Supporting Ontology Development with ODEd

Paula Gomes Mian, Ricardo de Almeida Falbo
Federal University of Espírito Santo
Fernando Ferrari Avenue, 29060-900
Vitória – ES – Brazil
+55 (27) 3335-2167
{pgmian, falbo}@inf.ufes.br

ABSTRACT
Ontologies are becoming an important mechanism to build information systems. However, ontology construction is not a simple task. So, it is necessary to provide tools that support ontology development. This paper presents ODEd, an ontology editor that supports the definition of concepts and relations using graphic representations, besides promoting automatic generation of some classes of axioms and derivation of object frameworks from ontologies.

Keywords
Ontologies, Domain Engineering, Ontology Editors.

1. INTRODUCTION
In contexts where knowledge has to be modeled, structured, and interlinked, ontologies can help formalize the knowledge shared by a group of agents [1].

However, 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 constrains, described as axioms [2]. Therefore, tools for supporting ontology development are necessary. These tools must support the definition and modification of concepts, relations, properties, axioms, and constraints, and must enable the inspection, browsing, and codifying of the resulting ontologies [3].

In this paper, we present *ODEd*, an ontology editor that supports the definition of concepts and relations, using graphic representations, and promotes automatic generation of some classes of axioms. Also, ODEd supports the derivation of object-oriented frameworks from ontologies. In section 2 we briefly discuss some aspects of ontologies in software development. Section 3 discusses the ontology development process that underlies ODEd functionalities. Section 4 presents an overview of ODEd architecture. Sections 5 and 6 discuss an example of a ontology generated in ODEd. Section 7 presents how ODEd supports domain investigation, allowing ontology browsing. In section 8 we discuss related works. Finally, in section 9 we report our conclusion and future work.

2. ONTOLOGIES
People, organizations and software systems must communicate between and among themselves. However, due to different needs and backgrounds contexts, there can be widely different viewpoints and assumptions regarding the same subject matter. The way to solve this problem is to minimize conceptual and terminological confusion and come to a shared understanding of the domain of interest [4].

However, it is impossible to represent the real world, or even a part of it, with all its details. 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 consists of developing a conceptualization: a set of informal rules that constrain the structure of a piece of reality, which an agent uses to isolate and organize relevant concepts and relations [5].

According to Guarino [6], "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". An ontology can take a variety of forms, but necessarily it should include a vocabulary of terms, and some specification of their meaning. This includes definitions and an indication of how concepts are inter-related which collectively impose a structure on the domain and constrain the possible interpretations of terms [7]. Thus, an ontology consists of concepts and relations, and their definitions, properties and constrains expressed as axioms [2].

Jasper et al. [8] classified applications of ontologies in four main categories, emphasizing that an application may integrate more than one of these categories:

- Neutral Authoring: an ontology is developed in a single language and it is translated into different formats and used in multiple target applications.
- Ontology as Specification: an ontology of a given domain is created and it provides a vocabulary for specifying requirements for one or more target applications. In fact, the ontology is used as a basis for software specification and development, allowing knowledge reuse.
- Common Access to Information: an ontology is used to enable multiple target applications (or humans) to have access to heterogeneous sources of information that are expressed using diverse vocabulary or inaccessible format.
- Ontology-based Search: an ontology is used for searching an information repository for desired resources, improving precision and reducing the overall amount of time spent in searching.

Analyzing these scenarios, we can notice that working with ontologies has several advantages. One of the main benefits of the use of ontologies in software development is to reuse domain specifications in the requirement specification phase. In traditional Software Engineering, for each new application to be built, a new conceptualization is developed. This reflects on how requirements are elicited: for each new application, an elicitation phase is accomplished almost always from scratch, focusing on all

particularities of the system at hand. This approach is extremely expensive since requirement elicitation is a very time-consuming activity. Experts are scarce and costly resources, and they are essential to this activity. So they should be better used. Therefore, it is important to share and reuse the knowledge captured [9].

In an ontology-based approach, requirement elicitation and modeling can be accomplished in two stages. First, the general domain knowledge should 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, diluting the costs of the first stage and allowing knowledge sharing and reuse [9]. In this context, ontologies can act as both a domain model and a component in a repository of reusable artifacts. Also, it can be used for structuring this repository.

Since the current leading paradigm in Software Engineering is the object technology, to put ontologies in practice for developing information systems, it is worthwhile to code the resulting ontologies using objects. We believe that coding ontologies in object frameworks may lead to reuse in several levels of software development: from analysis to project and implementation.

To support ontology development and its coding in Java, we developed ODEd, an ontology editor. Before presenting ODEd, however, we need to describe the ontology development process that the tool automates.

3. AN ONTOLOGY DEVELOPMENT PROCESS

Falbo et al. [2] have proposed an ontology development process, that encompasses the following activities, as shown in Figure 1:

- Purpose identification and requirement specification: it concerns to clearly identify the ontology purpose and its intended uses, that is, the competence of the ontology. To do that, competency questions are used.
- Ontology capture: the goal is to capture the domain conceptualization based on the ontology competence. The relevant concepts and relations should be identified and organized. A model using a graphical language, with a dictionary of terms, should be used to facilitate the communication with domain experts.
- Ontology formalization: aims to explicitly represent the conceptualization captured in a formal language.
- Integration of existing ontologies: during the capture and/or formalization steps, it could be necessary to integrate the current ontology with existing ones, in order to seize previously established conceptualizations.
- Ontology evaluation: the ontology must be evaluated to check whether it satisfies the specification requirements. It should also be evaluated in relation to the ontology competence and some design quality criteria, such those proposed by Gruber [10].
- Documentation: all the ontology development must be documented, including purposes, requirements and motivating scenarios, textual descriptions of the conceptualization, the formal ontology and the adopted design criteria.

The dotted lines indicate that there is a constant interaction, albeit weaker, between the associated steps. The filled lines show the main workflow in the ontology building process. The box involving the capture and formalization steps enhances the strong interaction, and consequently iteration, between them.

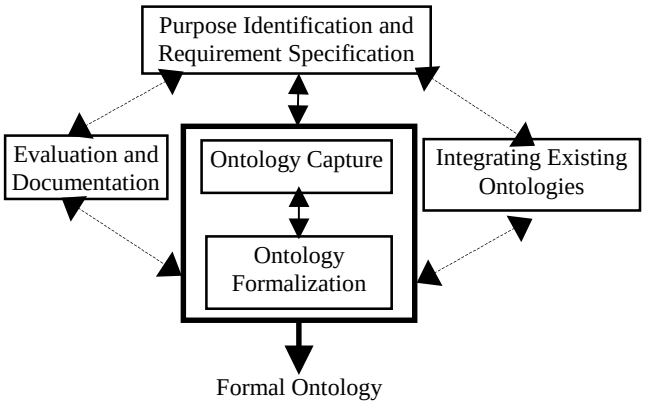


Figure 1. Steps in the ontology development process.

ODEd aims to support this process. ODEd supports ontology capture by supporting the definition of concepts and relations using graphical representations, and it promotes automatic generation of some classes of axioms. Also, ODEd supports codifying the resulting ontologies in Java. To do that, it works based on the approach defined in [9], that defines a set of directives, design patterns and transformation rules for deriving object frameworks from ontologies. The directives are used to guide the mapping from the epistemological structures of the domain ontology (concepts, relations, properties and roles) to their counterparts in the object-oriented paradigm (classes, associations, attributes and roles). The design patterns and transformation rules are applied in axioms mapping. The application of these guidelines is supported by a Java Set framework that implements the mathematical type Set [9]. In this phase, the following activities should be performed:

- Set-based ontology axiomatization: to derive objects from domain ontologies, it is worthwhile to adopt a formalism that lies at an intermediate abstraction level between first-order logics and objects. For this purpose, a hybrid approach based on pure first-order logic, relational theory and, predominantly, set theory was proposed in [9]. So, the first step is to perform the complete axiomatization of the domain ontology using this set-based formalism.
- Class identification: starting from the sets formally defined, a preliminary list of the classes of the object-oriented model can be established;
- Epistemological structure translation: since the classes are defined, relations among concepts and epistemological axioms should be translated to the corresponding object-oriented structures, producing an initial class diagram;
- Consolidation and ontological axioms translation: the class diagram derived in the step above should be refined to consider consolidation and ontological axioms.

4. ODEd'S ARCHITECTURE

In order to support the ontology development process described above, ODEd implements a three-layered architecture shown in Figure 2. The ontologies are developed through the *presentation* layer and they are described according to a model defined in the *domain* layer. The *data management* layer is responsible for the storage of the ontologies designed.

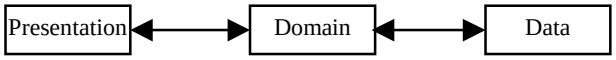
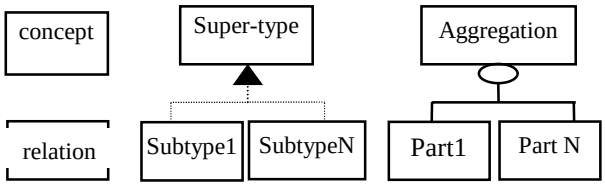


Figure 2. ODEd Architecture.

This architecture uses is a project philosophy that suggests that the central classes, in the domain layer, are not aware of how the ontologies are presented to the user or stored by the system. The portion of the system that handles the graphic representation of the ontologies (presentation layer) is independent from the rest of the architecture and it communicates with the domain layer. The data layer provides the basic infrastructure for the storage and the recovery of objects in the system. Its purpose is to isolate the impacts of the technology of data management on the editor's architecture.

The presentation layer supports the ontology capture. In this step, the use of a graphical representation is essential in order to facilitate the communication between domain engineers and experts. In ontology building, such representation is basically a language representing a meta-ontology. So, this language must own basic primitives to represent a domain conceptualization and, in its simplest form, it should have notations to represent only concepts and relations [2]. Falbo et al. [2] proposed a Graphical Language for Expressing Ontologies (LINGO). LINGO has the basic primitives to represent a domain conceptualization, i.e., in its simplest form; its notations represent only concepts and relations. Nevertheless, some types of relations have a strong semantics and, indeed, hide a generic ontology. In such cases, specialized notations have been proposed. This is the striking feature of LINGO and what makes it different from other graphical representations: any notation beyond the basic notations for concepts and relations aims to incorporate a theory [2]. This way, axioms can be automatically generated. These axioms concern simply the structure of the concepts and are said epistemological axioms (EA). Figure 3 shows the main notations of LINGO and some of the axioms imposed by the whole-part relation. These axioms form the core of the mereological theory as presented in [11]. Irreflexivity (EA1), anti-symmetry (EA3) and transitivity (EA4) axioms denote sufficient and necessary properties for all kinds of whole-part relations. The remaining axioms complete the theory by defining suitable ontological distinctions.

ODEd uses LINGO as a graphic language to describe ontologies, allowing the automatic generation of the LINGO's notations built-in axioms. Upon using these notations during ontology capture, an ontology developer is also defining the group of axioms that they represent. ODEd uses this feature to automatically generate these types of axioms. In this way, ODEd embeds a powerful mechanism of theories inclusion. Each relation type specifying a generic theory has its own notation and whenever it is used, generic ontologies are integrated automatically [2]. Besides the epistemological axioms, other axioms can be used to represent knowledge. These axioms can be of two types: consolidation axioms (CA) and ontological axioms (OA) [2]. The former aims to impose constraints that must be satisfied for a relation to be consistently established. The latter intends to represent declarative knowledge that is able to derive knowledge from the factual knowledge represented in the ontology, describing domain signification constraints.



(EA1) $\forall x \ \neg partOf(x,x)$

(EA2) $\forall x,y \ partOf(y,x) \leftrightarrow wholeOf(x,y)$

(EA3) $\forall x,y \ partOf(y,x) \rightarrow \neg \ partOf(x,y)$

(EA4) $\forall x,y,z \ partOf(z,y) \wedge partOf(y,x) \rightarrow partOf(z,x)$

(EA5) $\forall x,y \ disjoint(x,y) \rightarrow \neg\exists z \ partOf(z,x) \wedge partOf(z,y)$

(EA6) $\forall x \ atomic(x) \rightarrow \neg\exists y \ partOf(y,x)$

Figure 3. LINGO's main notations and some axioms.

UML has also been used as an ontology modeling language [12]. Therefore, ODEd's presentation layer also supports ontology capture using UML. However, it is necessary to emphasize that there are some problems in using UML as an ontology modeling language. First, an important criterion to evaluate ontology design quality is minimum ontological commitments [10]. Based on this principle, an ontology modeling language must embody only notations that are necessary to express ontologies. This is not the case of UML and majority graphical languages available. Second, since an ontology intends to be a formal model of a domain, it is important that the language used to describe it has formal semantics. Again, this is not the case of the majority graphical languages available, including UML [13]. However, we cannot ignore that UML is a standard and its use is widely diffused. Moreover, there are efforts to define UML semantics, such as pUML [14]. Based on that, in ODEd, we defined a subset of UML that plays the same role of LINGO's notation.

As shown in Figure 4, stereotyped classes (<<*Concept*>>) represent concepts. Relations are defined as labeled associations, and properties are represented as attributes. Relations that contain properties or relation of arity bigger than two are represented as stereotyped associative classes (<<*Relation*>>). Super-type and whole-part relations among concepts are represented as generalization/specialization and aggregation relationships, respectively. Here, the same approach of LINGO is adopted: specific notations, such as aggregation, composition and specialization, should incorporate well-defined theories. Thus, the semantic meaning of UML modeling elements can be captured precisely. For instance, the epistemological axioms that compose the whole-part theory presented in Figure 3 are also automatically generated by ODEd when the aggregation notation of UML is used.

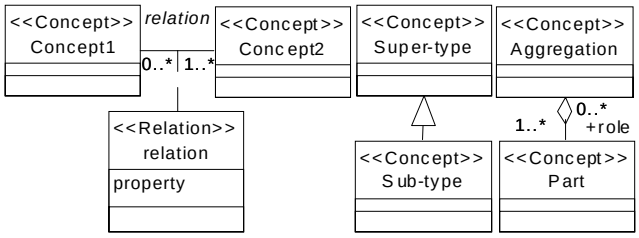


Figure 4. A Subset of UML to represent ontologies.

ODEd allows configuring what graphical representation use to develop the ontologies. The ontology can be captured in LINGO or UML. The objects that represent the ontology are created in the domain layer independently of the graphical representation used and the presentation layer may present them to the user in two distinct forms. In spite of different graphic representations, the ontology domain model is the same. The presentation layer provides an interface to create the objects of the domain layer and improves modularity by encapsulating the way the objects are constructed and represented.

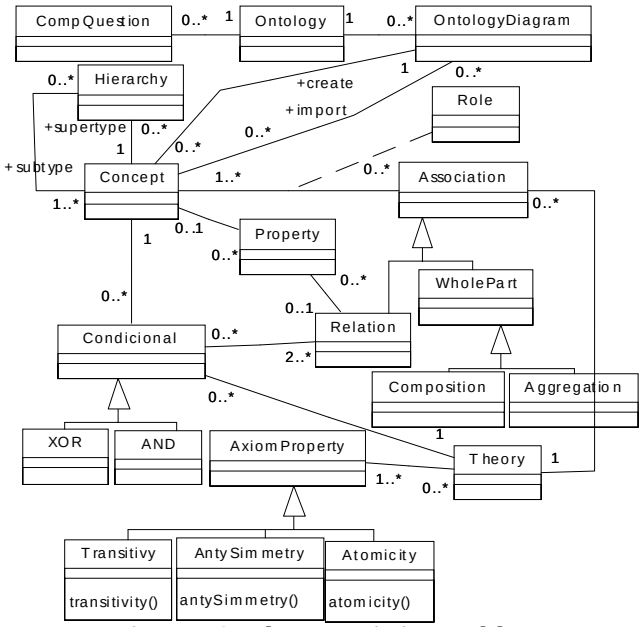


Figure 5. Ontology Description Model.

The ontologies are described in the domain layer by an independent ontology description model presented in Figure 5. The *ontology* purpose and its intended uses are identified through *competency questions*. It is represented by a *diagram,* which contains *concepts created* in or *imported* to the ontology. Concepts are related by *relations*, *hierarchy, whole-part* and *conditional* relationships. Whole-part relations are classified in two kinds (*aggregation* and *composition*) as well as conditional relations (*XOR* and *AND*). Concepts and relations may have *properties* and the *associations* between concepts have *roles* and cardinalities. These associations may have *theories* associated to them. Theories are composed by *axioms properties* such as *atomicity*, *anti-symmetry* and *transitivity*. Using the presentation layer, the user creates objects of this model and the objects generated do not depend on the graphical language used.

LINGO's and UML's notations have axioms built in and ODEd is capable of generating these axioms. Beyond generating pre-defined theories, the tool also allows the user to compose his/hers own theories and apply then to relations in the ontology. This approach to represent theories is similar to that presented in [15]. The core idea is to use a categorization that organizes axioms and that provides a compact, intuitively accessible representation. Axioms are classified according to axiom properties, such as *anti-reflexivity, anti-symmetry, atomicity, disjointed, exclusivity, reflexivity, symmetry* and *transitivity*. As shown in Figure 5, these properties are used to compose theories associated to relations on the ontology.

Each axiom property has a mini design pattern associated (some of these patterns are presented in Figure 5). These patterns are captured by classes capable to check if the axiom properties represented by the pattern hold. `For instance, the anti_symmetry() method of the anti-symmetry pattern is responsible for checking if a relation is anti-symmetric. It executes the method` *`relation`* `(representing an relation among concepts) of an object` *`obj`* `(representing an instance of a concept)`. If *obj* is not returned by *`relation`* then the anti-symmetry property is truth and *relation* is anti-`symmetric`.

Besides creating the design patterns to represent the properties axioms, it is necessary to define how they can to compose the theories. To do so, another design pattern was created.

There are some axioms, such those that represent theories, whose purpose is to describe preconditions that must be satisfied, or properties that must hold, so that a relation can be established between two concepts.

Generally speaking, this type of axiom has the following format: *∀x:X, y:Y relation(x,y) → (preCondition1) ∧ (preCondition 2) ∧ ... ∧ (preConditionN).* This generic format was mapped to the *PreCondition Pattern* [9] that guarantees the evaluation of each one of precondition before a relation can be established. This pattern uses the *Template Method* pattern [16]. In this case, the *template method* is the method `setRelation()` and the *hook methods* are those responsible for evaluating the fulfillment of the preconditions.

To support theories composition, the PreCondition Pattern defined in [9] was modified. The hook methods are now axiom property patterns responsible for evaluating the fulfillment of the preconditions of the corresponding relation theory. The generic format of the new PreCondition Pattern is: *∀x:X, y:Y relation(x,y) → (axiomProperty1) ∧ (axiomProperty2) ∧ ... ∧ (axiomPropertyN)*.

Object frameworks generated by ODEd incorporate the new PreCondition Pattern to compose and verify relation theories. If a relation possesses a theory, its pre-conditions are tested before including or removing objects.

5. DEVELOPING AN ONTOLOGY OF SOFTWARE QUALITY USING ODEd

To present an example of the ontology development in ODEd we present the *Quality Ontology* developed in [17]. Due to limitations of space, we present only part of this ontology.

The first step of the ontology development defined in the process presented in section 3 is the purpose identification and requirement specification. To support this phase, ODEd allows the user to define competency questions. The form presented in Figure 6 allows the user to create or remove competency questions of the ontology. The part ontology previously described concerns the competency questions presented in Figure 6.

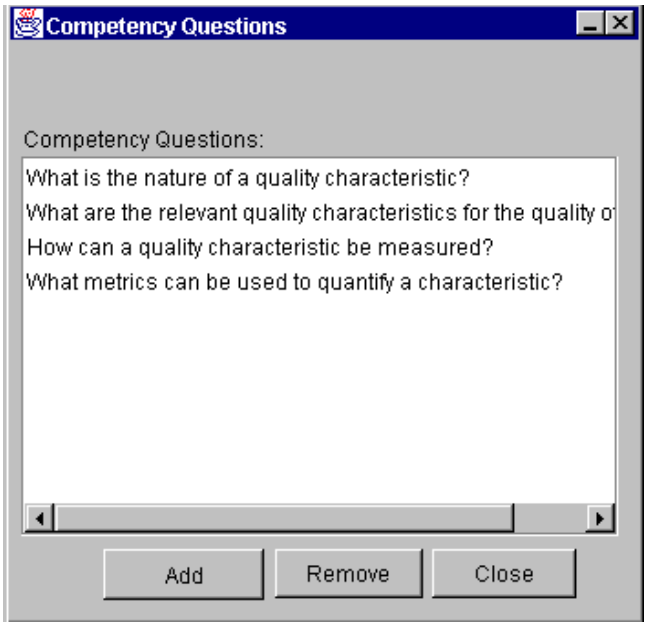


Figure 6. Competency questions of the Quality Ontology.

Once the competency questions are defined, it is possible to start the ontology capture. To support this phase ODEd supports the graphic representation of the ontologies concepts and relations using LINGO and UML, as discussed in section 4.

Figure 7 shows part of the Quality Ontology in LINGO. In the quality ontology, a software *quality characteristic* can be classified according to two criteria. The first one says if a quality characteristic can be directly measured or not. A *non-mensurable characteristic* must be decomposed into sub-characteristics (represented by the roles *super* and *sub characteristic*) to be computed by the aggregation of their sub-characteristic measures. A *mensurable characteristic* can be directly *quantified* applying some *metric*. The second classification enforces that *product characteristics* should only be used to evaluate software artifacts and *process characteristics* evaluate software processes. *Artifact* is a concept from the *Software Process Ontology* [2], which were integrated with the quality ontology been presented (see section 5.1). Product characteristics can be *relevant* to several artifacts. Finally, the *valuation* relation indicates that a non-mensurable quality characteristic can be valued through other quality mensurable or not mensurable characteristics.

Since this ontology was translated to an object framework using the approach described in section 3 [9], we used it to illustrate ODEd functionalities.

Cardinalities are used in the diagram to show how many instances of a concept can participate in the relation. In Figure 7, cardinality (1,n) in the relation quantification implies that an mensurable characteristic must be valued by, at least, one metric: $(\forall a)\ (mensqc(qc) \rightarrow (\exists m)\ (quantification(qc,m))$. Cardinality (1,1) still adds that an metric evaluates only one mensurable characteristic: $(\forall m,qc1,qc2)\ (quantification(s,qc1) \wedge quantification(s,qc2) \rightarrow qc1 = qc2)$. Since cardinality (0,n) does not impose any constrain, it is not represented. Some concepts and relations have properties. In Figure 7, mensurable characteristic has the property name, shown in the tree in the left size.

Although the example presented above represents only binary relations, the formalism used is expressive enough to model relations of any arity, including reflexive relations. Likewise, conditional relations (AND and XOR tight relations) can also be represented.

In Figure 8 it is presented the quality ontology captured using UML. The same objects modeled in Figure 7 are presented here but using a different graphical notation. A stereotyped class `QualityCharacteristic`, for example, represents the *QualityCharacteristic* concept and the relation *relevance* is presented a class association.

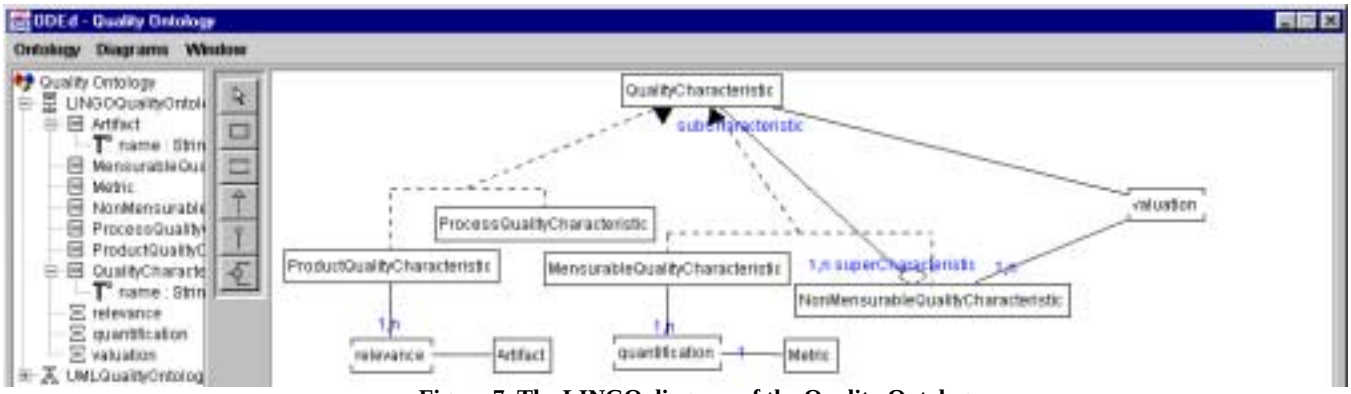


Figure 7. The LINGO diagram of the Quality Ontology.

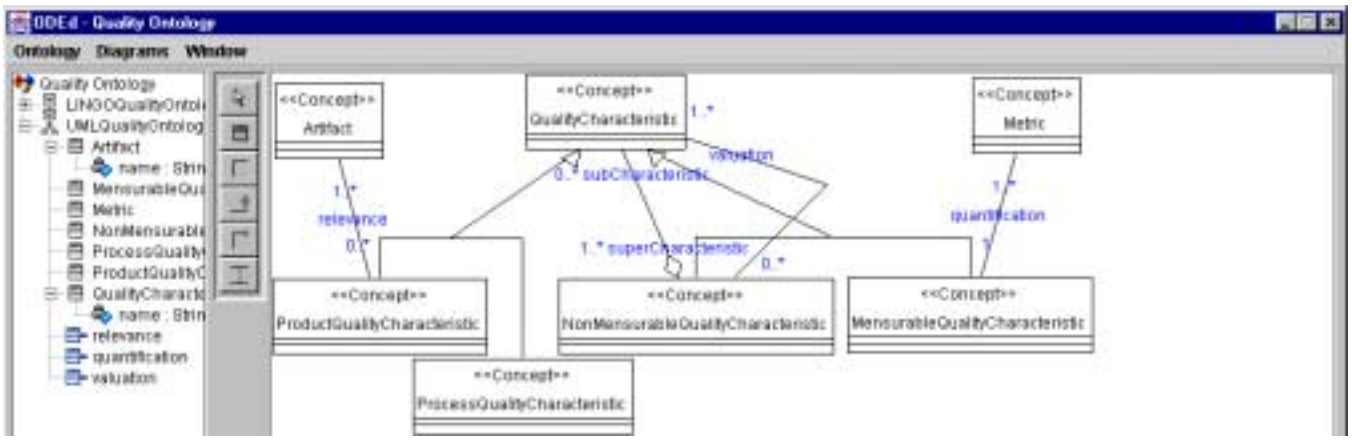


Figure 8. Representing the Quality Ontology using UML.

Table 1 presents some axioms of the activity ontology, indicating their type. Axioms (EA1) to (EA4) were derived from the super-type relation among quality characteristics. (EA5) to (EA8) are directly derived by the usage of the whole-part relation between quality characteristics. The axiom (OA1) is related to the valuation relation.

Table 1. Some axioms of the Quality Ontology.

ID	Axiom
EA1	$(\forall$ qc) (nmensqc(qc) $\rightarrow$ qchar(qc))
EA2	$(\forall$ qc) (mensqc(qc) $\rightarrow$ qchar(qc))
EA3	$(\forall$ qc) (prodqc(qc) $\rightarrow$ qchar(qc))
EA4	$(\forall$ qc) (procqc(qc) $\rightarrow$ qchar(qc))
EA5	$(\forall$ qc1, qc2) (subqc(qc1, qc2) $\rightarrow$ $\neg$ subqc(qc2 , qc1))
EA6	$(\forall$ qc) (mensqc (qc) $\leftrightarrow \neg (\exists$ qc1) (subqc(qc1, qc)))
EA7	$(\forall$ qc1, qc2, qc3) (subqc(qc1, qc2) $\wedge$ subqc(qc2, qc3) $\rightarrow$ subqc(qc1 , qc3))
EA8	$(\forall$ qc1, qc2) (disjointed(qc1, qc2) $\leftrightarrow \neg (\exists$ qc3) (subqc(qc3, qc1) $\wedge$ subcarq(qc3, qc2)))
OA1	$(\forall$ qc, qc1) (valuation(qc, qc1) $\rightarrow \neg$ valuation(qc1, qc))

The axiom (OA1) indicates that if quality characteristic qc1 is valuated by a quality characteristic qc2, then qc2 cannot be valuated by qc1. It means that the valuation relation is anti-symmetric and the anti-symmetry property should be incorporated to the relation theory. Figure 9 presents the theory associated to the valuation relation in the quality ontology. This form allows the user to associate several axioms properties to a relation. In this example, the only property that composes the valuation theory is anti-symmetry.

Initially, the axioms defined in relations' theories (such the axiom (OA1) presented above) are the only type of ontological axioms that can be represented in ODEd. It is not defined yet how to handle other ontological axioms that cannot be captured as theories. This issue will be solved in the next versions of the editor.

Figure 9. Properties of the Valuation relation.

ODEd also incorporates software agents that help the ontology designer during the ontologies development.

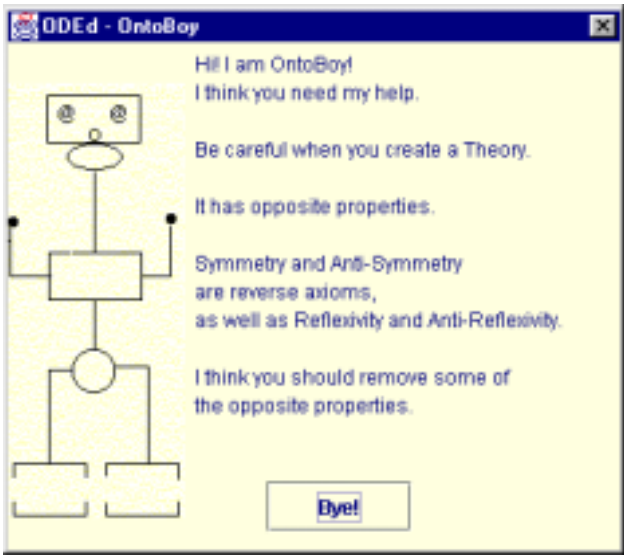


Figure 10. The agent OntoBoy.

There are some constraints that must be satisfied during ontology development. Thus, software agents were added to ODEd to alert the user about eventual modeling structural mistakes or to offer advices on how to solve them according to the user's actions.

5.1 Importing Concepts and Relations

The main purposes of the quality ontology are to promote software quality knowledge integration in a Software Engineering Environment (ODE) and to support the development of quality management tools for it [13]. Therefore, this ontology must be integrated to the software process ontology [2] used to support software process definition and automatization in ODE.

ODEd supports ontology integration in a very simple way. It is possible to import concepts from existing ontologies to the current one. If more than one concept is imported and there are relations between them, these relations are also incorporated to the ontology. Then, these concepts can be connected to the concepts of the current ontology.

For example, in Figure 7, the *Artifact* concept was imported from the software process ontology and a relation between *Artifact* and *ProductQualityCharacteristic* was created (*relevance*).

If an imported concept or relation is removed from the original ontology, it is automatically removed from the ontology it was imported. No kind of notification is sent to the knowledge engineer responsible for that ontology.

It means that if the *Artifact* is removed form the software process ontology, it will be removed from the quality ontology, as well as the *relevance* relation and the framework of both ontologies should be generated again to apply the modifications. Since no notification is sent, an application could be using an old version of the framework. Thus, a version control mechanism should be integrated to ODEd to guarantee integrity of the generated frameworks.

6. FROM DOMAIN ONTOLOGIES TO OBJECT FRAMEWORKS

As pointed in section 3, for deriving object frameworks from ontologies, Guizzardi et al. [9] defined a set of mapping directives, design patterns and transformation rules. In its current stage, ODEd considers the mapping directives and some design patterns. But, since ODEd does not support axiom definition, except those described through theories, the transformation rules are not being treated. In the next sections, we presented how the Ontology formalization is support by ODEd to derive the quality ontology framework.

6.1 Mapping Directives

According to [9], concepts and relations are naturally mapped to classes and associations in an object model, respectively. Relations between three or more concepts (n-ary relations) and relations with properties give rise to associative classes. Properties of concepts and relations are mapped to attributes of the corresponding classes.

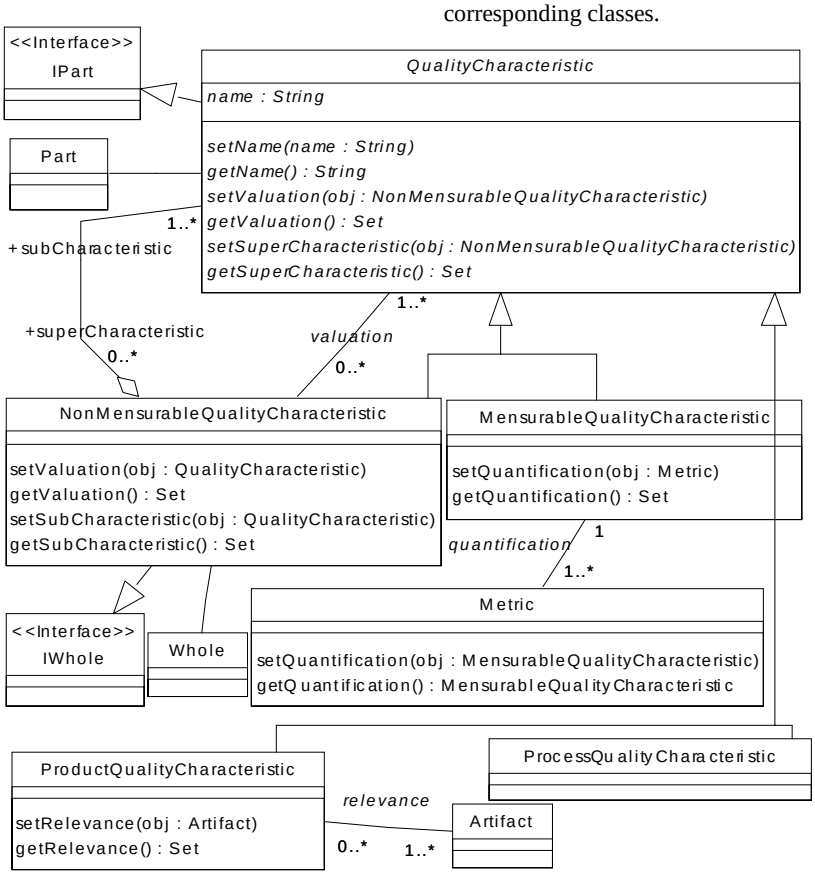


Figure 11. The Quality Framework generated by ODEd.

In the case of the quality ontology, the classes `QualityCharacteristic` and `NonMensurableCharacteristic` were derived from the corresponding concepts, as well as the associations `quantification`, `relevance`, and `valuation`, as shown in Figure 11. Properties of the concepts were mapped as attributes of the corresponding classes, as is the case of the property *name* of the concept *QualityCharacteristic*, which was mapped as the attribute `name` in the class `QualityCharacteristic`. Also, for each derived attribute, methods to get and set values were created.

Still considering the mapping of relations, there are other issues that must be discussed. First, since in an ontology relations are bi-directional, the corresponding associations must be navigable in both directions. Thus, the associations are implemented as attributes, and there are methods in both classes to return them. The returned type of the relation methods depends directly on the cardinality associated to the relation [9]. For instance, since in the scope of the *quantification* relation an mensurable characteristic may be evaluated by several metrics, `quantification()` is mapped to a `Set` variable in the class `MensurableCharacteristic` and, hence, this is the type returned by the invocation of the synonymous method on this class. In the class `Metric`, the return type of the `quantification()` method is an `MensurableCharacteristic`, since an a metric values just one characteristic.

Reflexive relations are also mapped as associations, and generate a method for each association end. The name of these methods is, instead of the relation's name, the name of the roles played by the concept. Whole-Part relations also are represented by the name of its roles. In Figure 11, the aggregation relation originates methods the `subCharacteristic()` and `superCharacteristic()` in `NonMensurableCharacteristic` and `QualityCharacteristic` respectively.

Subtype-of relations among concepts can be directly mapped to inheritance among classes. So, axioms (EA1) to (EA4) do not require any special treatment. In our example, the subtypes of quality characteristic give rise to the following sub-classes: `ProcessQualityCharacteristic`, `ProductQualityCharacteristic`, `NonMensurableCharacteristic` and `MensurableCharacteristic`. The class that represents the super-type (`QualityCharacteristic`) is mapped to an abstract class.

6.2 The Whole-Part Relation

Figure 3 presents the theory (mereology) embodied by a generic whole-part relation. Notwithstanding, the underlying axioms implied by the proposed notation are not well mapped to aggregations in an object model, i.e., UML notation for aggregation does not guarantee the fulfillment of the imposed constraints of whole-part relations. To deal with this problem, Guizzardi et al. [9] proposed the *Whole-Part Pattern*, shown in Figure 12. In this pattern, the `Whole` class is able to guarantee to its associated concrete classes the verification of the suitable set of constraints before a relation between them can be established. The interfaces `IWhole` and `IPart` must be implemented by the concrete classes.

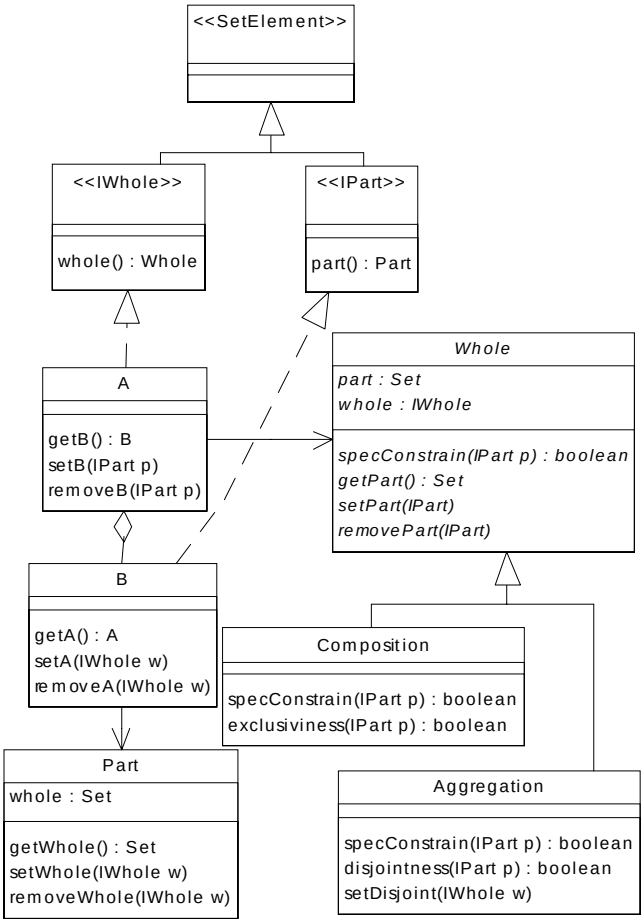


Figure 12. The Whole-Part pattern [9].

In the framework derived from the quality ontology (Figure 11), the classes `NonMensurableCharacteristic` and `QualityCharacteristic` implement interfaces `IWhole` and `IPart` respectively. Likewise, they are related to the handlers `Aggregation` and `Part`. The class `QualityCharacteristic` has attributes of `Part` type and `NonMensurableCharacteristic` has a object of `Whole` type. As shown in the code fragment below, the access to the sub-characteristics of a non-mensurable characteristic is made through an attribute `Aggregation.` The inclusion of a new sub-characteristic is made by including a new part in the aggregation variable. The axioms (EA5) to (EA8) are checked when the method `setPart()` is evoked.

```
public   class   NonMensurableCharacteristic
                          implements IWhole
{  Aggregation a = new Aggregation();
   public    boolean    setSubCharacteristic
                    (QualityCharacteristic c)
   { return a.setPart(c); }
```

```
   public Set getSubCharacteristic ()
   { return a.part(); }
}
```

The theory incorporated to the *valuation* relation in Figure 9 is presented in the code fragment below. The class `QualityCharacteristic` is related to the pattern `AntiSymmetry` through the attribute `s`. Before setting a non-mensurable characteristic as capable of valuing the current quality characteristic, the valuation theory is checked. To verify the axiom (OA1), the method `s.anti_symmetry(this,c,"valuation")` of the Anti-Symmetry pattern is executed. This method evokes the `getValuation()` method of the non-mensurable characteristic `c`. If the current characteristic (`this`) is not in the valuation list, then it doesn't value `c`. Therefore, the axioms (OA1) holds and `c` can be added to the `valuation` list of the current quality characteristic.

```
public abstract class QualityCharacteristic
                          implements IWhole
{  Set valuation = new Set();
   AntiSymmetry s = new AntiSymmetry();
   public Set getValuation()
   { return valuation; }
   public boolean setValuation
            (NonMensurableCharacteristic c)
   {  boolean result = false;
      if s.anti_symmetry(this,c,"valuation")
      {
         result = true;
         valuation.add(c);
         c.setValuation (this);
      }
      return result;
   }
}
```

Since the PreCondition Pattern composes the Whole-Part Pattern, the modifications made require changes in the last pattern. Instead of encapsulating the axioms of the generic whole-part theory, the `Whole` class is now related to the axioms properties that characterize the whole-part relation.

```
public abstract class Whole
{  IWhole whole;
   Set part = new Set();
   AntiSymmetry s = new AntiSymmetry();
   AntiReflexivity r=new AntiReflexivity();
   Atomicity a = new Atomicity();
   Transitivity t = new Transitivity();
   public boolean setPart(IPart c)
   {   boolean result = false;
       if (transitivity(this,c,"getPart")&&
        anti_symmetry(this,c,"getPart")&&
        anti_reflexivity(this,c,"getPart")&&
        atomicity (this,c,"getPart")&&
        specConstrain(c))
       {   result = true;
           part.add(c);
           (c.part()).setWhole(whole);
       }
       return result;
   }
}
```

For this reason, the `setPart()` method in the Whole class evokes the axiom properties patterns to check if the mereology theory holds. The `specConstrain()` method in the Aggregation class evokes the Disjointed pattern.

7. BROWSING ONTOLOGIES

ODEd provides automatic generation of hypertexts based on the ontologies designed. Using these hypertexts, developers are able to browse and search the domain's concepts, relations and constrains.

The language chosen to build these documents was XML [18], because it allows defining the syntax of structured documents. Besides, XML schema and ontologies have a common goal: to provide vocabulary and structure for describing information to be exchanged.

To generate the XML documents, a set of tags was defined to represent the meta-ontology's concepts and relations. The ontologies' data (concept, properties, etc.) were introduced in the XML files, marked with theses tags.

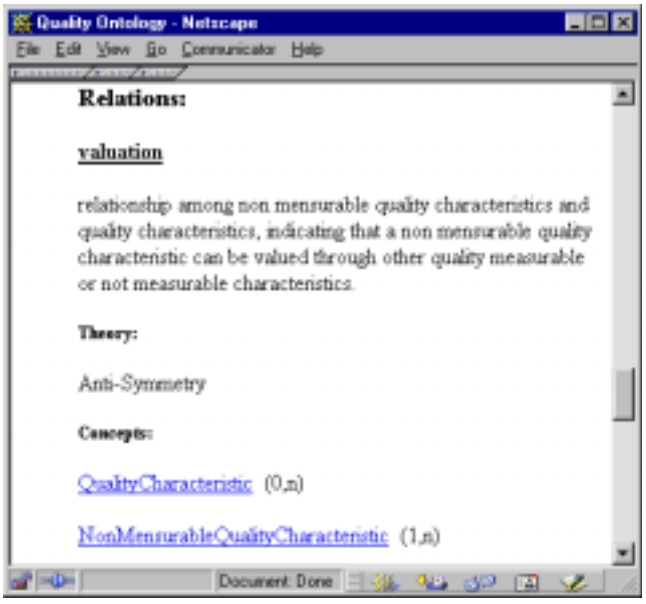


Figure 13. Browsing the Quality Ontology.

The tutorials are presented to the user as HTML documents. In order to do so, the editor uses XSL (eXtensible Style sheet Language), a document transformation and formatting language [19].

Figure 13 shows the hypertext derived from the software quality ontology. It is possible to visualize all ontology's concepts and relations and their definitions and properties. From the valuation relation, for example, the user can browse its concepts and visualize their definition.

8. RELATED WORK

There are many ontology editors presented in the literature, such as Ontolingua Server, OntoEdit, OILEd, JOE, Protége-2000 and WebODE.

Ontolingua Server [20] supports ontology development and sharing. It provides access to a library of ontologies, and allows new ontologies to be created. Remotely distributed groups can use their web browsers to browse, build and maintain ontologies stored at the server.

OntoEdit [15] pursues the modeling of ontologies such that graphical means exploited for modeling of concepts and relations scale up to axiom specifications (using RDFS). The core idea is to use an axiom categorization. This categorization is centered around axiom semantic meaning rather than syntactic representation.

OILEd [21] 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.

The *Java Ontology Editor* (JOE) [22] was developed to help users build and browse ontologies. It enables query formulation at several levels of abstraction. JOE provides a graphical user interface for editing ontologies. It uses Entity Relationship diagrams to represent them.

Protége-2000 [23] aims to support knowledge acquisition, and to reach interoperability with other knowledge representation systems. It has classes, instances of these classes, slots representing attributes of classes and instances, and facets expressing additional information about slots. Protégé-2000 generates knowledge-acquisition forms automatically based on the types of the slots and restrictions on their values allowing ontology instantiation.

Ontobroker [24] provides languages to annotate web documents with ontological information, to represent ontologies, and to formulate queries. The tool set of Ontobroker allows users to access information and knowledge from the web and to infer new knowledge with an inference engine based on techniques of logic programming. This environment is the basis for realizing the *Knowledge Acquisition Initiative (KA)*^{2} and for developing a knowledge management system for industrial designers in regard to ergonomic questions.

WebODE [25] is a workbench for ontological engineering that provides a scalable architecture for the development of other ontology development tools and ontology-based applications. WebODE's ontology editor allows the collaborative edition of ontologies at the knowledge level, supporting the conceptualization phase of METHONTOLOGY and most of the activities of the ontology's life cycle (reengineering, conceptualization, implementation, etc). It provides several services as ontology import/export, translation `of ontologies`, ontology browser, inference engine and axiom generator. The graphical user interface allows browsing all the relationships defined on the ontology as well as graphical-pruning these views with respect to selected types of relationships. Mathematical properties such as reflexive, symmetric, etc. and other user-defined properties can be also attached to the "ad hoc" relationships.

Most of these tools emphasize the definition of concepts and relations, but they have none or little support to constrains definition. The most interesting initiative is the creation of axioms templates in OntoEdit [15] and a similar approach is provide by WebODE [25]. OntoEdit's approach aids the construction of axiom classes that has similar structure, but it cannot be applied to axioms that do not fit in its classification. This approach was incorporated to ODEd in order to facilitate axioms definition, but it is still necessary to define how to represent other types of axioms as provide in WebODE [25].

Reasoning services are an important feature [21, 25] because they can be used in ontology evaluation. Other desirable services provided by some of these tools are the support to the cooperative work and the automatic generation of ontology documentation in HTML [21, 23, 25]. This last feature is addressed by ODEd but no reasoning service is available. ODEd does not provide functionalities for collaborative ontology development such as versioning, integration and merging of ontologies. Also, knowledge acquisition aspects, such those find in [23], were not considered yet.

Despite of being an important requirement for ontology design, only JOE and WebODE use some kind of graphic representation [22, 25]. But the first one uses Entity Relationship models that are not adequate to ontology development [9] and the second one doesn't define any special notation for the kinds of relations supported by the editor. ODEd adopts LINGO, a graphic language specially designed for ontology's representation. However, ODEd does not ignore the importance of other graphical languages available. Therefore it also supports ontology capture using UML.

All editors previously mentioned were developed to support ontology design in the context of Semantic Web. None of them was developed aiming to support the development of information systems, using frameworks. In this way, ODEd address domain engineering.

9. CONCLUSIONS AND FUTURE WORK

In this paper, we presented ODEd, an ontology editor that supports ontology development using graphic representations, besides promoting automatic generation of some classes of axioms and derivation of frameworks from ontologies.

Although most phases of ontology development process were supported by ODEd, important activities such as ontology integration and evaluation were not completely addressed. These features must be considered in future versions of ODEd. We believe that to support these activities reasoning services will be necessary.

In the approach presented, ODEd is capable to derivate epistemological, consolidation and ontological axioms coming

from relation theories. Other ontological axioms, which do not fit in any category of axioms properties, could also be described in the ontology. It is necessary to map ontological axioms to the object model. These axioms are formalized to answer to the competency questions of the ontology. For this type of axioms, a set of transformation rules was defined in [9]. However, it was not defined yet how this axioms should be represent in the editor.

Like Protégé [23], ODEd could also generate knowledge-acquisition forms automatically based on ontology, going a step ahead towards knowledge management.

10. ACKNOWLEDGMENTS

The authors acknowledge CAPES for the financial support to this work.

11. REFERENCES

[1] Staab, S., Studer, R., Schnurr, H.P., Sure, Y., Knowledge Processes and Ontologies, In: IEEE Intelligent Systems, p. 26-34, January/February 2001.

[2] Falbo, R.A., Menezes, C.S., Rocha, A.R.C., A Systematic Approach for Building Ontologies, In: Proceedings of the IBERAMIA'98, Lisboa, Portugal, 1998.

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

[4] Uschold, M, Gruninger, M., Ontologies: principles, methods and applications, In: Knowledge Engineering Review, Volume 11 No 2, June 1996.

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

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

[7] Uschold, M., Knowledge level modelling: concepts and terminology, Knowledge Engineering Review, vol. 13, no. 1, 1998.

[8] Jasper, R., Uschold, M., A Framework for Understanding and Classifying Ontology Applications, in Proc. of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW'99), Alberta, Canada, 1999.

[9] Guizzardi, G., Falbo, R.A., Pereira Filho, J.G., Using Objects and Patterns to Implement Domain Ontologies, In: Proceedings of XV Simpósio Brasileiro de Engenharia de Software, October 2001.

[10] Gruber, T.R., Towards principles for the design of ontologies used for knowledge sharing, Int. Journal of Human-Computer Studies, vol. 43, no. 5/6, 1995.

[11] Borst, W.N., Construction of Engineering Ontologies for Knowledge Sharing and Reuse, PhD Thesis, University of Twente, Enschede, The Netherlands, 1997.

[12] Cranefield, S., Purvis, M., UML as an Ontology Modelling Language, In: Proceedings of the IJCAI-99, Workshop on Intelligent Information, 16th International Joint Conference on AI, Stockholm, Sweden, July 1999.

[13] Falbo, R.A., Guizzardi, G., Duarte, K.C., Natali, A.C.C., Developing Software for and with Reuse: An Ontological Approach, Proceedings of the CSITeA'2002 (to appear).

[14] Evans, A., Kent, S., Core Meta-Modelling Semantics of UML: the pUML Approach, In: 2nd International Conference on the Unified Modeling Language, Colorado, EUA, 1999.

[15] Staab, S., Maedche, A., 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.

[16] Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design patterns: elements of reusable object-oriented software, Addison-Wesley, 1995.

[17] Duarte, K.C., Falbo, R.A., Uma Ontologia de Qualidade de Software, In: XIV Simpósio Brasileiro de Engenharia de Software, WQS'2000 - Workshop de Qualidade de Software, João Pessoa - PB, October 2000.

[18] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Extensible Markup Language (XML) 1.0, W3C Recommendation, 1998.

[19] Rabarijoana, A., Dieng, R., Corby, O., Exploitation of XML for Corporate Knowledge Management, In: Proceedings of EKAW'99, p. 373-378, 1999.

[20] Farquhar A., Fikes, R., Rice, J., The Ontolingua Server: a tool for collaborative ontology Construction, Int. J. Human-Computer Studies, 46, p. 707-727, Knowledge Systems Laboratory, Stanford University. Stanford, CA, USA, 1997.

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

[22] Mahalingam, K., Huhns, M.N., A Tool for Organizing Web Information, In: IEEE Computer, p. 80-83, June 1997.

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

[24] Fensel, D., Decker, S., Erdmann, M., Studer. R., Ontobroker: Or How to Enable Intelligent Access to the WWW, In: Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop, Banff, Canada, April 1998.

[25] Arpírez, J.C., Corcho, O., Fernández-López ,M., Gómez-Pérez, A., WebODE: a Scalable Workbench for Ontological Engineering, K-CAP'01, Canadá, 2001.