Supporting Ontology Axiomatization and Evaluation in ODEd

Vítor Estêvão Silva Souza, Ricardo de Almeida Falbo
Computer Science Department, Federal University of Espírito Santo
Av. Fernando Ferrari, CEP 29060-900, Vitória – ES, Brazil
vitorsouzabr@yahoo.com.br, falbo@inf.ufes.br

Abstract

Recently, we have seen an increasing interest in ontologies as artifacts to represent knowledge and as critical elements in knowledge management, requirements engineering and several other application areas. In Domain-Oriented Software Development Environments, ontologies are used as domain models that can be used to guide requirements engineering. Thus, in this kind of environment, tools supporting ontology development are necessary. In the context of ODE (Ontology-based software Development Environment), we developed ODEd, an ontology editor. The initial version of ODEd, however, offered limited support for defining axioms and for ontology evaluation. In order to overcome such limitations, we improved ODEd adding an axiom editor to it. In this paper we discuss how this axiom editor supports axiom definition and ontology evaluation using an inference engine.

1. Introduction

As pointed by Brewster and O'Hara [1], recently, we have seen an explosion of interest in ontologies as artifacts to represent human knowledge in several areas of Computer Science, such as Knowledge Management, Semantic Web and Software Engineering, among others. In Software Engineering, ontologies are useful as domain models [2], for structuring Software Development Environment (SDE) [3], and for improving learning software organizations [4].

An important goal of using ontologies in Software Engineering is to describe domain knowledge to support software development in that domain. In this case, ontologies can be used: (i) for establishing a shared conceptualization about the domain, (ii) as a basis for software specification and development, allowing knowledge reuse, and (iii) for searching information repositories for desired resources, improving precision and reducing the overall amount of time spent in searching. All these uses of ontologies match with the purposes of Domain-Oriented Software Development Environments (DOSDEs) [5]. DOSDEs are a special class of SDEs that uses domain knowledge to guide software developers across the several phases of the software process. DOSDEs organize the application domain knowledge facilitating problem understanding during system development [5]. In this way, DOSDEs can use ontologies as a model that makes explicit the basic conceptualization of the domain.

But building ontologies is not a simple task. It involves the specification of concepts and relations that exist in the domain, besides their definitions, properties and constraints, described as axioms [6]. Therefore, tools supporting ontology development are necessary. These tools must support definition of concepts, relations, properties, and constraints, and must enable the inspection, browsing, and coding of the resulting ontologies [7].

In order to support ontology development in a software engineering environment called ODE (Ontology-based software Development Engineering) [8], we developed ODEd [9], an ontology editor. The main goal of ODEd was to support domain orientation in ODE, turning it into a DOSDE. In its initial version, ODEd allowed the definition of concepts, relations and properties, using graphic representations, and the description of some pre-defined classes of axioms, such as transitivity and symmetry. Also ODEd supported the derivation of object frameworks and hypertexts from ontologies.

However, ODEd presented some problems, most of them concerning ontology axiomatization. To deal with those problems, we built AxE, an Axiom Editor integrated to ODEd, which allows defining axioms in a general way. AxE has an inference engine integrated to it, to better support ontology evaluation.

In this paper we present AxE. Section 2 briefly discusses some aspects of the use of ontologies in DOSDEs and presents SABiO [6,10], the ontology development process that underlies ODEd functionalities. Section 3 presents an overview of the first version of ODEd and points some of the problems detected. Section 4 presents AxE, discussing how it treats ontology axiomatization and evaluation. In section 5, we discuss related works. Finally, section 6 reports our conclusion and future work.

2. Ontologies and Domain Oriented Software Development Environments

To represent a phenomenon or part of the world, which we call a domain, it is necessary to focus on a limited number of concepts that are sufficient and relevant to create an abstraction of the phenomenon at hand. Thus, a central aspect of any modeling activity, including various Software Engineering activities, consists of developing a conceptualization [11]. An ontology is an explicit specification of a shared conceptualization. In this context, a conceptualization refers to an abstract model of how people think about things, usually restricted to a particular subject area. An explicit specification means that concepts and relations of this abstract model are given explicit terms and definitions [12].

According to Guarino [13], "an ontology refers to an engineering artifact, constituted by a specific *vocabulary* used to describe a certain reality, plus a set of explicit assumptions regarding the *intended meaning* of the vocabulary words. This set of assumptions has usually the form of a first-order logical theory, where vocabulary words appear as unary or binary predicate names, respectively called concepts and relations. In the simplest case, an ontology describes a hierarchy of concepts related by subsumption relationships; in more sophisticated cases, suitable axioms are added in order to express other relationships between concepts and to constrain their intended interpretation".

One of the main benefits of the use of ontologies in Software Engineering is to reuse domain specifications. In traditional Software Engineering, for each new application to be built, a new conceptualization is developed. In an ontology-based approach, requirement elicitation and modeling can be accomplished in two stages. First, the general domain knowledge can be elicited and specified as ontologies. These ontologies are used to guide the second stage of the requirement analysis, when the particularities of a specific application are considered. This way, the same ontology can be used to guide the development of several applications [6]. In fact, in a more general way, ontologies can be useful for developers understand the application domain in which the system being built will take place. This is the basic premise for Domain-Oriented Software Development Environments (DOSDEs) [5]. DOSDEs extend the traditional notion of Software Development Environment (SDE) by introducing into it domain knowledge (in the form of domain ontologies) to guide software developers through the several software development phases [5].

A DOSDE, like any other SDE, should have a repository storing all the information related to the software projects, and a set of tools to support the activities of the software process. On the other hand, this new class of SDE requires two additional features: representation of domain knowledge and use of this knowledge during software development [5]. Thus, we need to integrate an ontology editor into a DOSDE, in order to support ontology development and use in it. Ideally, this ontology editor should support an ontology-based domain engineering approach, allowing the development of ontologies and their use in some software process activities.

Falbo et al. [2] proposed an ontological approach to domain engineering that considers ontology development (domain analysis), its mapping to object models (infrastructure specification) and Java frameworks development (infrastructure implementation). This approach seems to be very useful for DOSDEs, because it deals with ontology development and latter use as object frameworks.

To perform the ontology development process, a method for building domain ontologies called SABiO (Systematic Approach for Building Ontologies) [10] is suggested. SABiO encompasses the following activities [6, 10], shown in Figure 1:

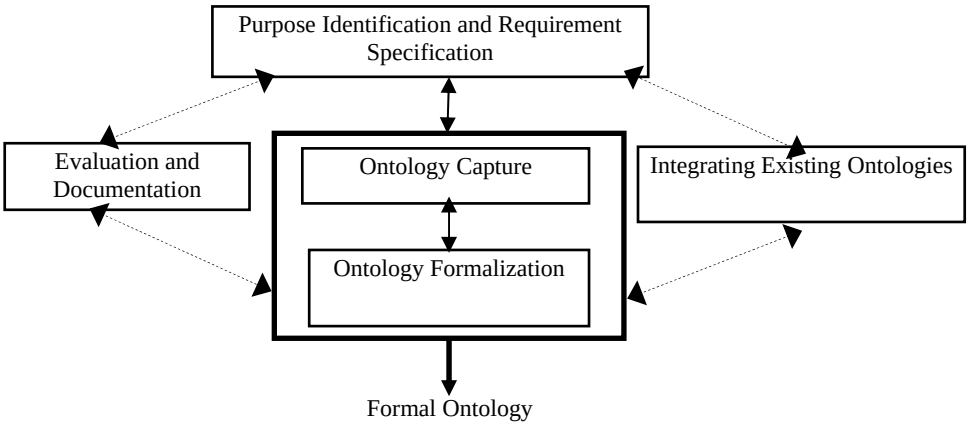

Figure 1: SABiO's development process

- Purpose identification and requirement specification: concerns to clearly identify the ontology purpose and its intended uses, i.e. the competence of the ontology. To do that, competency questions should be used. Competency questions are the questions that the ontology should be able to answer [14];
- Ontology capture: the goal is to capture the domain conceptualization based on the ontology competence. Relevant concepts and relations should be identified and organized. A model using a graphical language and a dictionary of terms should be used to aid communication with domain experts. Also, constraints should be written down as axioms, in this phase written in natural language;
- Ontology formalization: aims to explicitly represent the conceptualization captured in a formal language. When a logical formalism is adopted, concepts, relations and properties should be mapped to predicates, and formal axioms can be written as sentences using these predicates (what we call axiomatization);
- Integration of existing ontologies: during ontology capture or formalization, it could be necessary to integrate the current ontology with existing ones, in order to use previously established conceptualizations;
- Ontology evaluation: the ontology must be evaluated to check whether it satisfies the specification requirements. Since SABiO advocates the use of competency questions to capture the ontology requirements, one of the most important checks to be done is to verify if the ontology is able to answer the competency questions. To automate this evaluation process, during ontology formalization, it is interesting to choose a formal language that is supported by an inference engine. This way, we can formalize the competency questions and pose them as questions to be answered by the ontology;
- Documentation: all the ontology development process must be documented, including purposes, requirements and motivating scenarios, textual descriptions of the conceptualization, the formal ontology and the adopted evaluation criteria.

Once we have the ontology, an object framework can be generated from it. This framework can be added to the DOSDE repository as a domain-oriented reusable artifact, and can be reused later in activities such as design and implementation. The ontology-based domain engineering approach [2] proposes a set of directives, design patterns and transformation rules for deriving Java frameworks from ontologies. These directives are used to guide the mapping from the epistemological structures of the domain ontology (concepts, relations and properties) to their counterparts in the object-oriented paradigm (classes, associations and attributes, respectively), and to implement them as Java classes.

Finally, as pointed by Oliveira et al. [5], the use of a DOSDE introduces a change in the software development process. In several activities of the software process (such as requirements analysis, design, and so on), a sub-activity called domain investigation should be introduced, specifically targeted to take advantage of the knowledge available. Domain investigation concerns studying the application domain and to support this task, it is worthwhile to generate hypertexts from the ontologies.

ODEd partially supports the ontology-based domain engineering approach described above. In the next section, we present it, focusing on its initial version. Based on the problems detected in this initial version, we improved ODEd, adding to it an axiom editor that is presented in section 4.

3. ODEd: An Ontology Editor for making ODE a DOSDE

ODEd was developed to support domain engineering in ODE (Ontology-based software Development Environment) [3, 8], so that ODE could evolve to a Domain Oriented Software Development Environment (DOSDE). To do this, ODEd partially supports the ontology-based domain engineering approach discussed in section 2. This section presents briefly the main features of the first version of ODEd and present some of the problems detected when using it. To better clarify how ODEd supports ontology development, we also present part of the activity ontology [6], which is used as an example. As shown in figure 2, some concepts of this ontology are activity, artifact and resource. An activity can be decomposed into sub-activities and can be preceded by other activities. Activities produce and consume artifacts. Finally, to be accomplished, activities require resources, including human, software and hardware resources.

Several constraints should be taken into account in this context, deriving some axioms. For example:

A1. The precedence relation is transitive, then if *a1* is preactivity of *a2* and *a2* is preactivity of *a3*, then *a1* is also preactivity of *a3*: $\forall$ *(a1,a2, a3) preActivity (a1, a2)* $\wedge$ *preActivity (a2, a3)* $\rightarrow$ *preActivity(a1, a3)*;

A2. If *a* is preactivity of *b* and *a1* is part of *a*, then *a1* is also preactivity of *b*: $\forall$ *(a, b, a1) preActivity(a, b)* $\wedge$ *subActivity (a1, a)* $\rightarrow$ *preActivity(a1,b)*;

Concerning ontology development, the first version of ODEd [9] supported the following activities of SABiO:

- Purpose identification and requirement specification: ODEd allowed the definition of informal competency questions, that is, competency questions in natural language. These competency questions were used only for documentation purposes;
- Ontology capture: during ontology capture, the use of a graphical representation is essential in order to facilitate the communication between domain engineers and experts. Thus, ODEd supported the definition of concepts, relations and properties using a graphical language (LINGO [6] or UML [9]). Figure 2 shows part of a software activity ontology, written in UML. ODEd also supported a limited way to define constraints, only allowing the description of those imposed by relations, called association axioms. Association axioms allow the classification of relations as being reflexible, irreflexible, symmetric, anti-symmetric, atomic, disjointed, exclusive and transitive;
- Ontology formalization: the first version of ODEd did not directly treat ontology formalization, since only the predefined types of association axioms could be defined. However, for this type of axioms, an axiomatization was automatically provided. In this version, axiom A1, described above, could be formalized, but axiom A2 could not;
- Integration of existing ontologies: ODEd supported ontology integration in a very simple way. It was possible to import concepts from existing ontologies to the current one. If more than one concept was imported and there was a relation between them, that relation was also imported to the ontology;
- Ontology evaluation: the first version of ODEd did not directly support ontology evaluation. The ontology engineer could instantiate the ontology and try to manually check if the ontology was answering the competency questions;
- Documentation: although ODEd stored several ontology development information, such as competency questions and ontology diagrams, it did not generate any documents.

As pointed in section 2, after developing an ontology as a domain model, it is useful to map this model to a reusable framework (domain design and implementation). ODEd automatically generates object frameworks from ontologies in Java [9], considering also the association axioms.

Finally, to support domain investigation, ODEd provides automatic generation of hypertexts based on the ontologies designed. Using these hypertexts, developers can browse and search the domain concepts, relations, properties and constraints to learn about them [9].

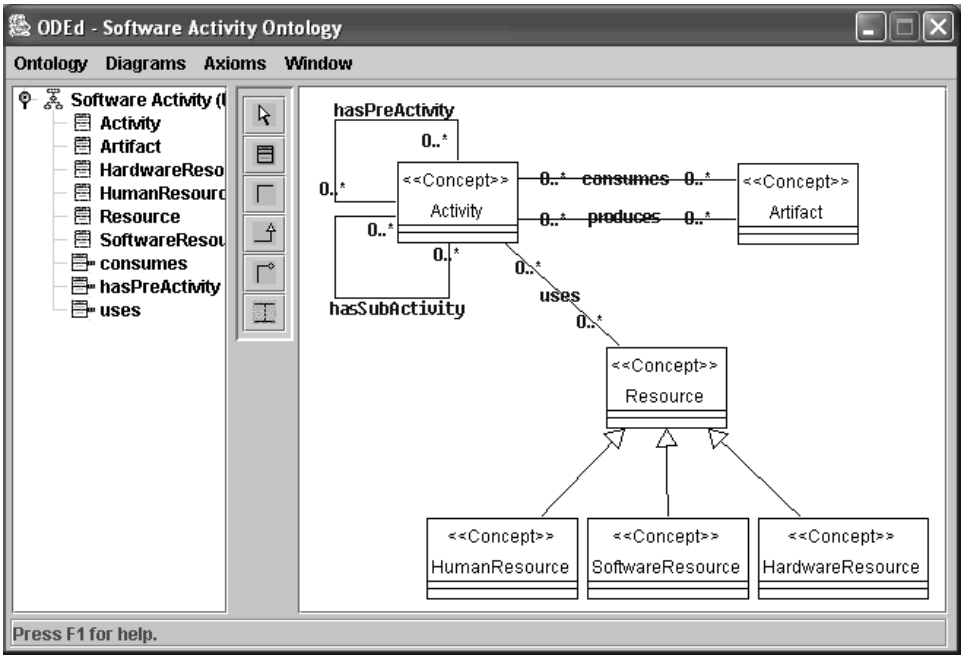

Figure 2: Software Activity Ontology in ODEd

Although most phases of ontology development process were supported by ODEd, there were many aspects to be improved, such as [9]:

- ODEd should support the definition of formal competency questions. This feature is related to ontology evaluation: once competency questions are formalized, they can be used to evaluate if the ontology satisfies its requirements. To support these features, ODEd should support an ontology language, and reasoning services are necessary;
- Only certain types of axioms could be captured in ODEd. Other axioms which did not fit in these categories were not treated. So, it is important to improve axiomatization in ODEd, allowing free definition of axioms. To do that, ODEd should support an ontology language;
- Ontology evaluation was not properly supported. Formalizing competency questions and axioms, ontology evaluation can also be improved.

Based on this improvement opportunities, we developed AxE (Axiom Editor), an axiom editor integrated to ODEd, which is presented in the following section.

4. AxE: ODEd's Axiom Editor

In order to better support axiom definition and ontology evaluation, the ontology must be formalized using an ontology specification language. In 2002, when we started developing AxE, the W3C Ontology Web Language (OWL) was still a working draft and was strongly based on DAML+OIL [15]. Hence, the latter was chosen as the ontology specification language for AxE. DAML+OIL is a semantic markup language for Web resources. It builds on earlier W3C standards such as RDF and RDF Schema, and extends these languages with richer modelling

primitives. DAML+OIL provides modelling primitives commonly found in frame-based languages. However, DAML+OIL did not have enough expression power to represent axioms in a general way. For example, an axiom such as the axiom A2 presented in section 3 cannot be written in DAML+ OIL. Thus, another language was needed to complement it, and we chose KIF [16] for that purpose.

KIF (Knowledge Interchange Format) is a formal language for the interchange of knowledge among disparate computer programs [16]. It is not intended for internal representation of knowledge, therefore the ontology and the axioms are stored as objects in the system and converted to DAML+OIL and KIF for the interchange with the inference engine (this is further detailed later). KIF has declarative semantics and provides for the expression of arbitrary sentences in predicate calculus, thus complementing DAML+OIL for our purposes.

Before discussing how axioms are treated in AxE, we need to first present how ontologies are internally represented in ODEd. Figure 3 shows partially the current ODEd's meta-ontology model. This meta-ontology model was proposed as part of the improvement efforts done in ODEd, in which we aimed to make it compatible with UML meta-model specification. In other words, the current ODEd's meta-ontology model can be seen as an UML light weight extension. With this approach, we uniformly treat all models in ODE (including ontologies), allowing reuse and facilitating integration.

The ontology purpose and its intended uses are identified through *competency questions*. An *ontology* is represented by a set of *ontology diagrams*, which contains *concepts*. Concepts are related through *relationships* that can be *associations* or *generalizations*. The latter denotes subsumption relationships, while the former can be *relations* or *whole-part* relations, which in turn can be of two types: *aggregation* or *composition*. Concepts and relations may have *properties* and, in an *association*, concepts have *roles* and *cardinalities* (*association ends*).

Making the ontology meta-model compatible with the UML meta-model imposes a bit more of complexity to the design of the system. The relationship between the *Ontology* and its *Diagrams* is represented by the association between *Model* and *Diagram*. Similarly, *Ontology* and *CompetencyQuestion* are associated at the superclass level: *Model* and *ModelElement*. The same thing happens to all the other classes in the ontology meta-model (which are color-filled for emphasys): their relations with *Ontology*, *OntologyDiagram* and among themselves are modeled at the UML meta-model.

Although the axioms are written in KIF, ODEd also converts them to an internal model, following the structure presented in Figure 4. An *axiom* is modeled as a tree, having an *operator* as root, which is the central operator of the axiom. Operators have *symbols* as arguments. A symbol can be an *operator* or an *operand*. *Operands* can be *variables*, *concepts*, *instances of concepts* or *undefined concepts*. *Operators* can be *logical connectives*, *relations* or *undefined relations*, which are like *relations* but do not specify which *concepts* it associates (they are used to create axiom templates).

Axiom Templates are a special kind of axiom that defines general association axioms, such as those treated in the first version of ODEd (symmetry, transitivity, and so on). Axiom templates can use only undefined relations and connectives as operators and undefined concepts and variables as operands. In the case of symmetry, for example, it can be defined by the following axiom template: `(C1 R C2) <=> (C2 R C1)`, where `C1` and `C2` are undefined concepts and `R` is an undefined relation. When this axiom template is linked to a specific association in the ontology, a new axiom is automatically generated, substituting the undefined relation `R` by the association, and the undefined concepts `C1` and `C2` by the corresponding concepts of the association.

To aid writing axioms, AxE provides an easy-to-use graphical interface, shown in Figure 5. On the left side of the window, we can see the concepts, relations, instances and variables that can be used for defining an axiom. In the bottom part, there are buttons for easy access to common KIF and DAML+OIL operators. In the top-right, the description field can be used to supply the axiom with a description in natural language, while the axiom formalization field is where the KIF sentence is formed. Finally, the infix/prefix radio buttons determine in which notation the axiom is being written.

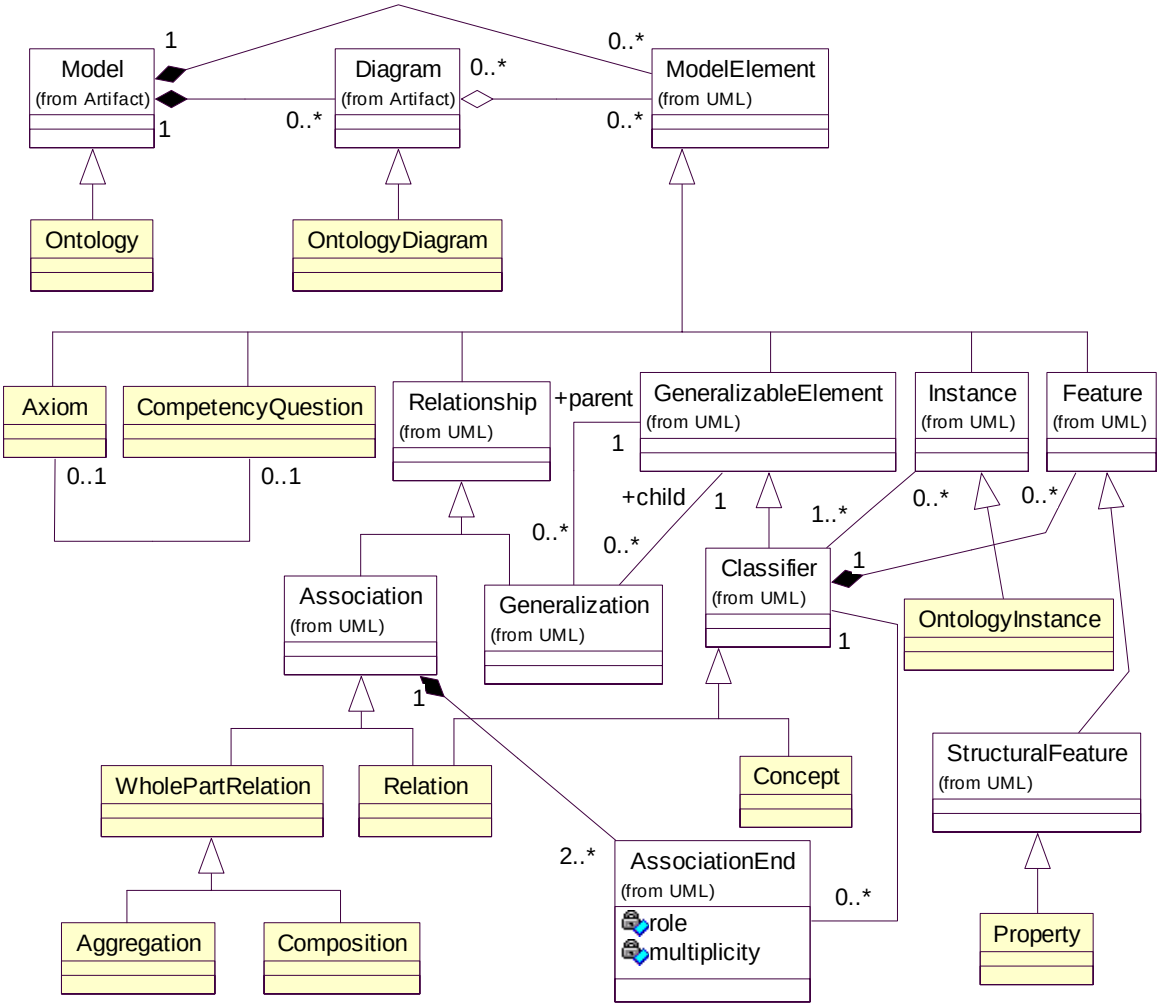

Figure 3: ODEd's meta-ontology model

Once the ontology is formally axiomatized, it is possible to support ontology evaluation by integrating an inference engine into the axiom editor. For this purpose, JTP [17] was chosen and integrated to AxE, because it is able to parse DAML+OIL ontologies and KIF statements, and also because it is an open source project written in Java, thus, being seamlessly integrated to AxE.

Figure 6 shows AxE's query interface. On the top text area, the domain engineer writes the query, which consists of an axiom with open variables. The axiom editor (Figure 5) can be used to formulate the query. JTP's response — the concepts that fit in the open variables — is returned in the bottom text area.

By formulating queries the domain engineer can check an ontology. However, there is a more appropriate way to do this: by formalizing the competency questions. In this case, first, each competency question must be formalized. That is, we have to define an axiom representing the competency question. This is the reason why competency questions are linked to axioms in Figure 3. Those axioms must have at least one open variable as an operand, and are said to be queries. Next, these queries can be submitted to the inference engine. If all the queries return satisfactory results, then we can say that the ontology is answering all the competency questions, hence fulfilling its requirements.

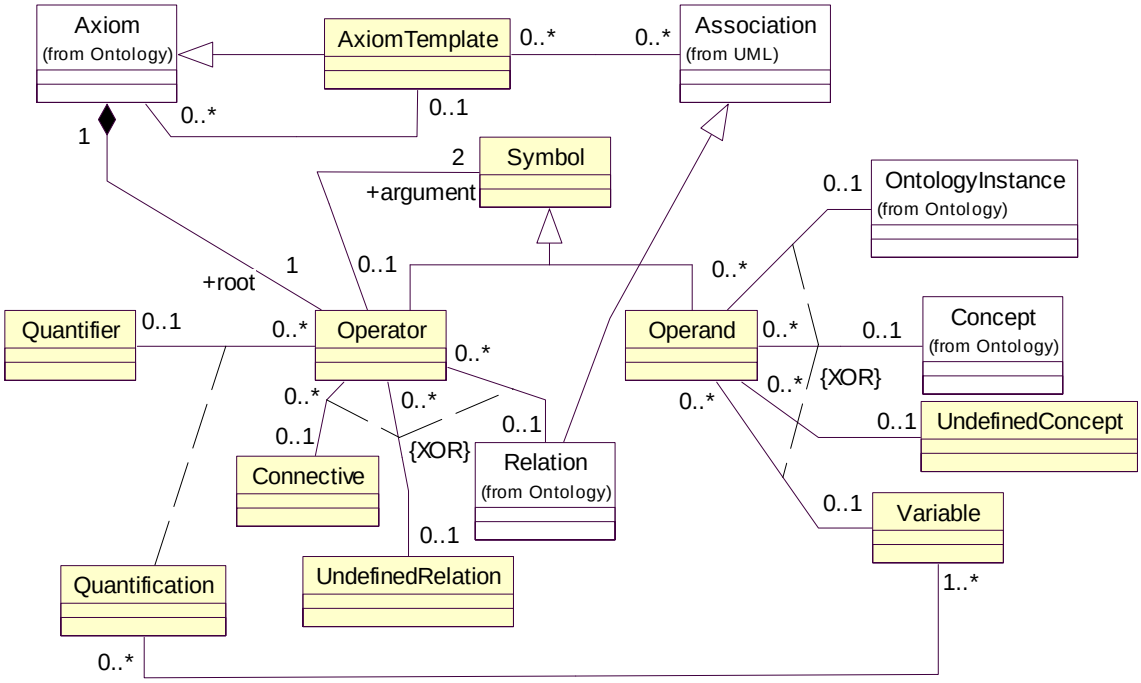

Figure 4: Axiom representation in AxE

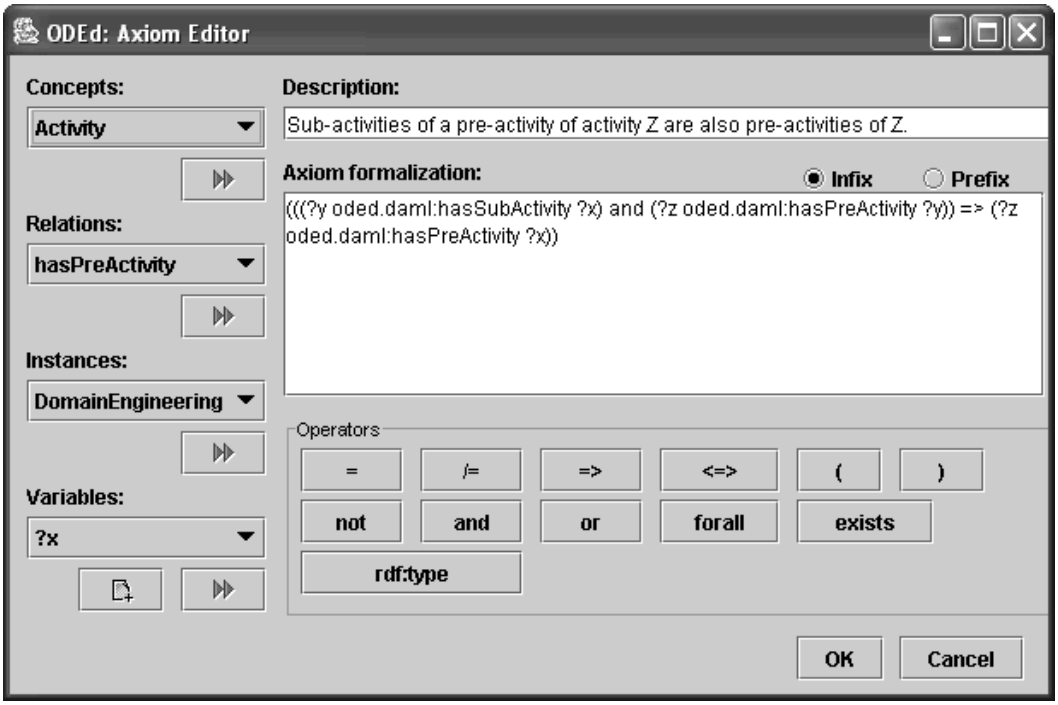

Figure 5: Writing axioms in AxE

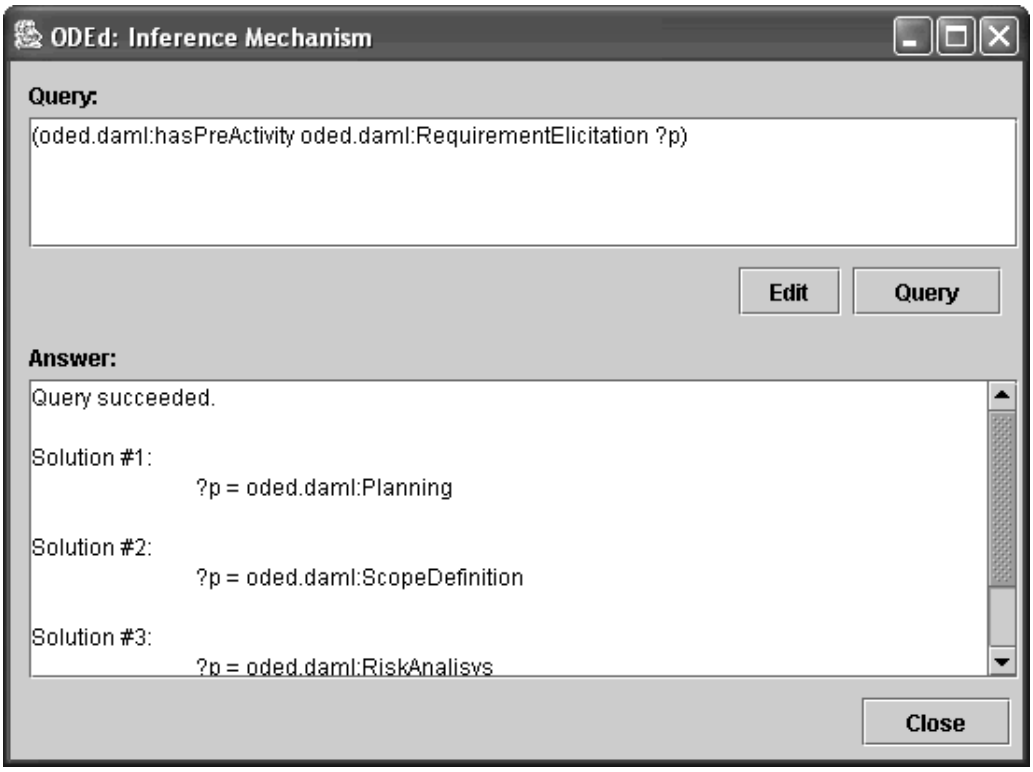

Figure 6: Querying an ontology in AxE

As told before, one of the main goals of the ontology-based domain engineering approach implemented in ODEd is to reuse the domain knowledge throughout the software development process, in activities such as requirement analysis, design and implementation. ODEd generates two artifacts that can be reused: one of them is the ontology itself, which can be the basis for requirement analysis and for domain investigation, as discussed in section 3; the other one is a framework of Java classes derived from the ontology.

As discussed in section 3, ODEd supports the derivation of Java frameworks from ontologies. This derivation maps concepts to classes, properties to attributes, and relations to bi-directional associations (and corresponding methods). Since an inference engine has been integrated to ODEd to provide the ability to work with axioms, it was also integrated to the generated frameworks with the same purpose, allowing the developers to reuse all of the domain knowledge modeled during domain engineering. This is done by making a call to the inference engine whenever a method that corresponds to a relation in the ontology is called, to check whether this relation participates in any axioms registered within the ontology. If it does, we should translate the axioms into method calls that will change the result of the method that was called in the first place.

Axiom A2 (Section 2), for example, is written in infix KIF as follows:: `((?A hasPreActivity ?B) and (?B hasSubActivity ?B1)) => (?A hasPreActivity ?B1)`. When the activity ontology is derived to Java classes, the class Activity, representing the homonymous concept from the ontology, would have a `getPreActivities()` method that returns a set of pre-activities from an activity *a*. When this method is applied in an object `X`, it will check the inference engine for axioms related to the `hasPreActivity` relation and will find the one above. Then, it will convert the axiom to method calls. First, it will convert the first part of the axiom and obtain all pre-activities of `X` by calling `X.getPreActivities()` (with checks to prevent eternal loops). Then, for each object returned from that, it will apply the second part of the axiom and call `getSubActivities()`. The objects returned from those calls are added to the actual set of pre-activities of `X`, which is finally returned to the user.

Although this feature has been developed, it requires thorough analysis and testing to verify if integrating an inference engine in domain objects that are to be used by software developers is really a good idea. Things like performance, increased complexity and inference engine lock-in should be considered beforehand. As we didn't have time in our research, this should be considered future work.

5. Related Work

There are many ontology editors described in the literature, such as OntoEdit, OILEd and Protége-2000. *OntoEdit* [18] pursues an approach where graphical means exploited for concepts and relations modeling scale up to axiom specifications in RDF(S), XML, DAML+OIL, F-Logic and SQL Scheme. Regarding ontology evaluation, it provides axiom editing and it has an inference engine attached to support ontology validation, consistency checking and even debugging.

OILEd [19] supports the construction of ontologies in OIL. The editor allows the definition of concepts and relations and also supports the definition of some pre-defined axioms. OILEd has reasoning services that supports ontologies construction, integration and verification. Ontologies are formalized using Description Logics, and OILEd has a DL Classifier (SHF and SHQ) integrated to provide subsumption consistency checking.

Protégé-2000 [20] aims to support knowledge acquisition, and to reach interoperability with other knowledge representation systems. It generates knowledge acquisition forms automatically based on the types of the slots and restrictions on their values, allowing ontology instantiation. It also allows axiom creation using Protégé Axiom Language (PAL), which is very similar to KIF. There is also a PAL Constraints Verifier.

As we can see, ODEd is in line with most ontology editors by using standards for ontology formalization, providing axiom editing and integrating an inference engine. Some editors cited above do have a few functionalities that are lacking in ODEd. However, ODEd exceeds the other editors when applied to domain engineering as part of the software development process. It has some features that are specific for this purpose, such as the generation of Java frameworks integrated to the inference engine, allowing carrying domain knowledge to further stages of the software process.

In fact, all editors previously mentioned were developed to support ontology design in the context of the Semantic Web. None of them was developed aiming to support evolving a Software Development Environment (SDE) to a Domain-Oriented SDE (DOSDE). An exception is the domain theory editor of the meta-environment TABA [5]. TABA, as a meta-environment, generates DOSDEs using the ontologies defined in the domain theory editor. This editor allows defining concepts, relations and properties of an ontology. This editor also allows entering axioms in pre-defined Prolog files. When the meta-environment generates a DOSDE, each concept becomes a class in the DOSDE. But this editor is not graphical, does not support axiom edition nor ontology evaluation, and its class generation is too simple. ODEd, as discussed in this paper, goes a step ahead in supporting ontology building and mapping to object frameworks in the context of a SDE.

6. Conclusions

In this paper, we presented an evolution of ODEd, an ontology editor developed in the context of a software development environment, focusing on the support to ontology axiomatization and evaluation offered by its axiom editor AxE.

By integrating AxE to ODEd, we provided the ability to define axioms in a general way, allowing expressing more complex axioms than in the first version of ODEd. The use of axiom templates, in turn, preserved the original approach adopted, since we can still pre-define some classes of association axioms, such as symmetry and transitivity. The languages chosen for representing axioms in AxE (DAML+OIL and KIF) have great expression power. Integrating an inference engine to AxE, we also improved ODEd's capability of evaluating ontologies. This

integration was also extended to the generated object framework, automatically carrying the domain knowledge to future stages of the software process.

However, there is a lot of room for improvement in ODEd. At this moment, the following can be noticed:

- Only binary relations are supported;
- There is no support for distributed/collaborative ontology development;
- Java framework derivation research, as explained at the end of section 4;
- There are several improvements to be implemented on its user interface, especially concerning diagram drawing;
- We are studying the possibility of changing the ontology representation language to a more powerful one, such as OWL DL. But it depends on the availability of ontology inference engines. As pointed by Franconi [21], it is not possible to build a complete inference engine for OWL Full yet. However, some existing description logics systems can be used as inference engines for OWL Lite and OWL DL.

7. Acknowledgments

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

8. References

[1] C. Brewster, K. O'Hara. "Knowledge Representation with Ontologies: The Present and Future", IEEE Intelligent Systems, pp. 72-73, January/February 2004.

[2] R.A. Falbo, G. Guizzardi, K.C. Duarte. "An Ontological Approach to Domain Engineering". Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, SEKE'2002, pp. 351- 358, Ischia, Italy, 2002.

[3] R.A. Falbo, F.B. Ruy, J. Pezzin, R. Dal Moro, "Ontologias e Ambientes de Desenvolvimento de Software Semânticos", 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering, JIISIC'2004, Madrid, Spain, November 2004.

[4] H. Holz, G. Melnik, "Research on Learning Software Organizations - Past, Present, and Future", Advances in Learning Software Organizations (Proceedings of the 6th International Workshop on Learning Software Organizations - LSO'2004), Melnik G. and Holz, H. (Eds.): LNCS 3096, pp. 1-6, 2004.

[5] 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, pp. 145-161, 2004.

[6] R.A. Falbo, C.S. Menezes, A.R.C. Rocha. "A Systematic Approach for Building Ontologies". Proceedings of the 6th Ibero-American Conference on Artificial Intelligence, Lisbon, Portugal, Lecture Notes in Computer Science, vol. 1484, 1998.

[7] O. Lassila, F. Van Harmelen, I. Horrocksm, J. Hendler, D.L. Mcguinness. "The Semantic Web and its Languages". IEEE Intelligent Systems, pp. 67-73, November/December 2000.

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

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

[10] R.A. Falbo, “Experiences in Using a Method for Building Domain Ontologies”, Proceedings of the International Conference on Software Engineering and Knowledge Engineering – SEKE’2004, Ontology in Action Workshop, 2004.

[11] N. Guarino. “Understanding, building and using ontologies”. Int. Journal Human-Computer Studies, 46(2/3), February / March 1997.

[12] M. Gruninger, J. Lee, Ontology Applications and Design, Communications of the ACM, vol. 45, no. 2, p. 39-41, February 2002.

[13] N. Guarino. “Formal Ontology and Information Systems”. In N. Guarino (Ed.), *Formal Ontologies in Information Systems*, IOS Press, 1998.

[14] M. Grüninger, M.S., Fox. “Methodology for the Design and Evaluation of Ontologies”. Technical Report, University of Toronto, 1995.

[15] D. Connolly, F. van Harmelen, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein. “DAML+OIL (March 2001) Reference Description”, December, 2001.

[16] M.E. Genesreth, R.E. Fikes. “Knowledge Interchange Format, Version 3.0 Reference Manual”. Technical Report Logic-921, Computer Science Department, Stanford University, 1992.

[17] R. Fikes, J. Jenkins, G. Frank. “JTP: A System Architecture and Component Library for Hybrid Reasoning.: Proceedings of the Seventh World Multiconference on Systemics, Cybernetics, and Informatics. Orlando, Florida, USA. July 27 - 30, 2003.

[18] S. Staab, A. Maedche. “Ontology Engineering beyond the Modeling of Concepts and Relations”. 14th European Conference on Artificial Intelligence, Workshop on Applications of Ontologies and Problem-Solving Methods, 2000.

[19] S. Bechhofer, I. Horrocks, C. Goble, R. Stevens. “OilEd: a Reason-able Ontology Editor for the Semantic Web”. Working Notes of the 14th International Workshop on Description Logics (DL-2001), pp.1-9, USA, August 2001.

[20] N.F. Noy, M. Sintek, S. Decker, M. Crubézy, R.W. Fergerson, M.A. Musen. “Creating Semantic Web Contents with Protégé-2000”, IEEE Intelligent Systems, March/April 2001.

[21] E. Franconi. “Using Ontologies”, IEEE Intelligent Systems, pp.76-77, January/February 2004.