis, Universität Leipzig, 2006.

[4] S. Auer and B. Pieterse. "Vernetzte Kirche": Building a Semantic Web. In *Proceedings of ISWC Workshop Semantic Web Case Studies and Best Practices for eBusiness (SWCASE05)*, 2005.

[5] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change, Second Edition*. Addison Wesley Professional, 2004.

[6] A. Cockburn. Selecting a project 's methodology. *IEEE Software*, 17(4), 2000.

[7] A. Cockburn. *Crystal Clear*. Addison-Wesley Professional, 2004.

[8] W. Cunningham. Wiki design principles. http://c2.com/cgi/wiki? WikiDesignPrinciples.

[9] M. Fernndez-Lpez. Overview of methodologies for building ontologies. In *IJCAI99 Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends*, 1999.

[10] A. Gebhardt. *Rapid Prototyping*. Hanser Gardner Pubns, 2003.

[11] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, June 1993.

[12] H. Knublauch. *An Agile Development Methodology for Knowledge-Based Systems*. PhD thesis, University of Ulm, 2002.

[13] B. Leuf and W. Cunningham. *The Wiki Way: Collaboration and Sharing on the Internet*. Addison-Wesley Professional, 2001.

[14] H. S. Pinto, S. Staab, and C. Tempich. DILIGENT: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of oNTologies. In *ECAI*, pages 393–397, 2004.

[15] G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. V. de Velde, and B. J. Wielinga. *Knowledge Engineering and Management: The CommonKADS Methodology.* MITpress, 2000.

[16] K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. Prentice Hall, 1st edition, Oct 2001.