Software Engineering and Ontology Engineering: Contributions, Perspectives, Challenges and Lessons Learned

Marcelo José Siqueira Coutinho de Almeida[1,2], Fred Freitas[2] and Alexandre Vasconcelos[2]

[1]Informatics Center, Federal University of Pernambuco – UFPE
Recife, PE, Brazil

[2]Informatics Department, Federal Institute of Technology - IFET
João Pessoa, PB, Brazil

{mjsca@cin.ufpe.br, fred@cin.ufpe.br, almv@cin.ufpe.br}

Abstract. *The growing value associated to semantics in software engineering and the necessity to adopt systematic development approaches in ontology engineering has provoked a convergence between these two areas. In this work we present the main benefits obtained from the software engineering inspirations in order to promote the advancement of ontology engineering. We believe that this will enable the development of more sophisticated forms to create, use and manage ontologies in more complex scenarios.*

Keywords: Ontologies, Software Engineering, Ontology Engineering.

1. Introduction

Unlike some decades ago, nowadays there is almost a consensus about ontologies' importance in different areas of computer science such as Knowledge Engineering, Artificial Intelligence, Software Engineering and Business Management, as they are being employed on different types of applications, like knowledge management, natural language processing, electronic commerce, intelligent integration of information, information retrieval, design and integration of databases, bio-informatics, education, and, mainly, semantic web [Devedzic 2002]. One of the consequences of this fact is that larger and increasingly complex ontologies are being developed regarding the demands of these domains.

Therefore it is important that systematic, efficient and, mainly, more productive forms of development can be established. Developments in this direction must be the result of a discipline that shifts the efforts from an ad-hoc, immature and amateur approaches to formal and systematic approaches, consisting of basic principles that can be shared throughout all developers community and guide them throughout the creation and maintenance of ontologies in an effective and efficient manner.

According to Gómez-Pérez and colleagues [Gomez-Pérez et al 2004], Ontology Engineering (OE) refers to the set of activities regarding to process, life cycle, methods,

methodologies, tools and languages that support the ontology development. For [Devedzic 2002], OE can be described as being a set of activities conduced during the conceptualization, design and implantation of ontologies. To Mizoguchi, OE covers a wide pleiade of disciplines ranging from human sciences like Philosophy and Metaphysics to diverse computer science theoretical topics covering topics such as knowledge representation, , common sense knowledge, knowledge reuse and sharing, development methodologies, as well as other practical ones, like knowledge management, business process modeling, domain knowledge systematization, retrieval, standardization and evaluation of information from Internet [Mizoguchi 1998].

Behind these efforts to conceptualize OE, there is a need to meet the growing demands of business, taking out the excessive conceptualization discussions and looking for principles like productivity, time and quality. Such initiatives are similar to those taken some decades ago when the software industry felt the necessity to use rigid forms of software development, giving birth to the Software Engineering area [Pressman 2006].

Despite the fact that ontologies are a kind of software, like for example a set of conceptual models, it is not suitable to use the methodologies of software engineering to develop them, what demands certain adaptations related to the its special characteristics [López 1998]. Ontologies are domain models, so they should be created bearing in mind that they should portrait the universe of discourse being modeled keeping its engagement with reality. Ontologies don't allow the generations of executable software, what imposes restrictions in its development model. On the hand, Software Engineering techniques proved successful to produce well documented, high quality artifacts, under time constraints. A good productivity has been achieved by this field, a good desiderata for OE.

In this work we present a survey on the current state of existing ontology engineering methodologies and discuss some of their limitations. Also it is our aim to make some comparisons of OE against the software engineering methodologies identifying similarities, differences and experiences from, possibly in order to apply them on ontology engineering.

The remainder sections are organized as follows: in Section 2 we present the motivation to develop this work. In Section 3 we describe the main methodologies of ontology engineering. In Section 4 we make considerations about some Software Engineering influences to OE, more specifically the agile methods, design patterns and modularity issues. After, in Section 5 we present and discuss the identified problems in current methodologies of OE. In Section 6, we approach the similarities, differences, perspectives and challenges of software engineering and ontology engineering. Finally, in Section 7 we conclude the paper and discuss some ongoing and future works.

2 Motivation

Although ontologies are not a new subject in areas like Artificial Intelligence and Knowledge Management, they only spread to computer area mainstream in the last years after with the advent of the semantic web. That fact made ontologies shift from a theoretical of philosophers and AI scientists to be the silver bullet of a new myriad of web applications.

However, ontology construction proved to be a time-consuming and error prone task. More than 15 years ago, Thomas R. Gruber already had advised ontologists to adopt an engineering perspective of ontology development [Gruber 1993]. Among the

reasons presented, knowledge reusability is outlined as one important issue in order to provide a systematic approach of ontology development. With market acceptance and the further rise of the semantic web, it is natural that the quantity of ontologies being developed grows more and more.

According to Gómez-Pérez and colleagues [Gómez-Pérez et al 2004], the development of the whole area depends basically of comprehensive methodologies in order to shift the ontology development approaches from an ad-hoc to a professional way. Issues as standardized phases, role identification, tool adoption, etc. are posed as crucial to the maturing of the area that, on its turn, should produce high quality knowledge-based products. However, Simperl and Tempich [Simperl and Tempich 2006] identified the impressive fact that developers use almost no methodology in their projects. This shows the lack of commitment to produce professional artifacts unlike the ones delivered from software engineering standards.

Despite to the strong interest on ontologies and the recent advancements of area, there is no methodology to accomplish these advancements and to propose modern forms to approach ontology development. In this sense, we can observe that important themes must be discussed in ontology methodology area. In next four paragraphs we outline some themes which deserve special attention.

Firstly, we highlight the importance of **reusability** to improve the whole process. We consider that the way reusability is being treated is not satisfactory to fulfill the current objectives both of research and business. Since the work of Gruber, reusability almost ever is being used in a (full) ontological level, leaving away the (partial) modular level. Some works like [Stuckenschimidt and Klein 2003] proposes and discusses the needs to adopt modules in ontology development and others like [Euzenat et al. 2007] outline the advantages from a software engineering perspective.

A second important theme is **quality**. According to ISO 9000, [Wikipedia 2008] quality refers to the degree to which a set of inherent characteristics fulfills requirements. In this sense, Deming affirms that productivity is augmented by means of quality [Kosciansky and Santos 1999]. Further, quality affects both the process and its final product. If ontologists aim to develop high level quality ontologies it is necessary to create forms to measure them. Roger Pressman considers development process quality as a major issue to assure the quality of product being developed [Pressman 2006]. Given that ontologies have its own peculiarities, specific forms to evaluate and enforce quality in the development processes become crucial.

The third important theme is **market**. First, ontologies are being adopted in software products development in many forms [Happel and Seedorf 2006]. Moreover, the emerging semantic web will be ever demanding an increasing number of ontologies. An example that links these two recent approaches of distributed computing like semantic web services are an example of how ontology based products can contribute to create complex and larger software architectures in order to provide more useful services. In this sense, we argue for the existence of an ontology market similar to the component market of today which products like Enterprise Java Beans (EJB), Distributed Communication (DCOM) etc. This will enable a faster, cheaper and better ontology development process.

The last theme is **standardization**. Nowadays software engineering community has been looking for standardizing their methodologies, approaches, products, etc. through different efforts, today, one of the most important organizations for software advancement and maturing is the Object Management Group (OMG). We believe that a

similar endeavor must be adopted by ontology community. In this way, concepts, models, methodologies and tools to assist ontology construction must be developed under some consensual standards, reducing efforts and conflicts.

In the other hand, software engineering is receiving contributions from the ontologies, improving all the software life cycle.

3. Ontology Engineering Methodologies

In the late 90's, efforts to promote integration and reuse of massive knowledge base written in the formal knowledge and representation of different languages, used in expert systems and intelligent agents, brought back the field of ontologies for the research agenda in artificial intelligence. From this fact, innovative tools and techniques were developed to tackle these tasks, and the field matured over the years [Freitas et al 2004].

One of the subareas of ontologies that evolved was the OE (ontology engineering). In the beginning, the development of ontologies was similar to software engineering in its early days because each team adopted its own set of principles, criteria and phases of the project [Gomez-Perez et al 2004]. The process of building ontologies is leaving out these ad hoc efforts to become a strict discipline of engineering. This change is still ongoing, and the methodologies of OE are an active area of today's research [Freitas et al. 2004].

Making an analogy with software engineering, we can say that the ontology engineering is still moving towards its development, especially regarding the methodological aspects. One of the biggest challenges remains that is finding productive ways to develop ontologies. The reuse of ontologies modules, similar to what is being done for years in object oriented programming, is not yet covered by existing methodologies. This facility would allow developers to make the process fast, efficient and economical.

In next section, we will summarize some of the main methodologies of ontology development in order to better understand the actual ontology engineering scenario.

3.1 Overview of Most Outstanding Methodologies

Along the last decade the ontology community witnessed the rising of many approaches to develop ontologies under a systematic approach. One of the first initiatives came from the experience of Grüninger and Fox on the TOVE (Toronto Virtual Enterprise) project at the University of Toronto, Canada [Gruninger 1995]. This methodology has been used to develop ontologies for corporations. Their inspiration was the development of knowledge-based systems using first order logic. It is composed of a sequence of steps which departs from motivating scenarios' description, very similar to software engineering use cases.

Almost at same time, Uschold and King [Uschold and King 1995] presented a methodology based on their experience in the ontology development for the Office of Enterprise Ontology Applications of Artificial Intelligence at the University of Edinburgh, Scotland, in partnership with several companies, among them IBM, Logica UK Limited, Lloyd's Register and Unilever.

Methontology was developed in the laboratory of Artificial Intelligence of the Polytechnic University of Madrid and its main inspiration came from the main activities identified by the standardized IEEE software development process [IEEE 1996] and the

knowledge engineering methods [Gomez-Perez 1997] [Waterman 1986]. It is certainly the most comprehensive from the existing ones for many reasons: Firstly, because it is based on a very well founded standard of software development. Secondly, because it can be both used to build ontologies "from scratch" as from other existing ones, i.e., it also supports the ontology reengineering.

Diligent is the result of Christoph Tempich doctoral thesis in the University of Karlsruhe [Tempich 2006]. It proposes to support the building and evolution of ontologies in decentralized environments in a peer-to-peer fashion. Remotely located users can benefit of methodology in order to share knowledge in organizational environments and further facilitating its reuse. Similar to Methontology, this approach presents a development process and some tools.

Recently Paul Doran and colleagues [Doran et al. 2007] presented a methodology which main characteristic is the reusing of ontology modules. As it is discussed further in section 4.1.4, this work does not approach composition activities, just leading with decomposition ones.

4. Some SE influences to Ontology Engineering

In this section we will outline three efforts related to ontology engineering which resemble and/or regard strong connection to software engineering techniques: agile development methods, patterns and modularity. Despite these efforts still have a state of art status, they can also be seen as successful examples of transporting techniques and methodologies of the software engineering field to the ontology field. We consider modularity as the most relevant among them due to the prospective impact it cause on the ontology field.

4.1 Agile Methods

In software engineering, agile methods are a response to traditional methodologies in order to turn the software development flexible, simpler, cheaper, less error prone, better understood development method and with quickest deliveries [Beck 00a] [Mountain 2008]. Similarly to SE agile methods, some works are being proposed in order to bring the agile principles to the ontology development. The methodologies portrayed here regard SE agile methods because they do not have any development cycle with well defined phases and milestones among them. Instead values are proposed in order to guide the development process throughout principles and practices. The motivation behind this is the simplicity and flexibility during the development. Naturally, not every SE agile practice must correspond to an ontology one (and vice-versa) because of the specificities of knowledge engineering.

In the next subsections we will describe the ontology agile approaches XP.K, RapidOWL and OntoAgile.

4.1.1. XP.K

XP.K (eXtreme Programming for Knowledge-Base Systems) is an agile development methodology based on XP (eXtreme Programming) [Knublauch 2002]. It is argued by its creator that heavy-weight methodologies are not suitable for KBS because they introduce high-costs during the life cycle, mainly when there is requirement changing.

The main ideas advocated by the author relate to need for collaboration and evolution. In first case, it is needed that the people involved during the ontology development, each one performing specific roles, for example domain experts, knowledge engineers and system developers can communicate and contribute for themselves. In former case, it is argued that requirements are hard to find and constantly changes.

Among the XP values XP.K considers principally the following values:

- **Simplicity**: It is suggested to start the process as simple as possible and evolve when new knowledge is obtained. Also, simplicity must guide the choice of language for the meta-models. For example, object-oriented languages are simpler to adopt for non-experts than formal languages.
- **Communication**: Since ontologies must be developed collaboratively, it is demanded that the evolved ones have a good communication. The *pair modeling* practice is intended to improve the model quality as in the same way that XP pair programming does in software development.
- **Refactoring and Design Patterns**: These two practices are applied in the same fashion of XP. For example, refactoring can be used to combine duplicate elements and design patterns used to apply succeed and well known modeling practices like for example to discover and organize hierarchical relationships.

4.1.2 RapidOWL

RapidOWL is a lightweight methodology for collaborative Knowledge Engineering. It is based mainly on two precedents approaches: XP.K and the Wiki idea, which established collaborative ideas to text editing [Leuf and Cunningham 2001].

RapidOWL methodology is grounded over a well defined structure already adopted by other agile methodologies. This structure is composed paradigms, process, people, models and tools. Each one of these elements describes a particular aspect of development. As can be seen in figure 1, paradigms influence the process. They work as the foundation for the models and they have to be internalized by people. Tools also are an important part of the approach since they implement the collaboration processes among the people involved.

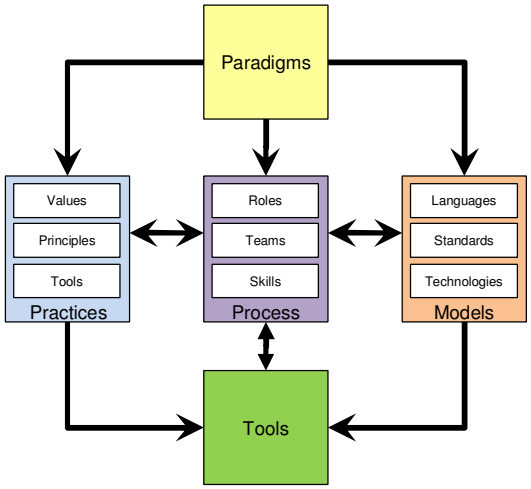

Figure 1: Agile development model.

Following one of principles of the agile philosophy in which developers must deliver functionalities as fast as possible, according to the authors, one of the major advantages of this methodology is the fast delivery of RDF chunks (Resource Description Framework) [Manola and Miller 2004] statements. These chunks are produced by developers from the User Stories. In each iteration, an user story is created describing a given aspect of the domain being studied. Next, developers analyze each story formalize it using the RDF language.

RapidOWL values is based in eXtreme Programming (XP), namely Communication, Feedback, Simplicity and Courage [Beck 2000]. As said above, values are proposed in order to guide the Process throughout Principles and Practices as can be seen in figure 1. RDF is a data model for objects ("resources") and relations between them, provides a simple semantics for this data model, and these data models can be represented in XML syntax [Bray et al 2006]. However, this is not enough to describe semantically the objects in ontologies. In order to support this, OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing and characteristics of properties (e.g. symmetry), and enumerated classes. Therefore we consider that it is not a good design decision to adopt RDF as being the knowledge representation language. In figure 2 it is depicted the values of RapidOWL.

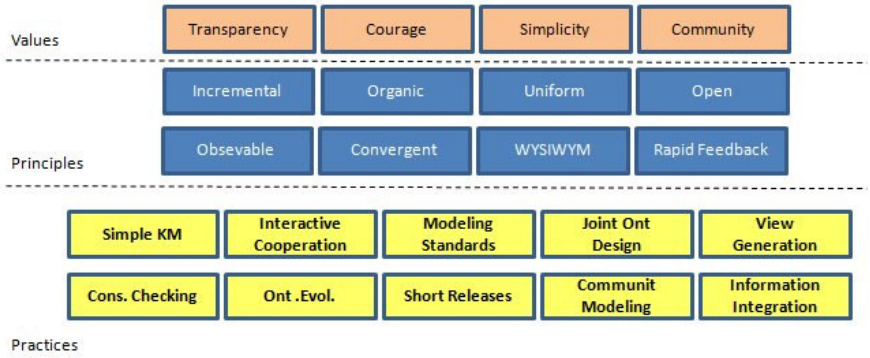

Figure 2: Values of RapidWOL.

4.1.3 OntoAgile

This methodology is influenced by two major agile methodologies: XP [Beck 2000] and Scrum [Mountain 2008]. The main idea of OntoAgile is to use only the necessary documentation for the development, concentrating the efforts on the client throughout short interactions in order to deliver products as fast as possible. Basically the OntoAgile process is composed of three interactive and incremental phases: (i) Planning Meeting, (ii) Construction, and (iii) Delivery Meeting. At each iteraction one small part of the ontology is produced.

In **Planning Meeting**, participants (customer, leader and others members of development team) develop user stories in order to identify what knowledge must be specified. The user stories will be form the Product Backlog, a repository of functional requirements, or in others words "what will be built" [Leuf and Cunningham 2001]. These stories will be organized in priority order and they will be the starting point of process.

Construction is divided in six sub-phases:

1. Concepts Acquisition: identify the main concepts;
2. Existing Ontologies: look for the concepts in existing ontologies in order to discover their contexts;
3. Relationship: connects the concepts discovered in previous subphases;
4. Codification: implements the ontology using a tool like Protégé or directly with a language like OWL;
5. Integration: puts together results from each iteration (called Sprint) with the ontology;
6. Consistency Checking: verifies concepts, relations and restrictions by means of a reasoner to check or SPARQL.

In the **Delivery Meeting** phase, stories are used to deliver that part of ontology being produced in the current iteration. The same people that were present in the first phase must attend its validation.

4.1.4. Discussion on Agile Methods for OE

Despite of its promises, we identified some problems with agile methodologies for ontologies. Firstly, any agile methodology scope is very limited basically working just with small-size ontologies and does not consider important activities of ontology engineering like reengineering and modular development. The first one is a necessary requirement when developers have an ontology and need to enhance or adapt it to fulfill a new demand. The former approach constitutes a trend in ontology engineering once ontology engineering inspires on software engineering.

Another strong point in every agile methodology is communication. However, the informality or the lack of standards (particularly in RapidOWL) during the activities can harm the communication among the participants and then harm the whole development process. Regarding to Knowledge Engineering, agile methods for ontologies focuses on the implementation level [Newell 1982], what limits the portability of the ontology being produced. Also this fact prevents both the usage of ontological modules from existing ontologies and the full reuse of top ontologies.

One important question related to these approaches is the lack of documentation and more details about their utilization, what prevents developers to adopt them.

4.2. Patterns in Ontology Development

Patterns are a successful strategy in software engineering to promote the reuse of both knowledge and expertise about the software development [Gamma et al. 1994]. Currently there are patterns for all life cycle activities, from analysis to test. Probably the major success regarding to patterns usage in software engineering is that one obtained on the Gang of Four" (GoF) catalogue [Gamma et al. 1994]. This catalogue was the very first initiative in computing to reuse experiences in a patterned way. It contains 23 patterns each one depicted in a well organized template in order to facilitate learning and understanding. The main advantages are time and effort reduction and the increased quality [Blomqvist 2004].

Although patterns were not adopted largely by ontology community yet, some recent initiatives have been put forward inspired by this experience. The reasons to invest efforts in promoting pattern-based development are straightforward: firstly, manual ontology development is a tedious and complex task, so automatic or semi-automatic ways to develop ontologies are highly welcome in such scenario [Blomqvist 2004]. Thus, reusable patterns simplify the work of domain experts and knowledge engineers. Moreover, patterns aid to integrate different ontologies [Gangemi 2005].

One interesting work was accomplished by the Semantic Web Best Practices and Deployment Working Group (SWBP) of W3C, including a task force on Ontology Engineering Patterns [Noy et al. 1995]. These efforts produced some design patterns in language level (for example, Web Ontology Language - OWL) regarding to whole-part, time and semantic integration issues.

The work of [Blomqvist 2004] proposes a classification scheme for ontology patterns. The scheme divides ontology patterns in five levels:

- **Application Patterns**: aim to describe generic ways to use the implemented ontologies in terms of purpose, context, interfaces etc. this idea of abstracting the best ways to apply and use an ontology, or several ontologies, within some context or application is an important issue.
- **Architecture Patterns**: aim to describe a generic way to design the overall structure of an ontology, in order to fulfill the goal of the ontology in question. Important issues here are whether to divide the ontology into components or modules, or to divide into layers or use other construction principles.
- **Design Patterns**: aims to describe a generic recurring construct in ontologies. Similar to Software Engineering Design Patterns (SEDP) [Gamma], the main idea is to be specific enough, allowing them to be used automatically or semi-automatically, and at the same time generic enough to be useful in several ontologies of a certain domain [Gangemi 2005].
- **Semantic Patterns**: aims to describe a certain concept, relation or axiom in a language independent way.
- **Syntactic Patterns**: aim to describe representation symbols in a language dependent way in order to create a certain concept, relation or axiom [Noy et al 1995].

Specifically to Design Pattern Level, [Gangemi 2005] has proposed conceptual patterns based in experiences in ontology engineering experiences in Laboratory for Applied Ontology (LOA). These patterns emerged out from different domains, different tasks and from experts from different areas. This approach is based on a set of typical competence questions like these following:

- Who does what, when and where?
- Which objects take part in a certain event?
- What are the parts?
- What is an object made of?
- What is the place of something?
- What is the time frame?
- Which technique, method or practice is being used?
- Which tasks should be executed in order to achieve a certain goal?
- Does this behavior conform to a certain rule?
- What is the function of that artifact?
- How is that object built?

Each question is called *Generic Use Case* (GUC). It provides the generic vision of a recurrent question in various different domain problems. Each GUC (or set of GUCs) must be encoded throughout a formal pattern called Conceptual Ontology Design Pattern (CODeP). A CODeP is a template to represent, and possibly solve, a modeling problem and describes a "best practice" of modeling [Gangemi 2005]. Some examples of CODePs are: participation pattern, role-task, information-realization, description-situation, design-object, attribute parametrization etc.

A very simple example is the Participation Pattern. As depicted in figure 3 it assumes a relation participant-in of various objects in a given event during a time interval. Some objects participate always while others participate during a certain period of time. Also, Objects are located in a certain Space Region.

In the next section, in order to focus on flexibility and productivity, we will concentrate on modular ontologies.

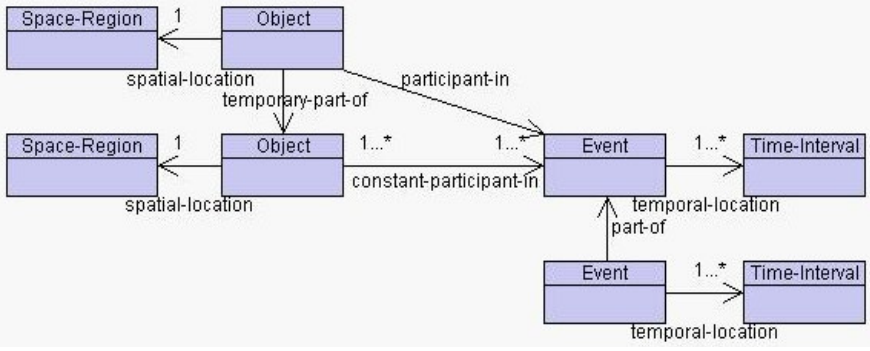

Figure 3: The design pattern Participation at spatio-temporal location [Gangemi 2005].

4.3. Modular Ontologies

Despite the fact that one of the major goals of ontologies is the reuse of knowledge, the resources currently available in existing methodologies have little or nothing to offer in that sense, almost always confined to the extension or import of whole ontologies. Thus, despite the fact that definitions like that presented in [Gruber 1993] suggest that the creation of ontologies in itself encourage the reuse of knowledge, this fact is not evidenced in practice. Thus, modularization is not taken into account, despite having significantly facilitated the collaborative development and reuse in SE along the last decades. Therefore they could, a priori, also be applied successfully in OE.

Also, it important to detach the long term existing tradition existing in information technology in applying the "divide and conquer" principle in order to reduce complexity, reduce costs and efforts etc.

The next sections will discuss some issues related to modularity of ontologies. This discussion aims to explain the modularization concept in the ontology context and to highlight the difficulties in the approach based on modules and therefore draw attention to the importance of a methodology based on modules that guide developers in conducting their work.

4.3.1. Module Definition and Description

Modularization is a technique already widely used successfully in SE. Because of the maturity of the oriented-object paradigm as a development approach in the last decade, the concept of module has been widely used as the basic unit of design during the construction of models in analysis phase, and as abstract data type during the implementation phase. More recently the concept of component has attracted the attention of the development community as the great promise for the development of software in the same way that traditional engineering (automotive industry, electric-electronics industry etc.) create their artifacts.

In order to turn the development of modular ontologies a mature discipline, it is needed that there exists a consensus in the community of developers about what is a module in this specific context. An intuitive understanding of the concept of module is that it is a part of a whole that makes any sense. Moreover, it can be separated from the whole, without having necessarily to keep the same functionality [Spaccapietra et al. 2005]. Therefore, part of an ontology should continue being an ontology, i.e., it should consist of a coherent set of classes, instances, relationships and axioms. There is must be also a form to reuse these elements into a different environment, probably different from that one for which it was projected.

A module should describe only a portion of a field so that it can be reused in other areas. For example, a module describing a human body organ could be used in a congenital diseases ontology and in an ontology about animals in general.

A very interesting definition is given by Doran: "An ontology module is a reusable component of a larger or more complex ontology, which it is self-contained, but at the same time, supports a clear relationship to other modules of the ontology" [Doran 2006]. As a result, a module can be used as designed or can be extended with new concepts and relationships.

A module is self-contained if all the concepts in the module are based in other concepts contained in the same module, rather than refer any concept outside. According [Grau 2007] "a module is the minimum subset of the ontology axioms that

captures its meaning precise enough and therefore the minimum set of axioms that are required to understand, process, evolve and reuse the entity". To enable the module reuse, there must be a way to describe its content and probably how it should connect to other modules or ontologies. Nevertheless this definition still needs to be better understood because a module is a "sense" (or not) is very subjective, leaving room for different interpretations. For example, in an application, modules must be able to respond to any query sent to the ontology as a whole.

4.3.2 Composition and Decomposition

Module is the basis for two major operations for the emergence of a modular ontology engineering methodology: composition and decomposition.

In composition, as shown in Figure 4, modules are developed independently and are linked together to form a new ontology and thus provide a new feature. This modular architecture should describe any facility to import the modules.

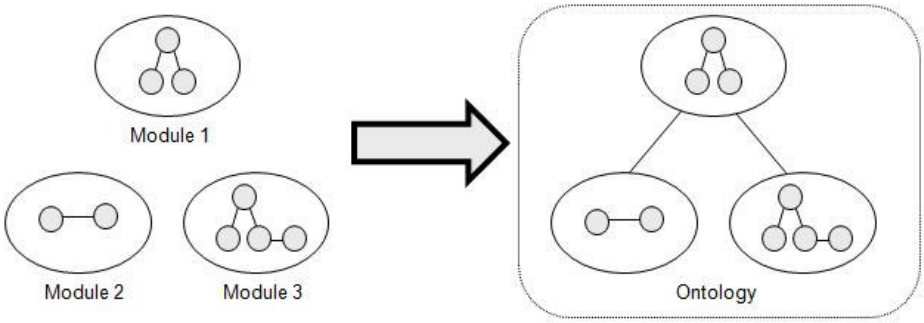

Figure 4: Composition of modules to form a larger ontology.

In decomposition, as shown in Figure 5, modules are built from the division of an existing ontology. Thus, according to criteria established by the designer of the new ontology, it is broken into parts that hopefully would constitute a coherent set of definitions. The problem with this approach is to define what will be the division criteria, because according to this is that it is possible to identify which classes, relationships, instances and axioms are held in each module. This division can be done manually (by a specialist), or (semi-)automatically (by ontology management systems).

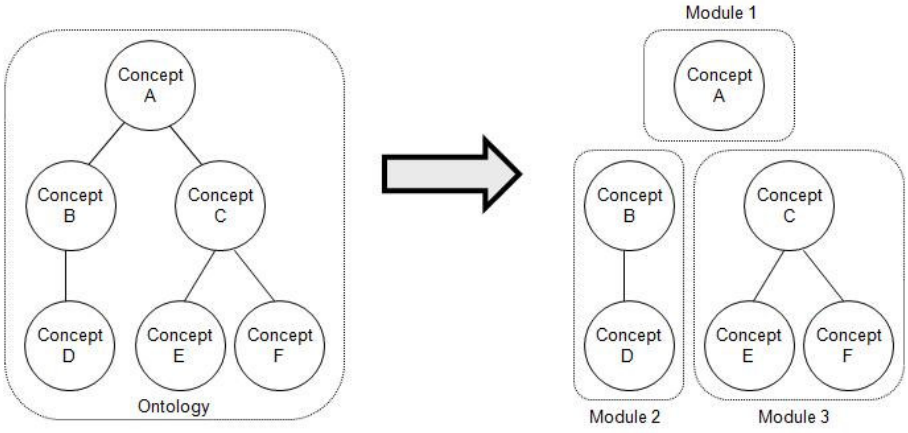

Figure 5: Decomposition of modules.

The understanding of what modularization means exactly, and what are the advantages and disadvantages, depends on the targeted objectives how it will be explained better in following subsection.

4.3.3 Objectives of Modularization

According to [Spaccapietra et al. 2005], the main objectives in the adoption of modules for construction of ontologies can be described as follows:

1. **Scalability**: This goal relates to the issue of performance during the processing of an ontology. It is known that reasoners have their performance deteriorated as the size and complexity of the ontology is increased. Thus, if the processing occurs only in a part of ontology (module), the resulting performance tends to be much more acceptable. The challenge in this case lies in the fact that we need to properly define the size of each module that can answer the query.

This goal is related to the process of composition and decomposition. In first case, it turns now possible to leave the modules separately and further integrate them to form an wider ontology. This would involve the adoption of mechanisms for distributed reasoning, since the modules could be possibly distributed in a networked environment. In former case, the concept of module appears as a natural approach, depending on the need to find more efficient ways of information retrieval. Furthermore, the issue of scalability can be divided into two topics:

- **Scalability for information retrieval**: in this case, modules serve as a prerequisite to define the areas where to make the searches to a given query, restricting the processor to focus only on certain classes, relationships, axioms and instances;
- **Scalability for development and maintenance**: in this case the modules help to understand the impact of an upgrade in the ontology. Thus, to maintain updates within a module would be a efficient form to avoid updating to propagate throughout different parts of ontology.

2. **Complexity Management**: This goal is very similar to the previous one, because there is a tendency to create an inverse relation between the project's ontology and its complexity. Therefore, it is expected that smaller and more specific modules can properly helps to manage them, and consequently compose them to form well designed ontologies.

3. **Understandability**: The understanding of an ontology depends much of its size and complexity. Thus, the modularization will provide a more rational effort during the learning, such as a human agent as an intelligent mobile agent in semantic web structure.

4. **Personalization**: The development of ontologies may be the result of efforts made by several people, possibly located in different places, or the availability of modules localized in repositories distributed over diverse information networks, such as the semantic web. This scenario shows the need to describe the individual responsibility for creating the various modules.

5. **Reuse**: This goal is related to the need for approaches that promote the rapid development of new ontologies from existing modules and it has a direct implication in a large adoption of this technology.

4.3.4 Research Challenges in Modular Ontologies

In this subsection we present a few issues that are seen as fundamental to the creation of a modular. Clarifying these issues will be of great importance for the development of methodologies, tools and languages based on the concept of module.

Criteria for Modularity: It is important that the definition of modules follow certain criteria. However, the way it should be done is an issue that remains open and needs further studies so that they can be clearly defined. The criteria may vary according to way that the creation, use or maintainability is made. If the composition forms a highest-level ontology, there is a concern about what should be contained in each module, because they already exist. Moreover, in the case of decomposition, the ontology already exists and the modules will be created in accordance with the established criteria. In this case, to set a criterion remains as a challenge for researchers. Two ways can be considered:

- **Manual decomposition**: Simplest solution, but the quality of the result depends entirely on the knowledge and expertise of designers of ontology. It is recommended the development of a mechanism for monitoring the reliability and trust services offered by the modules, which could allow a reorganization (or resizing) of the modules when necessary;
- **Automatic or semi-automatic decomposition**: This approach requires knowledge of the requirements of the application. This knowledge can be obtained by analyzing the queries that are sent to the ontology and the storage of ways (paths) that they will go in ontology during the implementation of these. The decomposition of an ontology can also be based on performance criteria. In that case the criteria for the application, such as semantic aspects, are not necessarily considered.

Properties of Modules and Modularization: A very important property in the context of modular ontologies concerns to correctness. The problem can be seen in two ways: (i) *correctness of the modules individually* and (ii) *correctness of the ontology formed by all the integrated modules*. Regarding correctness of modules, one should answer questions like: “what ensures that a module is an ontology?” and “what ensures a module makes sense semantically?”. Still regarding to correctness it should be considered the issue of the composition and decomposition:

- **Composition:** The correctness means the composition of the modules results in something semantically correct. Two issues must be considered: (i) if the composition goal is to produce a fully integrated ontology, verifying if no information was lost after the composition of the modules, and (ii) if the composition goal is the specification of connections (links) between the modules, correctness will be defined as being the guarantee that all the relevant routes were discovered or can be inferred from others, and no connection is duplicated;
- **Decomposition:** Such as in composition, it must be observed if no information was lost during the process. Also it should be verified if the result of a query in an individual module must be the same when it is made in the original ontology. In addition it must be verified if the result of the composition of the modules obtained through the decomposition leads to the same original ontology.

References between composition and modules: Even if a module is considered an independent sub-ontology, it is unlikely that it is capable of fully responding to all queries. In this case it is necessary that there exists a management service that identifies the modules required to meet a particular query, integrate and synchronize the individual responses in order to compose a global response.

Conflicts and Overlapping Modules: During an ontology composition process various types of heterogeneities can arise caused by the different perceptions and decision makings that developers may have about the world. These heterogeneities range from the formalisms used in the construction of the module (RDF, OWL etc.) to its semantic content (for example, a module could describe a mesh of roads and another, a public transport system). Despite the fact that modules can comes from various sources (authors, locations, areas etc.), it is natural to believe that many modules may overlap concepts and definitions (Figure 6).

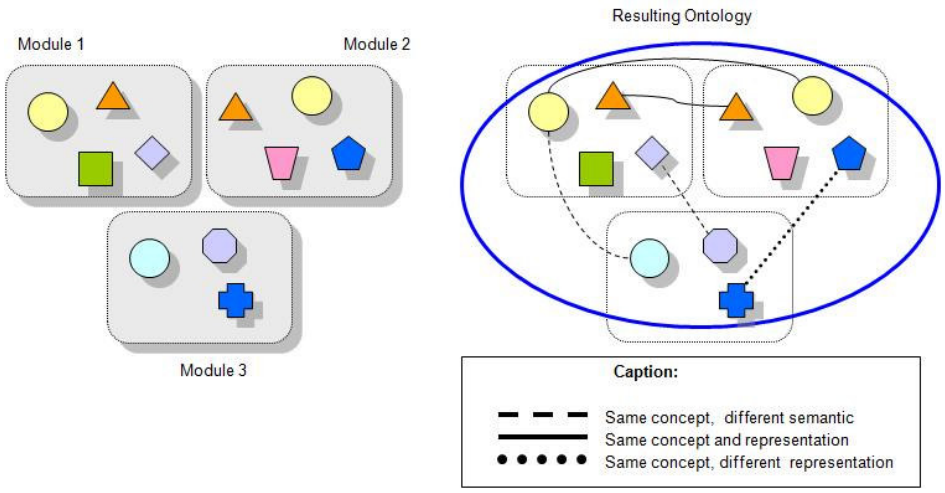

Figure 6: Conflicts in a modular ontology.

Several types of conflicts can arise from the union of several modules. For example, we could have the concepts *developer, author and creator*, each one in a different module, representing someone who produces a particular product. Other types of conflicts come from the fact that different modules can be concepts with different structures:

- **Structural conflicts**: occur when modules have the same external meanings, but with different internal structure. For example, a module could have the concept *restaurant* with an attribute named *evaluation*, which values would be "bad", "reasonable", "good" and "excellent" to represent its quality. Another module would also have a concept called *restaurant*, and this would have an attribute *indication* which also indicates its quality, but with the values "yes", "no" and "with restrictions".
- **Semantic conflicts**: occur when we have different classification schemes, based on understandings and / or different goals. For example, a module would have the concept *Class* with a connection to the concept *student*, where it would have the attribute *ratings*. Another module could have the *Class* concept containing the attribute *ratings*.

Many studies have been developed in order to enable the accommodation of concepts from different sources into a single ontology [Euzenat 2007] [Bezerra et al

2008] [Euzenat and Shvaiko 2007] [Euzenat et al. 2007]. The process known as *ontology alignment* aims to mitigate the various conflicts arising in the ontology merging. The idea is that differences (structural or semantics) are normal in the process of building ontologies. Therefore, a modular ontology construction methodology should provide mechanisms for implementation and documentation having alignment as being an essential activity.

5. Identified Problems in Current Methodologies

Despite of all existing methodologies, the ontology development activities still suffers of various problems and restrictions, what prevents the area to get a better status and finally reach popularity in its community. In the next subsections we will summarize some problems that we consider as being particularly important:

5.1. Lack of Mechanisms to Handle and Reuse Modules

Although of already existing research efforts to introduce modularization on ontology development process, neither one of the existing methodologies accomplish largely the module based approaches. We believe that this new promising approach will succeed just if it makes part of an ontology construction methodology. In other words, it won't be possible successfully to adopt a sophisticated concept like module without the use of a systematic set of guidelines as occurs in Software Engineering Component Based Development (SECBD) [Crnkovic 2007].

5.2. Lack of Mechanisms to Develop Distributed Ontologies

Distributed software development teams are very common in SE development in nowadays. It is very probable that the same approach can be used in OE, creating a distributed authoring environment which people localized in different regions worldwide can share their efforts to create new ontologies. In the same way, the networked communication infrastructures like the Internet can turn possible the existence of modules distributed over this infrastructure similar to what happens in database distributed environments. One foreseen benefit is the possibility to compose and decompose pieces of ontologies from different sources in order to obtain larger ontologies.

5.3. Lack of Mechanisms to Evaluate Inconsistencies among Modules

We believe that the inclusion of modules won't be an easy task, what will demand additional work. Probably modules coming from different ontologies will conflict one with others and questions like these will rise: How they conflict? How much do they conflict? Where is the conflict? Is the conflict acceptable?

5.4. Lack of Mechanisms to Trace Competence Questions

In the well known software engineering modeling language UML (Unified Modeling Language) [Larman 2004], it is possible to trace a single use-case through all the development phases, from the requirements to the source code. Further, one important issue when we are developing a module-based ontology is: which modules are related to what specific competence question(s)? Dynamic use of ontologies can include or remove modules along the time. Furthermore, some modules may not make sense and thus they must be removed from the ontology.

5.5. Lack of a Graphical Language to Better Communicate with the User

It is very important that the representation of ontologies be made by means of graphical notations, like UML. The model created with this language could generate an ontology. However a graphical language is not sufficient to introduce all the necessary information to describe a specific domain. The language could include constraints of diverse kinds, similar to OCL (Object Constraint Language) [OMG 2008]. The main vantage of this language is the simplification of communication between client/domain specialist and ontology engineers. Without knowing the details of ontology development and representation, users will can to analyze and confirm if what is on the picture is right or wrong.

5.6. Lack of a Role-Based Development Model

Ontology is a complex artifact. Furthermore its development process must be conducted by teams composed of experts. Some of them are specialized in analysis, some in tools, and others in testing the ontologies. A methodology must encompass the possibility to gather various persons and each one must concentrate in a certain set of particular tasks. In modern methodologies of software engineering like XP (eXtreme Programming) [Beck 2000] specific roles are defined in each team.

5.7. Lack of Standardization

To be universally accepted and adopted, it is important that a methodology is standardized like RUP, XP, Scrum etc., after to be used by a substantial quantity of users and gain maturity. Despite of standardization does not assure popularity, it is an important issue in order to gain credibility among the users community. The methodology should be able to deal with the requirements above, presents good technical qualities, and should be clear and easy to use.

5.8. Taxonomy of Problems in Ontology Engineering Methodologies

According to what was discussed in last subsections, it was organized the taxonomy depicted in figure 7 to better understanding the problem organization and further proposal of solutions.

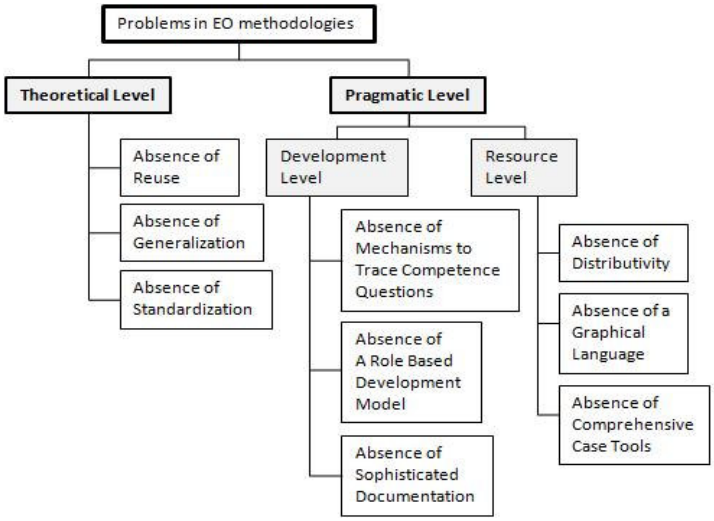

Figure 7: Taxonomy of problems in current OE methodologies.

This taxonomy is initially divided in two parts: *Theoretical* and *Pragmatical*. The first encompass the absence of reuse (monolithic approaches), absence of generalization and absence of standardization. The *pragmatical* level contains the following: *Development* and *Resource Levels*. *Development* level consists of the absence of mechanisms to trace competence questions, absence of a role-based development model and absence of documentation forms. *Resource* level consists of absence of distributivity, absence of a graphical language and absence of a suitable tool support.

6. Ontology Engineering and Software Engineering

Up to that point we have described some ontology engineering subjects that are in state of art yet. We believe that the continuous development of these subjects has a great importance to the advancement of area. In next subsections we will compare each aspect regarded to that in order to provoke discussions and enlighten the next steps of ontology engineering.

Despite the broadness of these subjects, because of space limitations, here we concentrate our efforts on those points that we consider more important to propose a working agenda that can leverage the ontological engineering practices in the next years.

6.1. Similarities and Differences

6.1.1. Preliminary Analysis

Since OE and SE are dedicated to the development of the same subject (software) it is straightforward identifying some differences and similarities.

The first difference concerns to the result being produced by each one. In one hand side, OE and SE deals with the development of systematic, well organized and tooled approaches to develop software. However, the nature of the software being produced is quite different and further the approaches must be slightly different. Despite ontologies are software like any other in their essence, ontologies are not an application. Also, in same way, ontologies are not simply data like strings, structures or literals representing attributes or even data schemas in a data base management system. Ontologies are situated in middle of both of them (application and data).

One important difference between OE and SE regards their origins. While SE was developed having as inspiration the traditional engineering, OE evoluted having as inspiration the Knowledge Engineering [Fernández-López 1998].

6.1.2. Theoretical Analysis

SE has served as an inspiration for the development of OE. In this sense, we believe that it is important to compare the current development state of OE with the current state of SE in order to take advantage of the lessons learned. In next paragraphs, we present an analytical comparison considering those aspects approached just before. Others like Standardization and Documentation will be discussed in future works. These comparisons summarize many topics discussed in early sections, like for example modularization, documentation, standardization, tools and patterns.

Modularization has been discussed and studied for many years in SE and today it can be considered as a mature approach in order to accelerate the process in SE. Although some initial efforts have been accomplished [Bao and Honavar 2006]

[Euzenat et al. 2007] [Aquin et al. 2006] [Doran 2006] [Grau 2007], no standard modularization approach has emerged, what hinders ontology modularity to be adopted largely in short term. Current methodologies and tools don't presents a modularization approach, maybe because to the fact that there exists no consensus about what exactly is an ontology module.

Documentation is one of the main deliverables of any software engineering development methodology. It guides the whole development, throughout the software life cycle, and aids to record the requirements, decision makings, problems found, limitations, roles etc. It plays a major role in preserving projects and even organization's history. Traditional SE methodologies like RUP [Kruchten 2003] and Larman [Larman 2004] make intensive use of documentation. Nevertheless, the large majority of OE methodologies doesn't document the building process. For example, the methodologies of Gruninger and Fox and TOVE don't provide forms to document the development process, except the little remarks contained in the own ontology. Methontology describes one specific activity aimed to document the process but neither it does not accomplish the whole life cycle and nor it defines systematic ways to do it (for example, which are the documents? what are their formats? when to fill up each one of these documents?).

Standardization regards acceptance and consolidation of technologies, tools, approaches, languages, etc. by a considerable amount of developers or organizations. In SE exists many standardization efforts like the Object Management Group[1] (OMG), the Java Community Process[2] (JCP), and the SWEBOK[3] (Software Engineering Body of Knowledge). All these initiatives aims to promote the development, divulgement and maturing of their respective subjects, as well as to integrate the efforts under the same philosophy.

Under an economical perspective, **quality assurance** is one of the most important activities in SE because it promotes the necessary ways to optimize the development processes and thus to improve the quality of products being developed. If in one hand, SE has very matured standards and organizations aimed to promote the quality assurance like CMMI and ISO/IEC 15504 [Kosciansky and Santos 2007], on other hand OE has nothing similar until this moment.

The large adoption of **tools** is a clear indication that an area is reaching its maturity. Since its beginning, SE community has paid attention to this aspect and in late 80's the "Computer Aided" tools generation experienced an exponential growth with the Computer Aided Software Engineering (CASE), Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) tools. Nowadays there is a vast quantity of tools to aid the developers in the whole software life cycle, like requirement editors (planning phase), domain modeling (analysis phase), coding (implementation phase), test and refactoring (test and maintenance phase) etc. In the ontology area, despite the existence of a considerable quantity of tools like Protégé (ontology modeler), Racer and Pellet (OWL reasoner), SPARQL (query language), and others, all of them with very good quality and acceptance by the ontology community, there are not environments dedicated to the whole ontology life cycle.

---

[1] http://www.omg.org

[2] http://www.jcp.org

[3] http://www.swebok.org

Patterns are being adopted largely in SE, particularly the Design and Architectural Patterns. In OE, patterns are still in their very early steps. Some proposals were made like those in [Blomqvist 2004] and [Gangemi 2005], but they still demand more efforts to reach consolidation. We believe that one task to be done is to turn available some automatic support.

The following table summarizes these topics:

Table 1: summary of comparisons among SE and OE.

Topic	SE	OE
Modularization	Mature	Early steps
Documentation	Mature	Little
Standardization Efforts	Good	Inexistent
Tools	Big diversity	Good evolution
Patterns	Mature	Early steps
Quality Assessment	Process and Product	Product (Early steps)
Methodology	Good acceptance	Little acceptance
Heavy Weight Methodologies	RUP, Clean Room, OpenUP	Methontology, TOVE, Cyc
Light Weight Methodologies	XP, Scrum, Crystal	XP.K, OntoAgile, RapidOWL

6.2. Mutual Contributions

OE and SE communities share a number of common topics. While SE has strived efforts towards a higher level of abstraction, devoting increasing attention in modeling activities in software development, OE has dedicated efforts to develop systematic and well organized processes in order to discipline the area. As it was possible to observe in many works in literature, these two areas can benefit mutually in many ways, exchanging experiences and techniques.

Happel and Seerdorf [Happel and Seerdof 2006] enumerate the problems existing in all phases of software life cycle and describes how the application of ontologies can solve each one. According to them, ontologies can improve, among others, (i) the communication abilities in requirement specification because of knowledge representation formalisms, (ii) the possibility to create richer models in analysis and design phases because of higher level abstractions, and (iii) the annotation of API elements with unambiguous concepts.

Ruiz and Hilera [Ruiz and Hilera 2006] surveys the using of ontologies in SE and proposes a very useful taxonomy to assist researches to identify, understand and choose ontologies in software development projects, to identify what proposals of new methodologies or previous adaptations exist for the construction of ontology-driven software, and what types of software artifacts can be formed by or include ontologies.

Other workings presents ontologies developed aiming to software maintenance [Anquetil and Oliveira 2006], software measurement [Bertoa 2006], development environments [Oliveira 2006] and quality assurance [Abran 2006] [Bertoa 2006] [Liao 2005] [Soydan 2006].

In previous sections of this work, we have examined distinct types of inspirations from SE to OE. Approaches based on SE concepts like patterns, components, and agile methodologies can contribute positively to OE evolution in a near future. We believe that other approaches like RAD (Rapid Application Development), distributed development, quality models etc. also can leverage the OE.

6.3. Perspectives

Analyzing these two areas it is possible to observe that they are converging in some aspects. Today we have still a separating each area. However, we believe that these mutual contributions tend to grow and such as semantic and ontologies will be make part of software development projects as the development of ontologies will be conducted more alike the software processes, well documented, systematically and observing quality issues.

This scenario allow us to design perspectives around a Semantic Software Engineering, which the software development will be based on ontologies, and an Ontology Engineering based on more mature methodologies, better suited to market demands.

6.4. Challenges

Although all increasingly growing efforts about OE there is still much to do. We believe that the maturity of OE will come just when its community adopt methodologies, techniques and tools to develop its products similar to what happens in SE today. The nowadays demands for ontologies in semantic web as well as in traditional areas like database management systems, distributed computing, artificial intelligence etc. are the main motivations for the progress of OE.

As it was discussed, the use of modules in ontology development can leverage all the ontology process development. Many works call attention to this, but no existing methodology deals with that yet.

7. Final Remarks and Future Works

The usage of ontology development methodologies is a crucial issue for the area evolution and maturing. The growing adoption of ontologies in the semantic web as well as other traditional areas computer science disciplines, such as databases and software engineering, leads to the development of applications that demands systematic ways to develop ontologies and the necessity to improve ontology quality.

In this paper we have presented a discussion about the convergence between software engineering and ontology engineering, mainly under the ontology development perspective. Thus, OE will benefit greatly from SE experiences since that there are many overlapping topics. Our strategy to develop the research that originated this work was to analyze how SE progress unfolded along the time, outlining the approaches being adopted that contributed to that and the main weaknesses that harmed the efforts being applied.

References

Anquetil, N., Oliveira, K., Software Maintenance Ontology, In: Ontologies for Software Engineering and Software Technology, Springer, pp. 154-173, 2006.

Abran et al., Engineering the Ontology for the SWEBOK: Issues and Techniques, in: Ontologies for Software Engineering and Software Technology, Springer, 2006.

Auer, S., Herre, H., Rapidowl - An Agile Knowledge Engineering Methodology. In Irina Virbitskaite and Andrei Voronkov, editors, Ershov Memorial Conference, volume 4378 of Lecture Notes in Computer Science,pp 424-430. Springer, 2006

Bertoa M. F., Vallecillo, A., and Garcia, F., An Ontology for Software Measurement, in: Ontologies for Software Engineering and Software Technology, Springer, 2006.

Bao, J. and Honavar, V. G, Divide and Conquer Semantic Web with Modular Ontologies - A Brief Review of Modular Ontology Language Formalisms. In: Proc. of the ISWC 2006 Workshop on Modular Ontologies. (2006)

Bechhofer S., Harmelen F., Hendler J., Horrocks I., McGuinness D.L., Patel-Scheineider P.F., Stein l. A., OWL Web Ontology Language Reference. Avaliable in http://www.w3.org/TR/owl-ref/. Last access in 02 September 2007.

Beck, K. Extreme Programming Explained: Embrace Change. Addison Wesley, 2000.

Blomqvist, E., State of the Art: Patterns in Ontology Engineering, Research Report 04:8. School of Engineering, Jnkping University, December, 2004.

Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., Yergeau, F., Cowan, J., Extensible Markup Language (XML) 1.1, (Second Edition), W3C Recommendation, 16 August 2006, edited in place 29 September 2006, available in http://www.w3.org/TR/2006/REC-xml11-20060816, last access in november 27, 2008.

D'Aquin, M., Sabou M., Motta E., Modularization: a Key for the Dynamic Selection of Relevant Knowledge Components. In: Proc. of the ISWC 2006 Workshop on Modular Ontologies. (2006)

Bezerra, C., Freitas, F., Euzenat. J., Zimmermann, A., ModOnto: A tool for modularizing ontologies, Second Workshop on Ontologies and Their Applications, Brazilian Symposium of Artificial Intelligence, Salvador, BA, Brazil, 2008.

Crnkovick, I., Component-Based Software Engineering, 10th International Symposium, CBSE 2007, Heinz Schmidt, Ivica Crnkovic, George Heineman, Judith Stafford, Springer, LNCS Series, Vol. 4608, ISBN: 978-3-540-73550-2, 2007

Devedzic, V., Understanding Ontological Engineering. Communications of the ACM, Vol. 45, pp 136-144, Abril, 2002.

Doran, P., Ontology Reuse via Ontology Modularization. In Proceedings of KnowledgeWeb PhD Symposium 2006 (KWEPSY2006) Budva, Montenegro, 2006.

Doran, P., Tamma, V., Iannone, L. Ontology Module Extraction for Ontology Reuse: An Ontology Engineering Perspective. Proceedings of the 2007 ACM CIKM International Conference on Information and Knowledge Management. Lisbon, Portugal. November 6-9, 2007.

Euzenat, J. et al, State of the Art on Ontology Alignement. Avaliable in http://www.aifb.uni- karlsruhe.de/WBS/meh/publications/euzenat04state.pdf. Last access em 12 de Outubro de 2007.

Euzenat, J., and Shvaiko, P., Ontology Matching, Springer-Verlag, 2007.

Euzenat, J., Zimmermann, A. and Freitas, F., Alignment-based modules for encapsulating ontologies, to appear in Bernardo Cuenca-Grau, Vasant Honavar, Anne Schlicht, Frank Wolter (Eds.): 2nd International Workshop on Modular Ontologies, WoMO 2007, Whistler, British Columbia Canada, October 28, 2007.

Fernandez-Lopez, M., Overview of Methodologies for Building Ontologies, Proceedings of IJCAI-99 workshop on ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden, August 2, 1999.

Fernandez-Lopez, M., Pazos A., Pazos J., Building a Chemical Ontology Using Methontology and the Ontology Design Environment, IEEE Inteligent Systems and their applications 4(1):37-46.

Fernandez-Lopez, M., Gomez-Perez A., Juristo N., METHONTOLOGY: From Ontological Art Towards Ontological Engineering, Spring Symposium on Ontological Engineering of AAAI, Stanford University, California, pp 25-34.

Freitas, F.; Stuckenschmidt, Heiner; Volz, Raphael (Eds.). Workshop on Ontologies and their Applications - WONTO'2004 - Proceedings, 2004, LivroRapido, Brazil.

Freitas, F., Stuckenschmidt, H., Noy, N., Ontology Issues and Applications: Guest Editors Introduction. 2005. Journal of the Brazilian Computer Society, Special Issue on Ontologias Issues and Applications, November 2005, Sociedade Brasileira de Computacao, Brazil.

Freitas, F., Euzenat, J., EXMO: Project Report, 2007.

Gamma, E., Vlissides, J., Helm, R., Johson, R., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.

Grau, B. C., A Logical Framework for Modularity of Ontologies, Avaliable in http://www.ijcai.org/papers07/Papers/IJCAI07-046.pdf. Last access in 06 de Junho de 2007.

Goméz-Pérez, A., Juristo, N., Montes, C., Pazos, J., Ingenieria Del Conocomiento: Diseno y Construccion de Sistemas Expertos. Ceura, Madrid, Spain.

Goméz-Pérez A, Knowledge Sharing and Reuse. In Liebowitz J (ed) Handbook of Expert Systems. CRC, Chapter 10, Boca Raton, Florida.

Gomez-Pérez, A., Fernandez-Lopez, M., Corcho, O., Ontological Enginering: with Examples from the areas of Knowledge Management, e-Commerce and the Semantic Web, Springer-Verlag, 2004.

Gruber, T., Toward Principles for the Design of Ontologies used for Knowledge Sharing. International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Publishers, Deventer, The Netherlands. 1993.

Gruninger, M.,Fox M., Methodology for the Design and Evaluation of Ontologies. In Skuce D (ed) IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing, pp 6.1-6.10.

Gangemi, A., Ontology Design Patterns for Semantic Web Content. In Motta E. and Gil Y., Proceedings of the Fourth International Semantic Web Conference, 2005

Herre, Rapidowl - An Agile Knowledge Engineering Methodology. In Irina Virbitskaite and Andrei Voronkov, editors, Ershov Memorial Conference, volume 4378 of Lecture Notes in Computer Science, pp 424-430, 2006.

J. Happel and S. Seedorf. Applications of Ontologies in Software Engineering. In 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006), held at the 5th International Semantic Web Conference (ISWC 2006), Athens, GA, USA, Nov. 2006.

Stuckenschmidt, H., Modularization of Ontologies. WonderWeb Deliverable D21, Avaliable in http://wonderweb.semanticweb.org/deliverables/ D21.shtml, éltimo access em 06 de Junho de 2007.

IEEE Standard Glossary of Software Engineering Terminology, IEEE Computer Society. New York. IEEE Std 610.121990.

IEEE Standard for Developing Software Life Cycle Processes. IEEE Computer Society. New York. IEEE Std 1074-1995.

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

Klyne, G., Carrol, J., Resource Description Format: Concepts and Abstract Syntax, availabe in http://www.w3.org/TR/rdf-concepts/, last access in november 27, 2008.

Kruchten, P., The Rational Unified Process: An Introduction, 3rd Edition, Addison-Wesley Object Technology Series, 2003.

Kosciansky, A., Santos, M., Qualidade de Software, 2nd. edition, Editora Novatec, 2007.

Larman, C., Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Prentice-Hall, 3rd Edition, 2004.

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

Liao, L., Qu, Y., Hareton, K., A Software Process Ontology and Its Application, ISWC 2005, Workshop on Semantic Web Enabled Software Engineering, 2005.

Manola, F., Miller, E., RDF Primer, 2004. Avaliable in http://www.w3.org/TR/rdf-primer/. Last access in 24 of august of 2007.

Mizoguchi, R., A Step towards Ontological Engineering, Procedings of the 12th National Conference on AI of JSAI, June, 1998, 24-31.

Newell, A., The Knowledge Level, Artificial Intelligence Magazine, 18(1), pp 87-127, 1982.

Noy, N., Uschold, M., Welty, C., Representing Classes As Property Values on the Semantic Web, available in http://www.w3.org/TR/swbp-classes-as-values/, W3C Working Group Note 5 April 2005, last access in January 26, 2009.

OMG, Object Constraint Language, available in http://www.omg.org/spec/OCL/2.0, Last access in november 27, 2008.

Pressman, R., Software Engineering. Preentice-Hall, 2004.

Mountain Goat, Uma Introdução ao SCRUM, Available in http://www.mountaingoatsoftware.com/system/hidden-asset/file/52/PortugueseScrum.pdf, last access in november 27, 2008.

Ruiz F., Hilera J., Using Ontologies in Software Engineering and Technology. In: Ontologies for Software Engineering and Software Technology. Springer. pp. 49-102. (2006)

Schlicht, A., Stuckenschmidt, H., Towards Structural Criteria for Ontology Modularization. In: Proc. of the ISWC 2006 Workshop on Modular Ontologies. (2006)

Simperl, E. P. B., Tempich C., Ontology Engineering: a Reality Check. In: 5th International Conference on Ontologies, Databases, and Applications of Semantics, ODBASE (2006).

Soydan, G. H., Kokar, M. M., An OWL Ontology for Representing the CMMI-SW Model, International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006), USA.

Stuckenschmidt, H., Implementing Modular Ontologies with Distributed Description Logics. In: Proc. of the ISWC 2006 Workshop on Modular Ontologies. (2006)

Stuckenschimidt, H., Klein, M., Modularization of Ontologies, In: Wonderweb: Ontology Infrastructure for the Semantic Web, IST Project 2001-33052, 2003. Avaliable in http://wonderweb.semanticweb.org/deliverables/documents /D21.pdf. Last Access in 5 de julho de 2007.

Spaccapietra, S., Rector, A., Napoli, G. Stamou, G. Stoilos, H.Wolger, J. Pan, M. D'Aquin, and V. Tzouvaras, Report on modularization of ontologies, Technical report, Knowledge Web Deliverable D.2.1.3.1, (2005).

Tempich, C., Ontology Engineering and Routing in Distributed Knowledge Management Applications, Doctorate Thesis, University of Karlruhe, 2006.

Uschold M, King M, Towards a Methodology for Building Ontologies. In: Skuce D (eds) IJCAI 95 Workshop on Basic Ontological Issues in Knowledge Sharing. Montreal, Canada, pp 6.1-6.10.

Waterman, A., A Guide to Expert Systems, Addison-Wesley, Boston, Massachussets, 1986.

Wikipedia, ISO 9000, available in http://en.wikipedia.org/wiki/ISO-9000, last access in 27 of november, 2008.

Wiederhold, Gio, An Algebra for Ontology Composition. Proceedings of 1994 Monterey Workshop on Formal Methods, Sept 1994, U.S. Naval Postgraduate School, pp 56-61. Monterey CA, USA.