An Ontological Approach to Domain Engineering

Ricardo de Almeida Falbo
falbo@inf.ufes.br

Giancarlo Guizzardi
gguizz@inf.ufes.br

Katia Cristina Duarte
duarte@inf.ufes.br

Computer Science Department, Federal University of Espírito Santo
Fernando Ferrari Avenue, CEP 29.060-900,
Vitória, Espírito Santo, Brasil
055-27-33352167

ABSTRACT
Domain engineering aims to support systematic reuse, focusing on modeling common knowledge in a problem domain. Ontologies have also been pointed as holding great promise for software reuse. In this paper, we present ODE (Ontology-based Domain Engineering), an ontological approach for domain engineering that aims to join ontologies and object-oriented technology.

Categories and Subject Descriptors
D.2.13 [**Software Engineering**]: Reusable software – *domain engineering.*

General Terms
Design, Theory.

Keywords
Ontology, software reuse.

1. INTRODUCTION
In order to get the main benefits of software reuse, we need first to develop *for* reuse, so then we can develop *with* reuse. Domain engineering concerns developing *for* reuse, and ontologies can play an important role in this context. An ontology can promote common understanding among developers, and can be used as a domain model.

However, one of the major drawbacks to a wider use of ontologies in Software Engineering is the lack of approaches to insert ontologies in a more conventional software development process. Since the current leading paradigm in Software Engineering is the object-oriented technology, to put ontologies in practice, we need an approach to derive object models from ontologies in order to derive widely reusable assets.

In this paper, we propose an ontology-based approach to domain engineering that considers two main phases: building ontologies and deriving object frameworks from them. Section 2 discusses domain engineering and why ontologies are useful in this context. In sections 3 and 4, we present our ontological approach to domain engineering and a study case in the software quality domain, respectively. Section 5 discusses related works. Finally, section 6 report our conclusions.

2. DOMAIN ENGINEERING AND ONTOLOGIES
Domain engineering concerns the work required to establish a set of software artifacts that can be reused by the software engineer. The purpose of domain engineering is to identify, model, construct, catalog and disseminate a set of software artifacts that can be applied to existing and future software in a particular application domain [1].

As pointed by Arango [2] in its forerunner paper, a domain engineering process should encompass at least three main activities: domain analysis, infrastructure specification and infrastructure implementation.

Domain analysis involves identification, acquisition and analysis of domain knowledge to be reused in software specification and construction. The purpose of domain analysis is to produce a model of the problem domain. The domain model should serve as: (i) an unified source of reference when ambiguities arise in the analysis of problems or latter during the implementation of reusable components; (ii) a repository of the shared knowledge for teaching and communication; and (iii) a specification to the developer of reusable components [3].

However, generally, a domain model is not directly useful to operational reuse. There exists a gap between the kinds and forms of the domain knowledge in a domain model and the content and form of software assets that can be reused in software construction. To bring this gap, we need to build a reuse infrastructure. This infrastructure should support the efficient operation of a reuse system and should also be adapted to its technology [3].

The purpose of the infrastructure specification activity is to define the aspects of the problem domain that should be supported by the component repository in order to achieve the reuse system requirements. This involves selecting and organizing the reusable information, and determining how it should be packaged into components and how these components should be indexed. The result is an architecture that specifies the components that should be available to the reuse system [3].

This infrastructure specification, together with the semantics captured by the domain model, are the input to the infrastructure implementation step that actually produces and tests the components [3].

In domain engineering, ontologies can play several roles. According to Uschold [4], "an ontology may take a variety of forms, but necessarily it will 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". Thus, an ontology consists of concepts and relations, and their definitions, properties and constrains expressed as axioms. An ontology is not only an hierarchy of terms, but a fully axiomatized theory about the domain [5].

In a general point of view, applications of ontologies can be classified in four main categories, emphasizing that an application may integrate more than one of these categories [6]:

- 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 this case, an ontology can be viewed as a domain model. The ontology is used as a basis for specification and development of domain applications, 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 searching.

Although we are most interested in the use of ontologies as specification, i.e. as domain models, the other purposes are also important to domain engineering.

The neutral authoring scenario is important, mainly when applications will be developed using different technology (e.g., objects and logics). This insight shows that we need to define different approaches to implement different reuse infrastructures, each one suitable to the corresponding system reuse technology.

Common access to information scenario is essential to avoid misunderstanding among developers. It is vital for reuse tasks, such as adapting components and creating new assets based on existing ones, as well as for selecting black-box components and for providing access to shared data and services.

Finally, an ontology-based search has great potential to improve structuring and searching in component libraries. An ontology can be used for structuring and organizing the information repository (in our case, a component library). It may be used as a conceptual framework to developers think about this repository and formulate queries. Also it can be used to perform inference to improve the query [6].

Analyzing these scenarios, we can notice that domain engineering can take several advantages from the use of ontologies. However, an ontology-based domain engineering process must be flexible enough to consider all these scenarios.

First, we need a systematic approach for building ontologies (domain analysis). Second, we need several approaches for reuse infrastructure specification and implementation, each one considering a specific reuse system technology. Since nowadays the object-oriented technology is widely used, we proposed an approach to derive object frameworks from ontologies.

3. ONTOLOGY-BASED DOMAIN ENGINEERING

In this section, we present ODE (Ontology-based Domain Engineering), an ontological approach to domain engineering, that considers ontology development (domain analysis), its mapping to object models (infrastructure specification), and Java components development (infrastructure implementation).

3.1 A Systematic Approach for Building Ontologies

Figure 1 shows the main activities in ODE's ontology development process [5]. The dotted lines indicate that there is a constant interaction, albeit weaker, between the associated steps. The filled lines show the main work flow in the ontology building process. The box involving the capture and formalization steps enhances the strong interaction, and consequently iteration, between them.

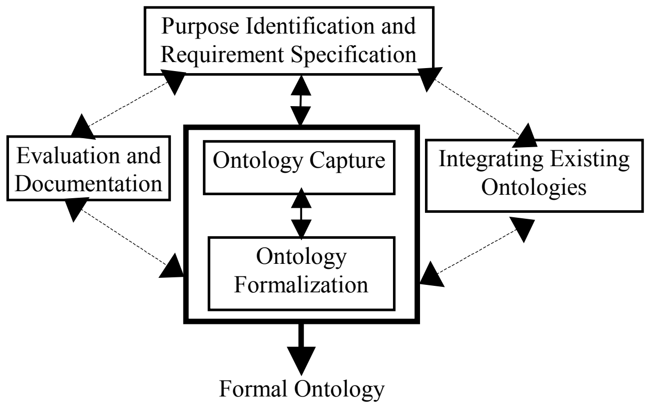

Purpose Identification and Requirement Specification
Evaluation and Documentation
Ontology Capture
Integrating Existing Ontologies
Ontology Formalization
Formal Ontology

Figure 1 - Steps in the ontology development process.

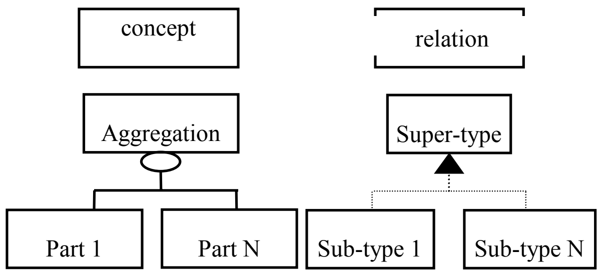
A brief description of the activities is presented below:

- **Purpose identification and requirement specification**: concerns to clearly identify the ontology purpose and its intended use, that is, the competence of the ontology. To do that, we suggest the use of competency questions [7].
- **Ontology capture**: the goal is to capture the domain conceptualization based on the ontology competence. The relevant domain entities (e.g. concepts, relations, properties, role) 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. This language should be able to represent in a precise and unambiguous way the elements that model the existing domain entities. One should be able to write formal axioms

that constrain the interpretation of the structure formed by these entities.

- **Integrating Existing Ontologies:** during ontology capture and/or formalization, it could be necessary to integrate the current ontology with existing ones to use previously established conceptualizations. Indeed, it is a good practice to develop general modular ontologies, more widely reusable, and to integrate them, when necessary, to obtain the desired result.
- **Evaluation:** the ontology must be evaluated to check whether it satisfies the specification requirements. It should be evaluated in relation to the ontology competence and some design quality criteria, such those proposed by Gruber [10]. It should be noticed that the competency questions play an essential role in the evaluation of the completeness of the ontology, specially when considering its axioms.
- **Documentation:** all the ontology development must be documented, including purposes, requirements, textual descriptions of the conceptualization, and the formal ontology. A potential approach to document an ontology is to use a hypertext, allowing browsing along term definitions, examples and its formalization, including the axioms. The use of XML can be worthwhile.

As pointed above, during ontology capture, we need a graphical language to improve the communication with domain experts. We have proposed LINGO [5] as a graphical language for expressing ontologies. LINGO basically represents a meta-ontology, and thus, it defines the basic notations to represent a domain conceptualization. That is, 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. This way, axioms can be automatically generated. These axioms concern simply the structure of the concepts and are said *epistemological axioms*. Figure 2 shows part of LINGO notation and some of the axioms imposed by the whole-part relation. These axioms form the core of the mereological theory as presented in [8].



(A1) $\forall x \ \neg partOf(x,x)$
(A2) $\forall x,y \ partOf(y,x) \leftrightarrow wholeOf(x,y)$
(A3) $\forall x,y \ partOf(y,x) \rightarrow \neg partOf(x,y)$
(A4) $\forall x,y,z \ partOf(z,y) \wedge partOf(y,x) \rightarrow partOf(z,x)$
(A5) $\forall x,y \ disjoint(x,y) \rightarrow \neg\exists z \ partOf(z,x) \wedge partOf(z,y)$
(A6) $\forall x \ atomic(x) \rightarrow \neg\exists y \ partOf(y,x)$

Figure 2 - LINGO main notation and some axioms.

Besides the epistemological axioms, other axioms can be used to represent knowledge at a signification level. These axioms can be of two types: *consolidation axioms* and *ontological axioms* [5]. 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.

Someone could argue that another graphical language is unnecessary. Cranefield and Purvis [9], for example, advocate the use of UML as an ontology modeling language. We partially agree with their arguments, but we decided not to use some existing graphical language due two main related reasons. First, an important criterion to evaluate ontology design quality is minimum ontological commitments [10]. Based on this principle, a graphical language in this context 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. 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 [11]. Based on that, we are also studying to define a subset of UML that can play the same role of LINGO following the same thread of [9].

We advocate, based on our experience in ontology development, that the approach described easies the development of ontologies, specially in those aspects concerning minimum ontological commitments criterion. However, when considering ontology as a specification, this striking feature is also a problem, since the ontology is built generally in a high abstraction level to be directly reused in software development. We have experimented to reuse ontologies in the development of knowledge-based systems, information systems (using object technology) and hypermedia systems. In all cases, we identify a need to lower the abstraction level of our ontologies to actually put them in practice. To deal with this problem, we have been working in ways to create more reusable assets from the ontologies. Next, we present our approach to derive object-based artifacts from ontologies.

3.2 From Domain Ontologies to Objects

If we want an object-based reuse infrastructure, we need an approach to derive objects from ontologies. We developed a systematic approach that is composed of a set of directives, design patterns and transformation rules [12]. 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). Contrariwise, design patterns and transformation rules are applied in the mapping of ontological and consolidation axioms, respectively. The application of these guidelines is supported by a Java Set framework that implements the mathematical type Set [12].

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, we used a hybrid approach based on pure first-order logic, relational theory and, predominantly, set theory. The choice to create a language mainly based on set theory was highly motivated by an important issue: set theory is a complementary extensional perspective to the intentional nature of first-order logic. For example, let the intention of the concept mortal be "A mortal is an entity whose life ceases in a point of time". The logic predicate **mortal(x)** states that **x** is a mortal and, therefore, the characteristics defined by the intention of this concept applies to **x**. It also (implicitly) states that $\mathbf{x} \in \mathbf{Mortal}$, i.e. to the set of all elements in the considered world to which the intention of the concept applies. In an object-oriented perspective, if **x** is an instance of **Mortal**, it means that **x** belongs to the **Mortal** class, i.e. to the set of all instances that share the same properties and the same definition.

In our approach, concepts are defined as sets, and, as mentioned before, the statement $\mathbf{x} \in \mathbf{Person}$ commits **x** to the concept **Person**, both intentionally and extensionally.

Another fundamental building block in the LINGO meta-ontology is the relation primitive. This primitive represents a semantic link that exists among concepts. In our approach, relations are mapped to the synonymous primitive in set theory. In set theory, a n-ary relation can be defined by the n-tuple $\mathbf{R} = (\mathbf{C_1,C_2,...,C_n,\ p(x_1,x_2,...,x_n)})$, where each $\mathbf{C_i}$ represents a different set involved in the relation and $\mathbf{p(x_{i},)}$ is a functional predicate open in *n* variables that maps each element from the cross-product $\mathbf{C_1 \times C_2 \times ...\ C_n}$ in a true or false value. In this case, the set $\mathbf{R^*}$ (solution set) is the subset of $\mathbf{C_1 \times C_2 \times ...\ C_n}$ whose members $\mathbf{e_i}$ all satisfy the predicate $\mathbf{p(e_i)}$. From now on, the propositional function $\mathbf{p(x_1,x_2,...,x_n)}$ will be used as synonym of the n-tuple that defines the relation, assuming that it is defined in some cross-product $\mathbf{C_1 \times C_2 \times ...\ C_n}$.

Figure 3 shows an example of a binary relation that links the concepts **Person** and **Organization** in a context of Enterprise Modeling. In this figure, **age** is a property of **Person**. The equivalent description of the contract relation in set theory is **contract = ((Organization, Person, contract(x,y))**.

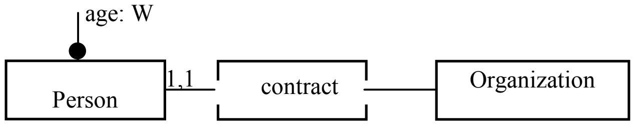


Figure 3 - Example of a binary relation.

Finally, it is important to notice that the relation **contract** defines a mapping from the set **Organization** to the powerset of **Person**. For this reason, for each **(x,y)** in **Contract***, $\mathbf{x} \in \mathbf{Organization}$ and $\mathbf{y} \in \mathbf{Person}$. Therefore, to navigate in the opposite direction, one must use the reverse mapping defined by the inverse relation **contract~**.

In set theory, some essential operations are defined to express the relations between sets (such as $\subset$ - subset; $\cup$ - Union; $\cap$ - Intersection; \ - difference and $\wp$ - power set), properties of sets (# - cardinality), restriction on relations (~ - inverse relation) and relations between sets and their members ($\in$ - Membership) [12]. In addition to this, we use the basic logic operators ($\wedge$ - conjunction; $\vee$ - disjuntion; $\oplus$ - exclusive disjunction; $\neg$ - negation; $\rightarrow$ - conditional; $\leftrightarrow$ - biconditional) and quantifiers ($\forall$ - universal; $\exists$ - existential; $\exists!$ - exists one and only one) to form the core of the formalism employed in this work.

In order to extend this core formalism, some additional functions were defined. The two most important among them are the functions *set relational Image* **(Im)** and the *element relational image* **(Im₊)**. The definitions[1] of **Im** and **Im₊** are given as follows:

$$\mathbf{Im_{+}(\_,\_): X \times (X \Leftrightarrow Y) \rightarrow \wp(Y)}$$

$$\mathbf{Im_{+}(x,R) = \{x{:}X,y{:}Y \mid ((x,y) \in R^*) \bullet y\ \}}$$

$$\mathbf{\forall\ a{:}A,b{:}B,R{:}(A \Leftrightarrow B)\ b \in Im_{+}(a,R) \leftrightarrow a \in Im_{+}(b,R\sim)}$$

$$\mathbf{Im(\_,\_): \wp(X) \times (X \Leftrightarrow Y) \rightarrow \wp(Y)}$$

$$\mathbf{Im(S,R) = \bigcup_{a \in S} Im_{+}(a,R)}$$

Using the relation of figure 3 as an example, a possible valid image set could be: $\mathbf{Im_{+}(Org_1,\ contract) = \{John,\ Paul,\ Mary\}}$ and, consequently, $\mathbf{Im_{+}(John,contract\sim) = \{Org1\}}$. It is important to notice that, as stated by the axioms above, the set **Im** function distributes over the element **Im₊** function, i.e., $\mathbf{Im(\{John,Mary\},contract\sim) = Im_{+}(John,\ contract\sim) \cup Im_{+}(Mary,\ contract\sim)}$.

The use of cardinality constraints of type (1,1) in this example implies that $\mathbf{\forall\ p{:}\ Person\ \#Im_{+}(p,contract\sim) = 1}$ and cardinality constraints of type (1,n) implies that $\mathbf{\forall o{:}Organization\ \#Im_{+}(o,contract) \geq 1}$.

The axiom below completes the formal definition of our set-based language providing the semantics of the selection operator in this context. In this definition we assume the set Φ as the superset of all basic types, such as, $\aleph$, $\Re$, Boolean, Strings, and so on.

$$\mathbf{\sigma(\_,\_,\_,\_): \wp(X) \times (X \rightarrow \Phi) \times (\Phi \Leftrightarrow \Phi) \times \Phi \rightarrow \wp(X)}$$

$$\mathbf{\sigma(X,R,O,z) = \{x{:}X,y,z{:}\Phi \mid ((x,y) \in R^*) \wedge ((y,z) \in O^*) \bullet x\}}$$

The selection operator takes as parameters: (i) a set **X** (e.g. **Person**); (ii) a property (function) existent between **X** and the a subtype of $\mathbf{\Phi}$ (e.g. **age**, where the subtype of $\mathbf{\Phi}$ is $\aleph$); (iii) a relation (operation) defined for the specific subtype of $\mathbf{\Phi}$ (e.g., $\geq$); (iv) and an instance value of $\mathbf{\Phi}$ (e.g., 20).

Since we have defined this set-based formalism to support ontology to objects mapping, the first step in our approach is the complete axiomatization of the domain theory using this formalism.

Once defined the Set-based axioms, we can initiate the object mapping. First, we should consider the epistemological aspects captured by LINGO. Concepts and relations are naturally mapped to classes and associations in an object model, respectively. Properties of a concept shall be mapped to attributes of the class

[1] One shall notice that the symbol $\Leftrightarrow$ (used as in $A \Leftrightarrow B$) is a meta-mathematical construct that represents the set of existent relations between the sets A and B. Differently, the symbol $\leftrightarrow$, represents the logical biconditional. Moreover, although the symbol $\rightarrow$ is used both for functions definition and for logical implication, its semantics shall be made clear by the usage context.

that is mapping the concept. In the example of Figure 3, **Person** and **Organization** would be mapped to classes, and **age** would be modeled as an attribute of **Person**.

Although this approach works well in most cases, it is important to point out some exceptions:

- some concepts can be better mapped to attributes of a class because they do not have a meaningful state in the sense of an object model;
- some concepts should not be mapped to an object model because they were defined only to clarify some aspect of the ontology, but they do not enact a relevant role in an object model;
- relations involving a concept that is mapped to an attribute (or that is not considered in the mapping) should not be mapped to the object model.

A class defines a formation rule for its instance and, therefore, can be seen as a set. Consequently, the classification relations in the formalism do not require any specific implementations, i.e. relations such as $\mathbf{a} \in \mathbf{A}$, are totally resolved by the programming language typing mechanism through the creation of an object **a** of type **A**. Likewise, subtype-of relations among concepts can be directly mapped to generalization/specialization relations among classes. However, it is not the case of Whole-Part relations. The UML notation for aggregation does not guarantee the fulfillment of the mereology theory constraints. To deal with this problem, we developed the whole-part design pattern [12].

For the mapping of *relations*, there are some issues that still must be discussed. As shown in Figure 3, there is a relation **contract** between the concepts **Person** and **Organization**. In our approach, this relation is translated to an association between the corresponding two classes in the object model and both classes have a method `contract()`. In this case, with the invocation of method `contract()` in an object o_1 of type `Organization`, it is possible to have access to all the people that work at o_1. This resulting set is formally specified by the formula $\mathbf{Im_+(o_1,contract)}$**)**. Likewise, the method invocation in a Person instance p_1 returns its employer Organization, or, $\mathbf{Im_+(p_1,contract\sim)}$. The returned type of the relation methods depends directly on the cardinality axioms associated to the relation. For instance, since in the scope of the **contract** relation an **Organization** may employ several Persons, **contract** is mapped to a `Set` variable in the `Organization` class and, hence, this is the type returned by the invocation of the synonymous method on this class.

Once mapped the epistemological structure, we should consider consolidation and ontological axioms. To address the consolidation axioms mapping, we developed a design pattern (consolidation pattern) whose purpose is to describe preconditions that must be satisfied or properties that must hold so that a relation could be established between two objects [12].

Finally, it is necessary to map ontological axioms to the object model. Methods are derived from ontological axioms, using a set of transformation rules, partially presented below.

T0: $\forall\, x{:}X, \forall\, y{:}Y\ r_1(x,y) \leftrightarrow y \in C \Rightarrow$
$\mathbf{Im_+(x, r_1)}$**:Type = C,** such that if # $\mathbf{Im(x, r_1) = 1}$
then **Type = Y** else **Type = Set**

This rule states that the type returned by the method that implements the function in the derived class depends on the cardinality of the relation. Hence, if **x** is related to only one instance of **Y**, the returned value shall be of type **Y**, otherwise, it shall be of type Set, in the case a set of **Y**.

T1: $\mathbf{Im_+(x, r_1)} \Rightarrow$ **x.r1()**

T2: $\mathbf{Im_+(y, r_1\sim)} \Rightarrow$ **y.r1()**

T3: $\mathbf{r_1(x,y)} \Rightarrow$ **x.r1()**

T4: $\mathbf{r_1(x)} \Rightarrow$ **x.r1()**

A relation $\mathbf{r_1}$ between two concepts **X** and **Y** is mapped to methods in the corresponding classes. Given an instance **x**, the invocation **x.r1()** returns the set of objects from **Y** associated to **x** in the relation $\mathbf{r_1}$.

T5: A SetTheoryOperation *a* $\Rightarrow$
A.SetTheoryOperationImplementation(*a*)

This rule deals with the translation between the essential set theory operations and the corresponding method implemented in the Set class. For instance, the set theory expression $A \cap C$ is translated to A.intersection(C), where A and C are instances of the class Set.

T6: $\mathbf{Im(A, r_1)} \Rightarrow$ **Set.Im(A,"** $\mathbf{r_1}$**")**

This rule promotes the replacement of the *Set Relational Image* function by the corresponding syntax through which it is implemented in the `Set` class.

T7: $x.r_1():Y \equiv C \Rightarrow$
```
public class X
{
    public Y r1( )
{
    return C;
}
}
```

Finally, this last rule directly translates the axiom written in the left side to the corresponding Java implementation syntax. All the existing references to the instance **x** in the scope of set **C** (to which **x** belongs) are replaced by the Java reserved word `this`, so that references to methods of the same class can be made.

4. APPLYING *ODE* IN SOFTWARE QUALITY DOMAIN

We have been using ODE in several domains, such as software process modeling [12], software quality and video on demand. In this section we present partially its application in the software quality domain.

As pointed by Crosby, cited by Pressman [1], "the problem of quality management is not what people don't know about it. The problem is what they think they do know". Before we can devise a strategy for producing quality software, we must understand what software quality means. But this is not an easy task. There are several information sources (books, standards, papers, experts, and so on) using many different terms with no clear semantics established. There is not a consensus about the terminology used, what causes misunderstanding and several problems in the definition of a quality program. To deal with these problems, we developed an ontology of software quality. Several books, standards, and experts were consulted and a consensus process was conducted.

4.1 A Software Quality Ontology

Due to limitations of space, we present only part of this ontology, concerning the following competency questions:

1. Which is the nature of a quality characteristic?
2. In which sub-characteristics can a quality characteristic be decomposed?
3. Which characteristics are relevant to evaluate a given software artifact?
4. Which metrics can be used to quantify a given characteristic?
5. To which paradigm a quality characteristic is applicable?

To address these competency questions, the concepts and relations shown in Figure 4 were considered. As shown in this figure, 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 subcharacteristics to be computed by the aggregation of their subcharacteristic measures. A mensurable characteristic can be directly measured applying some metric. The second classification enforces that product characteristics should only be used to evaluate software artifacts. Finally, there are some quality characteristics that can be useful only to evaluate processes or artifacts developed following some paradigm. Artifact and Paradigm are highlighted since they are concepts from the software process ontology [5], which were integrated with the quality ontology been presented.

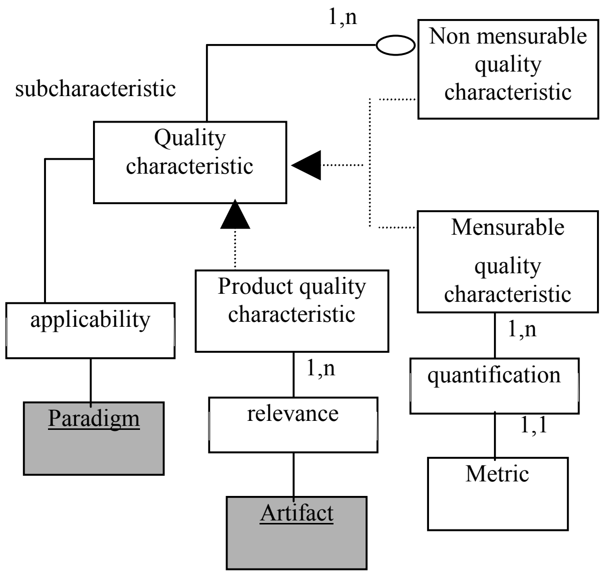


Figure 4 - Part of the software quality ontology.

From LINGO notation, the following epistemological axioms can be derived:

$(\forall\ qc)\ (nmensqc(qc) \rightarrow qchar(qc))$ (E1)

$(\forall\ qc)\ (mensqc(qc) \rightarrow qchar(qc))$ (E2)

$(\forall\ qc)\ (prodqc(qc) \rightarrow qchar(qc)\ )$ (E3)

$(\forall\ qc_1, qc_2)\ (subqc(qc_1, qc_2) \rightarrow\ \neg\ subqc(qc_2\ , qc_1))$ (E4)

$(\forall\ qc_1, qc_2)\ (subqc(qc_1, qc_2) \leftrightarrow\ superqc(qc_2\ , qc_1))$ (E5)

$(\forall\ qc_1, qc_2, qc_3)\ (subqc(qc_1, qc_2) \wedge subqc(qc_2, qc_3) \rightarrow subqc(qc_1\ , qc_3\ ))$ (E6)

$(\forall\ qc)\ (mensqc\ (qc) \leftrightarrow \neg\ (\exists\ qc_1)\ (subqc(qc_1, qc)))$ (E7)

$(\forall\ qc)\ (nmensqc(qc) \rightarrow (\exists\ qc_1)\ (subqc(qc_1, qc)))$ (E8)

$(\forall qc,m)(mensqc(qc) \rightarrow (\exists\ m)\ (quant(m, qc))$ (E9)

$(\forall qc,a)(prodqc(qc) \rightarrow (\exists\ a)\ (relev(a, qc))$ (E10)

where the predicates *qchar*, *nmensqc*, *mensqc* and *prodqc* formalize the concepts of quality characteristic, non mensurable quality characteristic, mensurable quality characteristic and product quality characteristic, respectively, and the predicates *subqc/superqc*, *quant* and *relev* formalize the whole-part, quantification and relevance relations, respectively.

Axioms (E1) to (E3) are derived by the subsumption theory. Axioms (E4) to (E7) are some imposed by the whole-part relation. Finally, axioms (E8) to (E10) are given by cardinality constraints.Several consolidation axioms were defined, such as:

$(\forall qc,qc_1)(subqc(qc_1,qc)\wedge prodqc(qc)\rightarrow prodqc(qc_1))$ (C1)

This axiom says that if a product quality characteristic (qc) is decomposed in subcharacteristics (qc_1), then these subcharacteristics should also be a product quality characteristic.

Also several ontological axioms were defined, such as:

$(\forall qc)\ \neg(\exists p)(applicability(qc,p)\rightarrow\ pdgInd(qc)$ (O1)

This axiom states that if there is not a paradigm to which a quality characteristic qc is applicable, than qc is paradigm-independent.

4.2 Object-based Domain Reuse Infrastructure

From the ontology presented, we derived a framework, shown in Figure 5, following the approach described in subsection 3.2.

All classes derived directly from the ontology are prefixed by the character "K", indicating that their objects represent knowledge about the software quality domain. The remainder classes are from the Whole-Part design pattern used. The *Whole* class, for instance, is a handler that maintains a reference to the parts associated to this whole. The interfaces *IWhole* and *IPart* must be implemented by the concrete classes (respectively *KNonMeasurableQC* and *KQualityCharacteristic*). The methods *whole()* and *part()* on these interfaces provide access to its respective handlers (*Whole* and *Part*).

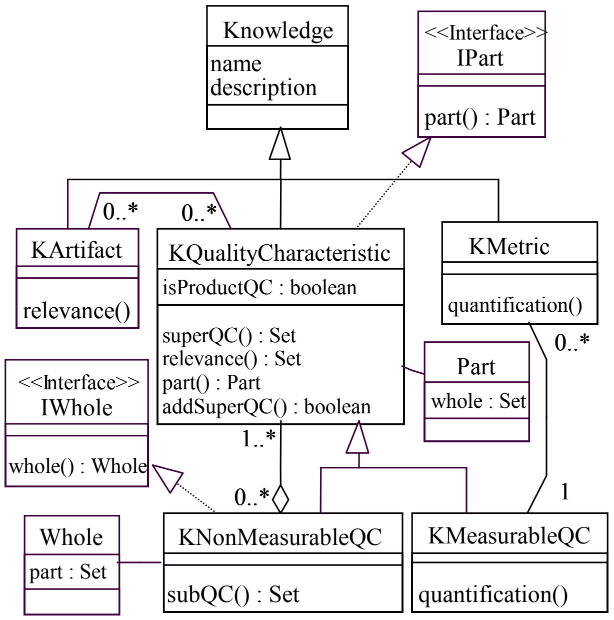


Figure 5 - Part of the Knowledge Package.

The consolidation axiom (C1) was implemented by the method *addSuperQC*, using the consolidation pattern [12], as shown in Figure 6.

```
addSuperQC (KNonMeasurableQC: qc): boolean
{
        boolean result = false;
        if (result = (qc.isProductQC && this.isProductQC))
        {
                superQC.add(qc);
                qc.addSubQC(this);
        }
        return result;
}
```

Figure 6 - Consolidation axiom mapping.

The ontological axiom (O1) was translated to the set-theory formalism and mapped to method `pdgInd()`, following the transformation rules, as follows:

1. $\forall$**qc:QualityCharacteristic pdgInd(qc,True)** $\leftrightarrow$ **#Im$_+$(qc,applicability) = 0** O1

2. **qc.pdgInd():Boolean** $\equiv$ **#Im$_+$(qc,applicability) = 0** 1,T0

3. **qc.pdgInd():Boolean** $\equiv$ **(qc.applicability().card()) = 0** 2,T1,T5

4.
```
public class KQualityCharacteristic
{                                                   3,T7
 public Boolean pdgInd()
 {
 return(this.applicability()).card()== 0);
 }
}
```

where method `card()` returns the number of elements of a given set, and is defined in the `Set` class [12].

5. RELATED WORK

There are several domain engineering methods described in the literature, such as FODA, RSEB and ODM. FODA [13] (Feature Oriented Domain Analysis) emphasizes descriptions of domain features as a way of capturing commonality and variability information. Features are captured in a feature model with semantics roughly equivalent to an AND-OR graph. However, the semantics of feature is not precisely defined, i.e. it is not formally clear what is meant by "feature" [14].

RSEB (Reuse-driven Software Engineering Business) [15] is a systematic model-driven approach to domain-specific, object-oriented software reuse. Use case models are central to all steps in RSEB. However, as pointed by Griss et al. [15], RSEB is incomplete with respect to domain analysis, since its domain analysis activities are distributed across various processes and RSEB does not provide explicit concepts/feature models. To solve this problem, the FODA's feature model was adapted to RSEB, originating FeatuRSEB [15]. The feature model is used as a catalog of feature commonality and variability and it is similar to a data dictionary (in domain engineering context, a domain terminology dictionary).

In ODM [14] a domain can be any "realm of discourse" where commonality and variability of multiple exemplars are examined. The core ODM model describes the conceptual and organizational processes that occur in such a modeling context. A goal of ODM is to define the core domain modeling process in a manner independent of assumptions about the specific modeling representation used. ODM uses a multi-modeling approach based on a mathematical formalism, called Sigma. In Sigma, the "feature" relation is formally defined as an essential property for its associated concept (a necessary condition). A concept is adequately modeled when its set of related features provides both necessary and sufficient conditions.

Comparing these methods with our approach, we should observe some aspects:

- Many methods commit a priori with a technology, mainly object technology. Like ODM, our ontology-based approach to domain analysis aims to be independent of reuse technology.
- Methods like FODA and particularly RSEB, which is use-case centered, are very interested in capturing domain functionality instead of capturing domain conceptualizations. In ODE, we agree with Guarino [16] who defends the thesis of the independence between domain knowledge and problem-solving (task) knowledge. So, in a domain ontology, we do not capture task knowledge. To deal with domain functionality, we are studying task ontologies and how to integrate them with domain ontologies in a more general reuse approach.
- Most methods do not define formal axioms to constrain the interpretation of terms. The feature model in FeatuRSEB, for example, resembles a data dictionary. In ODE, we advocate that formal axioms must be explicitly defined. This is very important for automated tools.

- In ODE, we are most interested in capturing domain conceptualizations. So, a model of concepts is the main output of ODE's domain analysis step. We do not explicitly treat features. As ODM, we think that features can be described as properties and axioms (conditions in ODM's vocabulary).

There are also several works that are related to some part of our approach. Uschold and King [17] proposed what they called "a skeletal methodology for building ontologies", defining a small number of stages that they believed would be required for any future comprehensive methodology. In this sense, the method here proposed followed some of their guidelines and stretched it towards a more systematic approach for building ontologies. In [18] a set of design patterns for constraint representation in JavaBeans components is presented. Constraints are equivalent to what we call consolidation axioms and our approach to implement these axioms is also based on design patterns. However, these axioms represent only a subset of the knowledge that must be made explicit at the ontological level. Thus we need other mechanisms to capture, for example, ontological axioms, such as the transformation rules we have proposed.

6. CONCLUSIONS

Ontologies have great potential to be used as a domain model. In this paper we presented ODE, an ontological approach to domain engineering, matching ontologies and objects, and we showed its application in software quality domain. We argue that using ODE, we can put ontologies in practice. We have tested ODE in different domains and we have obtained good results.

Since several steps of ODE can be at least partially supported by automated tools, we are working on a tool, an ontology editor, that supports ODE.

7. REFERENCES

[1] R.S. Pressman, *"Software Engineering: A Practitioner's Approach"*, 5th Edition, New York: McGraw-Hill, 2000.

[2] G. Arango, "Domain Analysis - From Art Form to Engineering Discipline", in Proc. of the 5th Int'l Workshop on Software Specification and Design, CS Press, Los Alamitos, California, USA, pp.152-159, 1989.

[3] G. Arango, R. Prieto-Diaz, "Domain Analysis Concepts and Research Directions", in Domain Analysis and Software Systems Modeling, IEEE Computer Society Press, 1991.

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

[5] R.A. Falbo, C.S. Menezes, and A.R.C. Rocha, "A Systematic Approach for Building Ontologies", in Proceedings of the IBERAMIA'98, Lisbon, Portugal, 1998.

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

[7] M. Grüninger and M.S. Fox, "Methodology for the Design and Evaluation of Ontologies", Technical Report, University of Toronto, April 1995.

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

[9] S. Cranefield and M. Purvis, "UML as an Ontology Modelling Language", in Proc. of the IJCAI'99 Workshop on Intelligent Information Integration, Sweden, 1999.

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

[11] A.S.Evans and S.Kent, "Meta-modelling semantics of UML: the pUML approach", in Proc. of the 2nd International Conference on the Unified Modeling Language. Editors: B.Rumpe and R.B.France, Colorado, LNCS 1723, 1999.

[12] G. Guizzardi, R. A. Falbo and J.G. Pereira Filho, "Using Objects and Patterns to Implement Domain Ontologies", in Proc. of the 15th Brazilian Symposium on Software Engineering, Rio de Janeiro, Brazil, 2001.

[13] K. Kang, S. Cohen, et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, Technical Report CMU/SEI-90-TR-21, SEI, Pittsburgh, 1990.

[14] M. Simos and J. Anthony, "Weaving the Model Web: A Multi-Modeling Approach to Concepts and Features in Domain Engineering", Proc. 5th Int'l Conference on Software Reuse, Victoria, Canada, 1998.

[15] M. Griss, J. Favaro, and M. Alessandro, "Integrating Feature Modeling with the RSEB", Proc. 5th Int'l Conference on Software Reuse, Victoria, Canada, 1998.

[16] N. Guarino, "Understanding, building and using ontologies", *Int. Journal Human-Computer Studies*, v. 45, n. 2/3 (Feb/Mar) 1997.

[17] M. Uschold and M. King, "Towards a Methodology for Building Ontologies", in Proc. Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI'95, 1995.

[18] H. Knublauch, M. Sedlmayr and T. Rose, "Design Patterns for the Implementation of Constraints on JavaBeans", in Proc. of the Net Object Days 2000, Erfurt, Germany, 2000.