Using Ontologies to Add Semantics to a Software Engineering Environment

Ricardo de Almeida Falbo, Fabiano Borges Ruy, Rodrigo Dal Moro
Computer Science Department, Federal University of Espírito Santo
Fernando Ferrari Avenue, CEP 29060-900, Vitória - ES - Brazil
{falbo, fruy}@inf.ufes.br

Abstract. Software Engineering Environments (SEEs) are systems designed to support software development and maintenance, and also for supporting project control and management. They provide means to integrate developers with the software process and the supporting technology. Since during software development many information resources are produced and used, it is very important to add semantics to them in order to improve the assistance given by the environment. In this context, ontologies are a key enabling technology for Semantic SEEs (SSEEs). A SSEE can be viewed as a SEE in which part of the information handled has associated a formal meaning (semantics), augmenting its tools' ability to work in conjunction with each other and with human developers. This paper discusses how ontologies are used in ODE, an Ontology-based software Development Environment, to make it a SSEE.

1. Introduction

Software development is a complex task, and thus it is essential to provide tool support for it. Stand alone CASE tools were the first initiative to provide this kind of support. Although these tools had significantly affected the practice of software development, their potential was limited by the difficulties involved in integrating them. This fact gave rise to the research in Software Engineering Environments (SEEs), which are integrated collections of tools that facilitate software engineering, supporting its activities across the software lifecycle [1]. SEEs have a history of about two decades, starting from supporting small fragments of the software process, until achieve the notion of process-centered SEEs [2].

Throughout this history, integration has been pointed out as one of the main challenges in the area. As SEEs evolve to incorporate knowledge about application domains, giving rise to Domain-Oriented SEEs [3], and about software engineering, incorporating knowledge management facilities [4], the integration problem seems to be more and more complex.

We believe that, to deal with this complexity, we need to treat semantics in SEEs, evolving them to Semantic SEEs (SSEEs). Semantics is related to the study of meaning. Ultimately, the relevance and success of an application system rest on what the symbols being manipulated by it mean in the real world. Not only what they mean, but, furthermore, to what extend people and other computer systems understand and agree with the meaning as implied by the system [5]. This is especially important to SEEs, since during software development many information resources are produced and used. Thus, it is essential to add semantics to them in order to improve the assistance given by the environment.

In this context, ontologies are a key enabling technology for SSEEs. A SSEE can be viewed as a SEE in which part of the information handled has associated a formal meaning (semantics), given by ontologies, augmenting its tools' ability to work in cooperation with each other and with human developers.

This paper presents how ontologies are being used in ODE [6], a process-centered SEE, in order to evolve it to a SSEE. Section 2 discusses briefly the evolution of SEEs, and highlights the need to deal with semantics as they become more complex. Section 3 presents ODE, and discusses how ontologies are used in it. Section 4 presents related works. In section 5, we report our conclusions.

2. Software Engineering Environments Evolution

The first generation of CASE tools supporting software process activities provided support only for single activities, without any real means of integrating tools. The identification of the need for integrated support for software engineering activities throughout the software lifecycle represents the genesis of Software Engineering Environments (SEEs) [1].

The first SEEs, however, did not support any notion of software process. To deal with this drawback, Process-centered Software Engineering Environments (PSEEs) emerge, with the goal of assisting in the modeling and automation through enactment of software processes [2].

The explicit representation of software processes is the foundation on which modern integrated development environments are built [1]. But, as the complexity of

software processes increases, SEEs have to evolve to offer a wider support to software developers.

Today, the use of knowledge during software development is being considered very important to support software development activities. Several different kinds of knowledge are useful in this context, including domain knowledge, past experiences, knowledge about software engineering, and so on. This claim represents the origin for Domain-oriented SEEs (DOSEE) [3] and for the use of Knowledge Management (KM) in SEEs [4].

Nowadays knowledge is viewed as one of the most valuable organization's assets, and thus, the importance of managing it is widely recognized. DOSEE extends the traditional notion of PSEE by introducing into it domain knowledge to guide software developers through several software development activities [3]. SEEs with KM support extends this view, allowing managing any kind of software engineering knowledge. Using a KM approach, knowledge created during software processes can be captured, stored, disseminated, and reused, so that better quality and productivity can be achieved.

In any time of the history of SEEs, the notion of integration is considered to be essential. Tool integration is about the extent to which tools agree, and it involves several dimensions such as [7, 6]:

- Presentation: refers to improving the efficiency and effectiveness of the user's interaction, considering the environment and its tools.
- Data: deals with the way the tools and the environment share data.
- Control: aims to support sharing functionalities between the environment and its tools.
- Process: intends to ensure that the tools interact effectively in support of a defined process, linking the tools and the process.
- Knowledge: refers to managing the knowledge captured during the software projects, and offering knowledge-based support to software engineers.

In any of these dimensions, we can see that the tools must share an understanding of the meaning, i.e. we need semantics. Semantics is often defined as the study of the meaning. In the case of computer-based applications, semantics is not only related to what the symbols being manipulated by an application system mean, but also to what extend people and other computer systems understand and agree with the meaning as implied by the system [5]. Looking semantics this way, we can clearly see that it pervades all the integration dimensions presented before: (i) presentation is directly related to the degree people and systems agree with the meaning of the human-computer interfaces; (ii) data and control integration are also extremely dependent on semantics, since tools must agree on the data structure, as well as the services provided by other tools and by the environment; (iii) process integration depends on semantics, since all the tools and the environment should share a common understanding of what is a software process; (iv) finally, semantics is fundamental for knowledge integration.

Computer systems are virtually impossible without semantic. But the degree to which a system is semantically aware varies greatly [5]. If a system has a high degree of semantic precision (i.e. the information in it is semantically tagged to a specific level of discernment), and a high degree of semantic veracity (i.e. the system implements procedures to ensure that the information is valid), then it is said to be a highly semantically aware system. In fact, semantic precision and semantic veracity are part of a broader issue that looks for answering questions such as [5]: How do we name things, how do we form categories, how do we ensure some constraints, and how do these aspects affect the systems we build? This is the subject of ontologies.

An ontology is a logical theory accounting for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualization of the world [8]. It consists of concepts, relations, properties and constrains expressed as axioms [9].

The importance of ontologies to express semantics is recognized in several areas, such as Semantic Web and Knowledge Management [10]. These areas have in common the problem of continued rapid growth in information volume, which makes it difficult to find, organize, access and maintain information. The use of machine-processable metadata based on ontologies is being pointed as one of the most promising ways to deal with this problem. As "data about data", metadata is almost pure semantics, that is, it stores the meaning of the data it describes [5].

Looking to the benefits that this approach has given to related areas, we claim that it can also be applied to evolve SEEs to Semantic SEEs. During a software project, many information resources are produced and consumed, and, in several situations, it is essential to establish connections between the information resources in order to obtain the required set of information to support performing an activity. In this case, ontologies can be used to establish a common understanding about the software engineering domain, application domains and tasks. Annotating SEE's information resources with ontology-based metadata, we can add semantics to them, and this will enable a SEE that provides a qualitatively new level of services. Next, we discuss how ontologies are being using in ODE [6], a PSEE, in order to evolve it to a Semantic SEE.

3. An Ontology-based Software Engineering Environment

ODE (**O**ntology-based software **D**evelopment **E**nvironment) [6] is a PSEE that is developed grounded

on ontologies. ODE's design premise considers that, if the tools in a SEE are built based on ontologies, integration can be improved. The same ontology is used for building different tools supporting related software engineering activities, and, if the ontologies are integrated, integration of tools built based on them can be highly facilitated [6].

ODE is implemented in Java and has several tools, such as tools supporting software process definition, risk analysis, estimation, and object modeling, among others. The environment and some of its tools are developed based on some software engineering ontologies. The most important of them is the software process ontology [9], since it describes the main concepts involved in software processes, such as process, project, activity, artifact, resource, procedure, and so on. The others ontologies (software quality ontology, software artifact ontology, software risk ontology and software organization ontology) are integrated to it, forming a net of concepts. Figure 1 shows part of this ontology using an extension of UML. In this extension, some axioms were assigned to UML's elements. For example an axiom treating transitivity is assigned to the aggregation notation [11].

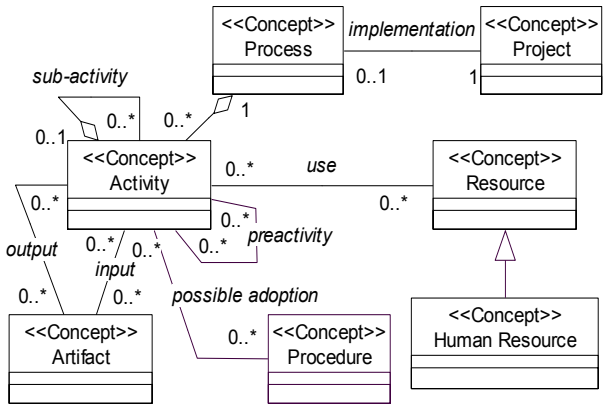

Figure 1- Part of the Software Process Ontology.

Besides the axioms associated to the notation, called epistemological axioms, other axioms are considered in this ontology (ontological axioms), such as the one that defines *pre-activities* from input and output relations [9]: $(\forall a_1,a_2)(\exists s)(input(s,a_2) \wedge output(s,a_1)) \rightarrow preactivity(a_1,a_2)$.

Since ODE is based on ontologies and implemented in Java, we needed to map ontologies into object models. This mapping was done based on the systematic approach for deriving object models from ontologies, described in [12]. In order to maintain the semantic binding among ODE's objects and, thus, to incorporate ontologies in it, its conceptual architecture[1] was designed in three levels:

- The Ontological Level (OL) concerns describing ontologies. Its model corresponds to the ODE's meta-ontology. Thus, the main goal of the OL is to register ontologies in ODE. Its instances are used to guide the definition of the other levels, originating the main classes of both the meta-level and the base level.
- The Meta-level (ML) encompasses the classes that describe knowledge about some part of the software engineering domain. Its classes are derived from the ontologies, and the corresponding instances act as knowledge about the objects in the base level. ML classes are directly derived from the ontologies.
- The Base Level (BL) defines the classes that implement ODE's applications, i.e. its tools and functionalities. Several of its classes are also derived from the ontologies, but, typically, they incorporate other details that are necessary only to implement applications, and thus that are not described in the ontologies. Thus, the BL also contains classes, associations, attributes and operations that do not have a counterpart in the ontological and meta levels.

This approach facilitates the establishment of a correlation between objects in the three levels, since one level serves as metadata for the others. Thus, it is possible to annotate the objects with semantic information given by the ontologies, as discussed next.

Describing Ontologies in the Ontological Level

To allow defining ontologies in ODE, a graphical ontology editor, called ODEd [11] was developed. ODEd supports ontology development through defining concepts, relations, properties, and axioms. ODEd uses the meta-ontology model shown in Figure 2 to store ontologies in the ontological level. This model is design using the Meta-Object Facility (MOF) model [13].

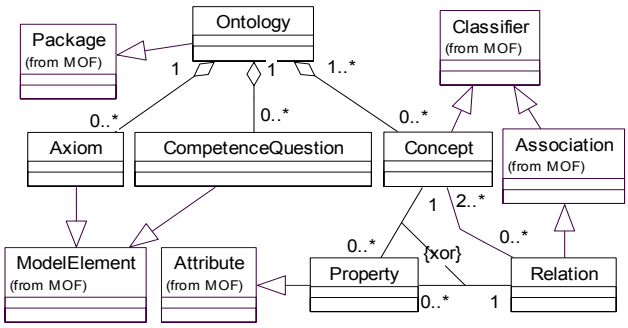

Figure 2- Part of ODEd's Meta-Ontology class model.

At this level, instances of these classes represent the elements of an ontology in ODE. Taking as example the software process ontology, we have the *Software Process Ontology* itself as an instance of the `Ontology` class; *Activity* and *Resource* are instances of the `Concept` class; *input*, *output*, and *pre-activity* are examples of instances

[1] The term "conceptual architecture" is being used to designate a high level decomposition of the ODE's packages, in opposition with the actual software architecture in layers used to effectively implement the environment.

of the `Relation` class; and, finally, there are, some competence questions and axioms (not shown graphically), which are treated as instances of `CompetenceQuestion` and `Axiom` classes, respectively.

Deriving Meta / Base Level Classes from Ontologies

Once an ontology is defined at the ontological level, it is possible to derive the objects models for the other two levels. Following the derivation approach proposed in [12] concepts, relations and properties are mapped, respectively, to classes, associations and attributes in an object model. Moreover, axioms are mapped to methods.

Since ODE's conceptual architecture has, beyond the ontological level, other two levels, the derivation process does not originate just one object model, but two. Thus, in the derivation process, a certain concept can originate a class only at the meta level, only at the base level, or at both levels. Table 1 shows this mapping for the part of the software process ontology shown in Figure 1. Figures 3 and 4 show respectively the object model derived for the Meta and the Base levels. It should be pointed out that the meta-level classes are named with the prefix K, and they are subclasses of a `Knowledge` class. Following, we discuss the rationale behind this mapping.

Table 1. Concepts and the classes derived from them.

Concepts	Meta Level Classes	BaseLevel Classes
Activity	`KActivity`	`Activity`
Artifact	`KArtifact`	`Artifact`
Human Resource	`KHumanResource`	`HumanResource`
Procedure	`KProcedure`	-
Project	-	`Project`
Process	`KProcess`	`Process`

- **Classes are created at the both levels.**

Many times, a concept should give rise to classes at both levels: at the meta-level (ML), determining the "type" of concrete objects of the real world, and at the base level (BL), representing the real world objects themselves. In this case, ML objects are used to describe knowledge about the BL objects. For instance, as shown in Table 1, the *Human Resource* concept, which is an instance of the `Concept` class in the ontological level, originates two classes: *KHumanResource* and *HumanResource*. The first class represents the kinds of human resources potentially important in software processes, such as *Software Engineer*, *Project Manager*, etc. These instances of the meta-level are used to classify the concrete objects in the base level (*John*, *Mary*, *Peter*, *Ann*, etc). This way, it is possible to know that *John* is a *Software Engineer*, which is a *Human Resource*.

In an analogous way, the *Activity* concept originates the classes `KActivity` and `Activity`, which can be used to describe, respectively, activities types (for instance, *Planning*, *Requirements Specification*, etc) and concrete activities performed in the context of a specific project (for instance, *Initial Planning of the X Project*, *Preliminary Requirements Specification of the X Project*, etc). In this case, once the `Activity` class is annotated with a reference to an instance of the `KActivity` class (see Figure 4), it is possible to point that *Initial Planning of the X Project* is an activity of the *Planning* type.

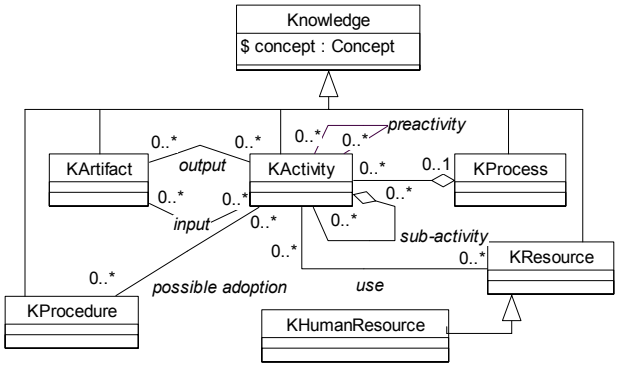

Figure 3- Software Process Meta-Level Model.

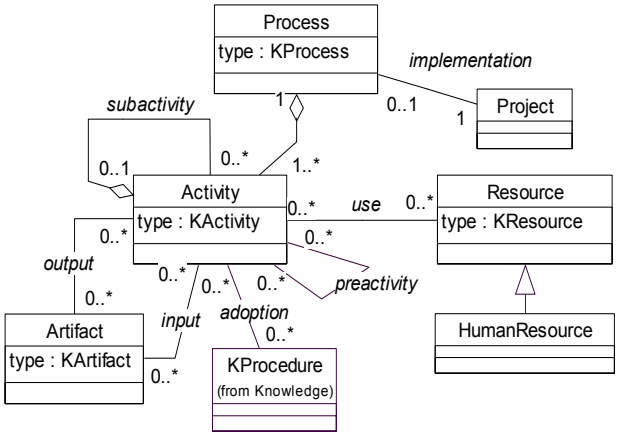

Figure 4- Software Process Base Level Model.

Once the software process ontology defines that "activities use resources", we say in the meta-level, that activities of the type *Requirements Specification* require human resources of the type *Software Engineer*. During human resources allocation to a specific project, e.g. *X Project*, this information is used to point that the *Preliminary Requirements Specification of the X Project* activity can allocate *John* (a *Software Engineer*), since he is a human resource compatible with the needs for performing this activity. Thus, the meta-level is used to guide the accomplishment of the base level activities.

- **A class is created only at the Meta/Base Level**

Some concepts may be relevant only in one of the other two levels. In this case, if it is an abstract entity, only one class is created in the Meta-Level. For example, the concept *Procedure* has instances as *Use Case*

Modeling, *Code Inspection*, etc. These instances describe knowledge about abstract procedures generally adopted when an activity is performed. They are enough for both the meta and the base levels. That is, it is possible to say, at the meta-level, that activities of the *Requirements Specification* type can adopt the *Use Case Modeling* technique. On other hand, at the base level, we can also say that the *Preliminary Requirements Specification of the X Project* is accomplished applying *Use Case Modeling*. Since the derived class describes knowledge about the procedures that can be adopted in software development, it is created only at the meta-level (`KProcedure`). As the base level has access to the meta-level, it can use this class when necessary.

On the other hand, in some cases, certain concepts should derive classes only at the base level. This happens when the concept has only one relevant level of instances and it can be viewed as a concrete entity. In this case, those instances are important only for the base level, because a type is not necessary, but the real world concrete objects are. This is the case of the *Project* concept. A project is a concrete entity, like *X Project*. Thus, this concept is mapped only to the base level, originating the `Project` class.

Focusing on the dependencies among ODE's levels, we can see that the base level depends on the meta-level, which in turns depends on the ontological level. These dependencies occur because, when applicable, a base level class has an association with its correspondent class at the meta-level (shown in Figure 4 as attributes), which, in turns, is associated to its concept in the ontological level (through a static reference in the `Knowledge` class, as shown in Figure 3). This way, we annotate ontology concepts in ODE's objects. This annotation allows identifying the concept from which an object is derived from. Thus, the navigation among the levels is simplified and several environment tasks (such as process definition and resource allocation) are better supported, some of them using inferences.

The potential of ontologies can be broadly explored if their axioms are used to derive knowledge through inferences. Although axioms can be mapped to methods, in some cases, a more interesting approach is to map axioms to rules that could be manipulated by inference engines. To deal with this, ODE has an inference layer, which allows describing rules and facts in Prolog.

As an example of the use of inferences in ODE, consider the following axiom of the software process ontology: $(\forall a,b,r)(use(a,r) \wedge subactivity(a,b)) \rightarrow use(b,r)$. This axiom says that if an activity *a* uses a resource *r*, and *a* is a sub-activity of another activity *b*, then *b* also uses *r*. This axiom was mapped to the rules shown in Figure 5, which are used during process definition to present suggestions of resources that a certain activity can use.

```
% Sub-activity Rule (A is sub-activity of B)
subActivity(A, B) :- subActivity_(A, B).
subActivity(A, B) :- subActivity_(A, X),
                     subActivity(X, B).
% Resource Rule (Activity Y uses Resource R)
use(Y, R):- subActivity(Z, Y), use(Z, R).
```

Figure 5- Some Prolog rules converted from axioms.

It is worthwhile to point out that the determination of which activities can use which resources is made not only using these axioms, but also using the ontological annotations. From a concrete activity, it is possible to go to the meta-level (through its `KActivity`) and determine which types of human resources are required to perform this type of activity (the rules above are used in this step). Knowing which types of human resources are necessary, the human resource allocation tool suggests some human resources (base level) that plays the required role.

In several other situations, this approach is applied. To illustrate a situation where the three annotation levels are used, let's take a look at knowledge retrieval in ODE's Knowledge Management (KM) infrastructure [4], partially shown in Figure 6. When searching for some knowledge items stored in ODE's organizational memory, several filtering criteria can be used. One of the most important criteria, however, is the ontology itself. The user can select the ontology and some of its concepts that are related to the items he/she wants to search for (ontological level). Then, instances of these concepts are listed (objects from the meta-level), and the user can select the "types" of the items wanted. Based on that, artifacts and other knowledge items (objects from the base level) are retrieved and presented.

In the example of Figure 6, the user selected the software process ontology and the activity concept as retrieval criteria. Instances of this concept (instances of the class `KActivity`) are presented and he/she can select the ones that refer to the items he/she is looking for. Based on the selected instances of the meta-level (in the example, Planning and Estimation), knowledge items (instances of the base level) are presented, including artifacts (project plans) and lessons learned and packages of messages related to Planning and Estimation.

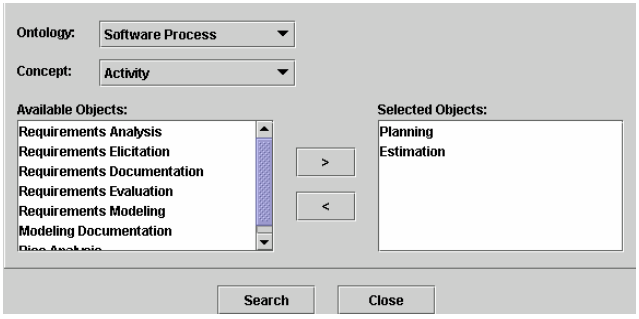

Figure 6- Search Service of ODE's KM Infrastructure.

4. Related Work

Few works have explored semantic issues in SEEs. Brown and McDermid [14] were ones of the firsts to talk about that. They proposed a classification of tool integration levels that includes what they call "semantic level", which could be achieve through including metadata in the SEE's repository.

More recently, Oliveira et al. [3] proposed the use of ontologies in Domain Oriented SEEs, and use the TABA Workstation to apply their ideas. Moreover, in the context of TABA Project, ontologies are also used to structure the environment and to support knowledge management [15]. However, the use of ontologies in TABA does not consider some aspects that we advocate as very important, such as incorporating constraints defined as axioms and the use of inferences to support the accomplishment of some software development activities.

Concerning the way how ODE uses metadata based on ontologies, it was defined based on several works done in the ontology field of study, such as the one done by Guarino [8]. Also, our approach is in conformity with the OMG Meta-modeling Architecture [13]. The base level corresponds to the User Object Layer (M0), the meta-level to the Model (Metadata) Layer (M1), the ontological level to the Meta-model Layer (M2), and the MOF model to the Meta-meta-model Layer (M3).

5. Conclusions

During the software development, many information resources are produced and generated. In several situations, it is essential to establish connections between them in order to "collect the dots", i.e. to get a set of items that are useful to support the accomplishment of an activity. In this context, it is very important to capture the semantics of the items in the SEE's repositories, evolving them to Semantic SEEs. Ontologies play a crucial role to Semantic SEEs, since they can be used to annotate information resources with semantic metadata.

In this paper, we presented our approach to evolve ODE, a Process-centered SEE, to a Semantic SEE. This approach is based on annotating ODE's objects with ontology-based metadata, and proved to be very useful. But, some aspects should be improved. First, we should investigate better ways to incorporate inference services in ODE. The use of Prolog showed to be not enough for our purposes. Some studies to use other inference mechanisms and a modern ontology language, such as OWL, and are being made. Other research directions that we are following include improving our approach to derive object models from ontologies and other mechanisms to support ontology-based browsing of ODE's resource information, such as semantic maps.

Acknowledgments

This work was accomplished with the support of CNPq and CAPES, entities of the Brazilian Government reverted to scientific and technological development.

References

[1] W. Harrison, H. Ossher, P. Tarr, "Software Engineering Tools and Environments: A Roadmap", in Proc. of the Future of Software Engineering, ICSE'2000, 263-277, Ireland, 2000.

[2] S. Arbaoui, et alli, "A Comparative Review of Process-Centered Software Engineering Environments", Annals of Software Engineering 14, 311-340, 2002.

[3] K.M. Oliveira, F. Zlot, A.R. Rocha, G.H. Travassos, C. Galotta, C.S. Menezes, "Domain-oriented software development environments", The Journal of Systems and Software 72, 145-161, 2004.

[4] R.A. Falbo, D.O. Arantes, A.C.C. Natali, "Integrating Knowledge Management and Groupware in a Software Development Environment", in Proc. of the 5th Int. Conf. on Practical Aspects of Knowledge Management, 94-105, Austria, 2004.

[5] D. McComb, Semantics in Business Systems: The Savvy Manager's Guide. Morgan Kaufmann Publishers, 2004.

[6] R.A. Falbo, A.C.C. Natali, P.G. Mian, G. Bertollo, F.B. Ruy, "ODE: Ontology-based software Development Environment", in Proc. of the IX Argentine Congress on Computer Science, 1124-1135, La Plata, Argentina, 2003.

[7] I. Thomas, B.A. Nejmeh, "Definitions of Tool Integration for Environments", IEEE Software, 29-35, March 1992.

[8] N. Guarino, "Formal Ontology and Information Systems", In: Formal Ontologies in Information Systems, N. Guarino (Ed.), IOS Press, 3-15, 1998.

[9] R.A. Falbo, C.S. Menezes, A.R.C. Rocha. "A Systematic Approach for Building Ontologies". In Proc. of the 6th Ibero-American Conference on Artificial Intelligence, Lisbon, Portugal, LNCS, vol. 1484, 349-360, 1998.

[10] J. Davies, D. Fensel, F. van Harmelen. Towards the Semantic Web: Ontology-driven Knowledge Management, John Wiley & Sons, 2003.

[11] P.G. Mian, R.A. Falbo, "Supporting Ontology Development with ODEd", Journal of the Brazilian Computer Science, vol. 9, no. 2, 57-76, November 2003.

[12] R.A. Falbo, G. Guizzardi, G., K.C. Duarte, "An Ontological Approach to Domain Engineering", in Proc. of the 14th Int. Conference on Software Engineering and Knowledge Engineering, 351- 358, Ischia, Italy, 2002.

[13] OMG, Meta Object Facility (MOF) Specification, Version 1.3, March 2000.

[14] A.W. Brown, J.A. McDermid, "Learning from IPSE's Mistakes", IEEE Software, 23-28, March 1992.

[15] G. Santos, K. Villela, L. Schnaider, A.R. Rocha, G.H. Travassos, "Building Ontology-based Tools for a Software Development Environment", in Proc. of the 6th Int. Workshop on Learning Software Organization, 19-30, Canada, July 2004.