From Domain Ontologies to Object Oriented Frameworks

Giancarlo Guizzardi
Universidade Federal do Espírito Santo
234 PUBLICATIONS **4,096** CITATIONS
SEE PROFILE

Ricardo de Almeida Falbo
Universidade Federal do Espírito Santo
172 PUBLICATIONS **1,661** CITATIONS
SEE PROFILE

José Vila Real Gonçalves
Universidade Federal de Ouro Preto
8 PUBLICATIONS **56** CITATIONS
SEE PROFILE

José Gonçalves Pereira Filho
Universidade Federal do Espírito Santo
44 PUBLICATIONS **274** CITATIONS
SEE PROFILE

From Domain Ontologies to Object-Oriented Frameworks

Giancarlo Guizzardi[1,2], Ricardo de Almeida Falbo[1], José Gonçalves Pereira Filho[1]

Computer Science Department[1]
Federal University of Espírito Santo
Fernando Ferrari Avenue,
CEP 29060-900 - Vitória - ES - Brazil
e_mail: {falbo, zegonc}@inf.ufes.br

Centre for Telematics and Information Technology[2],
University of Twente
P.O. Box 217, 7500 AE, Enschede,
The Netherlands
guizzard@cs.utwente.nl

Abstract: Ontologies are becoming an important mechanism to build information systems. Nevertheless, there is still no systematic approach to support the design of such systems using tools that are common to information systems developers. In this paper, we propose an approach for deriving object frameworks from domain ontologies and then we show the application of this approach in the software process domain.

Keywords: Ontologies, frameworks, methodologies for using ontologies.

1. Introduction

An Information system cannot be written without a commitment to a model of the relevant world —commitments to entities, properties, and relations in that world. Data structures and procedures implicitly or explicitly make commitments to a domain ontology [1].

Several projects in AI have focused on using ontologies to promote knowledge sharing, and to substitute the usual database or object-oriented schema with an ontology, which offers a semantically richer model of the domain [2]. This trend has also acquired followers in the Software Engineering community. However, one of the major drawbacks to a wider use of ontologies in this area 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 technology, we claim that we need a systematic approach to derive object models from ontologies in order to put ontologies in practice. In this paper we propose a systematic approach to derive reusable object artifacts from domain ontologies. In section 2, we briefly discuss some aspects of ontology building, including a method and a graphical language, and a past experience using them. In section 3, we present a set-based formalism for ontology representation. In section 4, we describe our approach to derive object models and frameworks from domain ontologies, showing how it was applied in the software process domain. In section 5, related works are discussed. Finally, in section 6, we report our conclusions.

2.Ontologies

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 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 objects and relations [3].

According to Guarino [3], "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". Based on such definition, an ontology consists of concepts and relations, and their definitions, properties and constrains expressed as axioms. An ontology should not be only a hierarchy of terms, but a fully axiomatized theory about the domain [4].

In [4], we proposed a Graphical Language for Expressing Ontologies (LINGO) and a systematic approach for engineering ontologies. In the knowledge acquisition process, the use of a graphical representation is essential in order to facilitate the communication between requirement engineers and experts. In ontology building, such representation is basically a language representing a meta-ontology. Hence, this language has basic primitives to represent a domain conceptualization. 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, e.g. the *subtype-of* relation and the several types of *mereological* relations. In such cases, specialized notations have been proposed. In fact, 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.

Concerning the approach for building ontologies, the proposed method basically wraps the following activities: purpose identification and requirement specification, ontology capture and formalization, integrating existing ontologies, and ontology evaluation and documentation [4].

Both language and method have been used in the development of complex information systems in areas such as Software Process, Port Management, Steel Metallurgy, Medicine and Media on Demand Management. Although they have proven to be useful, we identify a great concern from the developers: how to put those ontologies in practice, viz., how ontologies can support actual software development?

In [5], we developed a software process ontology and used it to promote knowledge integration in a Software Engineering Environments (SEE). Since the SEE was implemented using objects, we have to derive an object model from the domain ontology. This represented a design problem that was informally solved. More recently, other developers have experienced the same problem. Based on these facts, the methodology presented in this paper has been proposed

3. A formalism for Ontology Representation

In session 2, we mentioned our previous experience developing a software process ontology. In this experience, we used first-order logic as the language to specify the axioms of the formal theory. First-order logic is widely known for its expressive power and its ontological neutrality, therefore adding minimal ontological commitments. However, due to the goals of this work, it is very convenient to adopt a formalism that lies in an intermediate abstraction level, between first-order logic and object-orientation. 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 and, at the same time, a natural option as a conceptual model for reasoning about objects. To clarify this point, the following example is used: 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 $x \in$ Mortal, i.e. to the set of all the elements of 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 of the considered world that share the same properties and the same definition.

Because of these characteristics of set theory, to build a model using the proposed set-based language is a very important step in a systematic translation between the logic and the object worlds. Moreover, the language preserves the expressive power of the first-order logic without adding significant ontological commitments, therefore, being suitable to play the same role in the axiomatization process. Although formal, the language is kept as simple as possible, defining only what is absolutely necessary to accomplish its goals.

Finally, it is essential to explain our decision for defining a new set-based formalism instead of using an existent one, for instance Z [17]? The reasons motivating this choice are both philosophical and practical. From a practical point of view, Z has a complicated mathematical notation that contains some language constructs that are unnecessary for ontology specification, e.g. the primitives for method specification. As a consequence of this, practitioners normally find these notations hard to use, mainly if their rather complex textual syntax is the only vehicle to produce specifications. From a philosophical point of view, the language primitives do not offer all the necessary means to express important ontological distinctions. An example of the latter is the fact that the language considers concept relations and properties as equivalent constructs. This consideration holds true from a mathematical perspective but not from an ontological one. In other words, we claim that Z's set of primitives is neither sufficient nor necessary for ontological formal representation. For a deep discussion about the ontological distinctions between concept relations, roles and properties please refer to [16].

In the next sub-session, the theoretical foundation for our formalism is briefly presented. It is also discussed how the primitives of this formalism are related to the LINGO building blocks.

3.1 - A theoretical foundation for a Set-based language

Sets are collections of zero or more elements whose members are unique and their order is immaterial. Sets can be finite or infinite. Finite sets with a small number of elements are usually represented by the enumeration of their members. Otherwise, they are represented by formation rules or by the definition of the characteristics and properties that all their members must have in common (intention). In our approach, concepts are defined as sets and, as mentioned before, the statement $x \in$ Mortal commits x to the concept Mortal, both intentionally and extensionally.

Another fundamental building block in the LINGO meta-ontology is the primitive *relation*. This primitive represents a semantic link that exists among a set of (one or more) 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 $R = (C_1, C_2 \ldots C_n, p(x_1, x_2 \ldots x_n))$, where each C_i represents a different set involved in the relation and $p(x_{i,})$ is a functional predicate open in *n* variables that maps each element from the cross-product $C_1 \times C_2 \times \ldots C_n$ in a *true* or *false* value. In this case, the set R^* (solution set) is the subset of $C_1 \times C_2 \times \ldots C_n$ whose members e_i all satisfy the predicate $p(e_i)$.

Figure 1 shows an example of a binary relation that links the concepts Person and Organization. The equivalent description in set theory is contract = ((Organization, Person, contract(x,y)). From now on, the propositional function p(x,y) will be used as synonym of the n-tuple that defines the relation, assuming that the function is defined in some cross-product $C_1 \times C_2 \times \ldots C_n$.

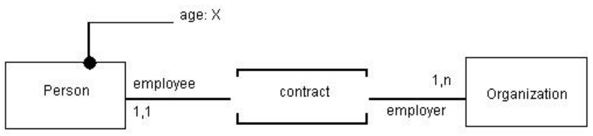

Fig.1 - Example of a binary relation

Figure1 also depicts two important modeling primitives: *properties* and *roles*. In the ontological level [16] there is a clear distinction between properties, roles and concepts. These distinctions are considered in our approach in a set of mapping directives discussed in section 4. However, at the structural level, the property age (of Person, fig. 1) represents an ordinary relation age(p,x) between instances of Person and instances of Natural numbers. Actually, this relation happens between the sets

Person and X, where X is a subset of IN, such that all its members represent the years of existence of another element involved in the age relation ($(X \subset IN) \wedge (\forall\ x{:}X,\ \exists y$ age(x,y))). In our formalism, the variable that represents an instance of a property is underlined, e.g. $\forall$ p:Pessoa, $\exists!\underline{a{:}X}$ age($\underline{a}$,p). Finally, in addition to represent characteristics of concepts, properties can characterize relations as well. This feature is discussed in [19].

In set theory, some essential operations are defined to express the relations between sets ($\subseteq$ - proper-subset or $\subset$ - subset; $\cup$ - Union; $\cap$ - Intersection; \ - set difference; $\wp$ - power set), properties of sets (**#** - cardinality**)**, restriction on relations (**~** - inverse relation, **;** - relation composition) and relations between sets and their members ($\in$ - Membership) [18]. 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 are defined. The most important among them is the function called **Image (Im).** This function plays a fundamental role in the specification of derivation axioms and solution sets to competency questions [6]. The definition of **Im** is given as follows:

$$\mathbf{Im(\_,\_)}: \wp(X) \times (X \leftrightarrow Y) \rightarrow \wp(Y)$$
$$\mathbf{Im(S,R)} = \{x{:}X, y{:}Y \mid (x \in S) \wedge ((x,y) \in R^*) \bullet y\}$$

The following axiom also holds for this function:

$$\forall\ a,b,R\ \ b \in Im(\{a\},R) \leftrightarrow a \in Im(\{b\},R)$$

Using the relation of figure 2 as an example, a possible valid image set could be: Im({Org1}, contract) = {John, Paul, Mary} and, consequently, Im({John},contract~) = {Org1}. It is important to notice that Im is a distributive function, i.e. Im({John,Mary},contract) = Im({John}, contract) $\cup$ Im({Mary}, contract).

In figure 2, cardinality constraints are used to specify the number of concept instances that can be involved in a relation. The cardinality (0,n) does not impose any restriction and, for that reason, its not graphically represented. Other cardinality possibilities include (0,1), (1,1) and (1,n). Whenever used, these cardinalities incorporate new axioms to the model. In figure 2, the cardinality (1,1) implies that $\forall$ p:Pessoa #Im({p},contract) = 1 and cardinality (1,n) implies that $\forall$ o:Organization #Im({o},contract) $\geq$ 1. Although the examples presented above represent only binary relations, the formalism used in this work is expressive enough to model relations of any arity. Likewise, reflective relations (relations between instances of the same concept) and conditional relations (AND and XOR tight relations) can also be represented [4].

4. From Conceptual Models to Computational Infrastructures – The Impedance Mismatch Problem

The problem of consistently generating computational infrastructures from conceptual models has been known for a long time by the software engineering community as the, so-called, *Impedance Mismatch Problem (IM)* [15]. In the scope of this work, the conceptual models are domain ontologies and the computational infrastructures are object-oriented frameworks. The use of domain ontologies to realize the domain analysis activity in a software engineering process contributes with innumerous advantages [19]. However, the impedance mismatch problem is amplified: instead of performing just one step to translate between two levels of abstraction (conceptual models to computational infrastructures), two steps are necessary. The first step is to translate from an ontological level model (domain axiomatized theory) to an epistemological conceptual model (conceptual view of class diagrams) without loosing the explicit representation of knowledge. The second step is the translation between the domain model to its computational concretization - an activity that, in domain engineering terms, is called *domain design*.

Our systematic approach to address this two-level IM problem is composed of a set of directives, design patterns and transformation rules. 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. 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 that is mapping the concept. Although this approach works well in most cases, it is worthwhile to point exceptions that we have found:

- some concepts can be better mapped to attributes of a class in an object model 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.

Furthermore, the directives consider non-trivial mappings, e.g. n-ary relations, relation properties and conditional relations. At last, they advise the choice between primitives to model a domain entity (Guarino discussion about sortals, temporal neutrality and ontological rigidity is a very good example of this [16]).

Besides the epistemological constructs, ontologies explicitly represent knowledge in a signification level through the use of formal axioms. These axioms can be of two types: *consolidation axioms* and *derivation axioms* [4]. 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. To guarantee the fulfillment of the constraints specified by consolidation axioms, we developed a general *precondition pattern*. Afterwards, this pattern is used to compose another pattern called *whole-part* *pattern* [19,20]. This was necessary because ontological whole-part relations are not well mapped to aggregations in an object model, i.e. UML notation for aggregation does not guarantee the constraints imposed by the theory (mereology) underlying the proposed notation [4].

For the sake of necessary brevity, the mapping directives and the consolidation axioms will no longer be discussed in this paper. In a future article, we will describe how the `Set` framework presented in the subsection 4.1 can help to solve most of the issues related to the IM problem. The bottom-up construction of our mereological pattern will be presented in yet another article. The emphasis of this paper is on how this framework and a collection of transformation rules can be applied together to generate consonant JAVA implementations for the derivation axioms.

4.1 - The `Set` Framework

The figure 2 shows a support framework that plays a fundamental role in our ontology-to-objects mapping. This framework implements the mathematical properties described by the theoretical foundation presented in section 3. The methods of the `Set` class are summarized in table 1.

The `Set` class is a generic container that is able to hold extension sets for all kinds of concept instances. To be accessible, each member of a set must have a unique identifier. The `SetElement` interface deals with these requirements, providing an identification mechanism through the `getKey` method. For an instance of any class to be held in a `Set`, it must implement the `SetElement` interface. Consequently, the `Set` class is actually a set of `SetElement` instances. The primary key for these elements is typed as Object, which is the top-most class in JAVA hierarchy. This is done in order to give the application classes total freedom regarding implementation decisions.

The framework also defines two other classes: `PersistentSet` and `MemberSet`, both sub-types of `Set`. The former is a set that is able to handle its permanent storage in total transparency from the perspective of the class users. When the `store()` method is invoked in a `PersistentSet`, the class performs the serialization of all its members. The original state of the objects (as well as their relations) can be afterwards restored by the invocation of the `retrieve`( ) method.

Finally, persistent sets can be used as an interesting alternative to implement databases [19]. Using this paradigm, a database can be seen as a **family** $\Im$ (set of sets), which contains all the sets existing in the application. Since, in this case, each set will be a member of another set, they must also be univocally identifiable. The `MemberSet` is, thus, provided to enable this situation.

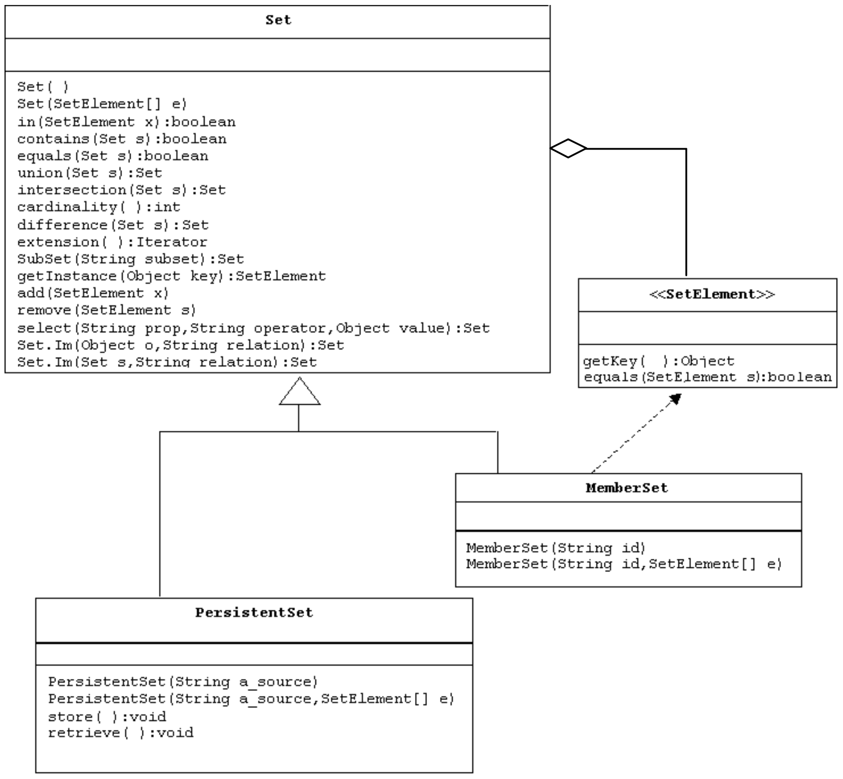

Fig.2 - *Framework* that implements the mathematical type Set

4.2 – Deriving object frameworks from domain ontologies

Figure 3 shows part of the LINGO model for a *resource-to-activity* allocation theory ontology [5]. The intention is to use it as an example over which we can explain our methodology. Therefore, not only the set of concepts and relations is incomplete and their intentions are not presented, but the fully ontology axiomatization will not be provided as well.

A software engineering process is a composition of a set of inter-related activities. Activities can be either of type construction, management or quality assurance. To make the realization of this process possible, a set of resources is allocated to be used in the activities inside the process. Resources could be of type software, hardware or human resources. Human resources can be managers, developers or project leaders, each one of them with specific skills. Each type of activity requires a special body of knowledge to be performed by a human resource. Thus, for instance, a management activity can only be performed by a Manager. Likewise, a construction activity can only be performed by a Developer. On the other hand, a Project Leader is an experienced human resource that is able to perform any kind of activity.

One central competency question in our example domain is: *for a given human resource allocated to the development of a process, what are the activities he/she can perform?*

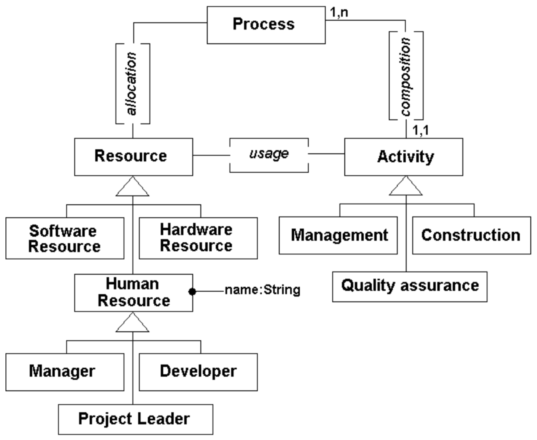

Fig.3 - Simplified ontology for a resource-activity allocation theory

The next subsections present the (partial) axiomatization of this ontology. The following notational convention is used: (C) - concept definition axioms, (R) - Relation definition axioms, (S) - specialization axioms, (CA) - cardinality axioms, (P) - axioms referring to properties and, finally, (O) ontological derivation axioms.

a) Definition Axioms

(C1) P = Process (C2) R = Resource (C3) A = Activity
(C4) H = Human Resources (C5) M = Management (C6) C = Construction
(C7) Q = Quality Assurance (C8) D = Developer
(C9) L = Project Leader (C10) G = Manager
(R1) allocation = (Resource, Process, allocation(r,p))
(CA1) ∀a:A #Im({a}, composition~) = 1
(R2) composition = (Process, Activity, composition(p,a))
(CA2) ∀p:P #Im({p}, posse) ≥ 1 (R3) usage = (Resource, Activity, usage(r,a))
(P1) ∀h:H ∃!n:String nome(n,h) (S1) M ⊂ A (S2) C ⊂ A (S3) Q ⊂ A
(S4) H ⊂ R (S5) T ⊂ H (S6) D ⊂ H (S7) G ⊂ H

As mentioned before, classes define a formation rule for its instance and, therefore, can be seen and manipulated as sets in a meta-level architecture. Consequently, the classification relations in the formalism do not require any specific implementations, i.e. relations such as $a \in A$, are totally resolved by the programming language typing mechanism through the creation of an object *a* of type *A*.

Like the classification relation, the sub-type-of relation does not require any additional implementation, i.e. sub-type-of relations among concepts can be directly mapped to generalization/specialization relations among classes.

For the *relation* mapping there are some issues that still must be discussed. In figure 2, a relation contract between the concepts Person and Organization is shown. In our approach, after this relation is translated to an object model, both correspondent classes have a method contract, named after the relation. In this case, the invocation of method `contract()` in an object o_1 of type `Organization`, it is possible to have access to all its hired employees (instances of `Person`). This resulting set is formally specified by the formula Im({o_1},contract)). Likewise, the method invocation in a person instance p_1 returns its contractor, or, Im({p_1},contract). The returned type of the relation methods depends straightly on the cardinality axioms associated to the relation. For instance, since in the scope of the contract relation a Organization may have several employees, contract is mapped to a `Set` variable in the `Organization` class and, hence, that is the type returned by the invocation of the synonymous method on this class. When a relation has a cardinality axiom imposing an inferior limit equals to 1, this constraint is reflected in the class constructors ensuring the establishment of the relation. As mentioned before, properties can be seen as a special kind of relation between sets and, ergo, the same rules are applied to them.

b) Ontological derivation axioms

(O1) $\forall$ l:L, a:A usage(l,a) $\leftrightarrow$ a $\in$ Im(Im({l},allocation),composition)
(O2) $\forall$g:G, a:A usage(g,a) $\leftrightarrow$ a $\in$ (Im(Im({g},allocation),composition) $\cap$ M)
(O3) $\forall$d:D, a:A usage(d,a) $\leftrightarrow$ a $\in$ (Im(Im({d}, allocation),composition) $\cap$ C)

The derivation axioms are formalized to answer to the competency questions of the ontology. The axioms above, for instance, answer to the following question: *for a given human resource* h_1, *what are the activities he/she could be allocated to?* The solution set for this question must be returned by the invocation of the method `usage()` in an object `h1` of the `HumanResource` class. However, for this type of methods to be derived from derivation axioms, a set of transformation rules must be defined. The set of rules presented as follows is a subset of our complete set of transformation rules. For a complete set of transformation rules, please refer to [19,20].

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

This rule states: if for each instance x of type X, x is engaged with all instances y from set C (and only instances of this set) in a relation r_1, the set returned by the function Im(x, r_1) will be exactly C. 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`.

T2: Im({x}, r_1) $\Rightarrow$ x.r_1()

As mentioned before, a relation (role or property) r_1 between two concepts X and Y is mapped in the classes that represent these concepts to methods named after the relation. For instance, given an instance x, the invocation x.r_1() returns the set of objects from Y associated to x in the relation r_1.

T6: Im(A, r_1) $\Rightarrow$ Set.Im(A," r_1")

The rules **T6** promotes the replacement of the mathematical function *Image* by the corresponding syntaxes through which they are implemented in the `Set` class.

```
T7: x.r1():Y ≡ C ⇒  public class X
                    {    public Y r1( )
                         {
                              return C;
                         }
                    }
```

Finally, rule **T7** directly translates the axiom written in its left side to the implementation correspondent syntax in the chosen programming language. All the references to the instance x existent 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 will be made.

The code fragment below shows the derivation process for the axiom O1, and also its implementation in the `ProjectLeader` class.

```
(O1) ∀ l:L,  a:A usage(l,a)  ↔ a ∈  Im(Im({l},allocation),composition)
1. Im(l,usage):Set ≡ Im(Im({l}, allocation),composition)            O1, T0
2. l.usage():Set ≡ Im({l}.allocation()),composition)                  1, T2
3. l.usage():Set ≡ Set.Im({l}.allocation(),"composition")             2, T6
4. public class ProjectLeader                                         3, T7
   {
      public Set usage()
      {
          return Set.Im(this.allocation(),"composition");
      }
   }
```

```
public class ProjectLeader extends HumanResource
{
  ProjectLeader(String name) { super(name); }
  public Set usage()
  {
    return Set.Im(this.allocation(),"composition");
  }
}
```

5. Related Work

The Peirce project is an international collaborative effort to build a conceptual graph workbench [7][8]. To accomplish interoperation among the different tools produced in the context of the project, a mathematical ontology was proposed and a software library was derived. The ontology contains taxonomic hierarchies for mathematical objects such as: sets, groups, categories, relations, functions, preorders, partial orders and lattices. In [7] a specification for a Set class is formalized in several languages (Z, KIF, Conceptual Graphs) and a set of C++ contracts is derived, showing pre/posconditions for the operations of the type. In [7], it is also presented a way to implement conceptual graphs primitives (such as concepts and relations) in C++ templates. However, due to the focus of these projects the emphasis is on the object-oriented implementation of a CG processor and not in how to create object-oriented artifacts from a conceptual model. Recent research in this field include the Notio Java API [9], always focusing on the object-oriented implementation of the meta-models (CG in this case) and not on the models themselves.

Another very interesting approach to address the impedance mismatch between the ontology and object-oriented abstraction levels is through the use of design patterns. In [12] a set of design patterns for constraint representation in JavaBeans components is presented and computation reflection mechanisms is used to evaluate these constraints at run-rime. Likewise, in [13], three design patterns are used promote JAVA implementation for ontologies - represented in the OKBC knowledge model [14]. In this case, ontology concepts are either represented by reflection-backed JavaBeans classes, by an Active Object-Model (AOM), or by a mixed approach based on extending the classes from the AOM.

Constraints are equivalent to, what we call, consolidation axioms. These axioms represent only a subset of the knowledge that must be made explicit in the ontological level. Constraints basically define preconditions that must be satisfied for a relation to be consistently established. As mentioned before, our approach to implement these axioms is also based on ontological design patterns. This approach will be discussed in a future article. In this paper, because of the limited amount of space, we chose to focus on another set of ontological axioms: derivation axioms.

Finally, in [10] and [11], it is presented an approach to create object models such as CORBA IDLs and Java classes and interfaces from Geographic Information Systems (GIS) Ontologies. The papers suggest the automatic generation of interfaces and IDLs from Ontolingua models. These interfaces constitute ontology skeletons that are, afterwards, complemented by implementation code written in Java. Ontology editors, such as Ontolingua, have the ability to create CORBA IDL headers automatically, however in this case, the behavior implementation for the interface methods would still rely on an ad-hoc translation process. Moreover, interfaces alone are not expressive enough to incorporate the knowledge related to all kinds of consolidation axioms, let alone, ontological derivation axioms.

6. Conclusions

Since Aristotle's theory of substance (objects, things and persons) and accidents (qualities, events and process) ontologies have been used in philosophy as a foundation for representing theories and models of reality. Their main purpose is to formally make explicit the semantic distinctions existent in portion of the world, accounted as a domain. Hayes [21] introduced the use of ontologies in Computer Science (more specifically in Artificial Intelligence). Since then, they have been employed in areas such as computational linguistics, knowledge engineering, information integration and multi-agent systems. In addition to that, they have been used in application areas such as enterprise modeling [6] and GIS [11], among several other examples.

In the software engineering realm, domain ontologies have been used to model the foundation over which meta-enviroments can be constructed [5]. Moreover, they can add important contributions to the domain engineering phase, promoting a reuse-based practice in the requirements engineering level [19].

Nevertheless, few of the ontology construction methodologies lead to executable code and, there was still no systematic approach to fully promote their integration to the object-oriented software development practice. For this reason, most of the object-oriented implementations of domain ontologies rely on informal derivation processes.

In this paper a contribution to address this problem is presented: a methodology through which object-oriented frameworks can be systematically derived from domain ontologies. To accomplish this goal, we also proposed a construction method and a formal representation language. The mathematical foundation of the language (set-theory) highly contributed to the feasibility of our approach. This is mainly due to its suitability to bridge the conceptual and implementation abstraction levels, respectively represented by first-order logic axioms and object models.

The derivation methodology proposed comprises a spectrum of techniques, namely, directives, ontological design patterns and transformation rules. This paper focused on the latter, showing how these rules together with the supporting `Set` framework can establish a sound path between our formally axiomatized theories and a related consonant implementation in JAVA classes. The other two techniques will be addressed in future articles.

We use the *resource-to-activity* allocation problem in software processes as an example, over which the methodology is presented. The ontology presented was over-simplified due to the lack of space. In despite of that, the methodology has been tested in several case studies, ranging from software process to video on demand management theories. In all these experiences, it was found very effective, mainly because of: (i) its ability to capture the domain knowledge without imposing additional ontological commitments; (ii) its ability to successfully derive object frameworks capable of answering the relevant competency questions.

As it can be seen, our methodology is highly focused on the structural part of domain ontologies, consequently, a natural extension of this work, is to study how a similar approach can be developed to address the dynamics aspects of domains, i.e. behavior ontologies.

References

[1] Chandrasekaran, B., et al., *"What are Ontologies, and Why Do We Need Them?"*, IEEE Intelligent Systems, pp. 20-26, January/February 1999.

[2] Valente, A., et al., *"Building and (Re)Using an Ontology of Air Campaign Planning"*, IEEE Intelligent Systems, pp. 27-36, January/February 1999.

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

[4] Falbo, R.A., et al.; *"A Systematic Approach for Building Ontologies"*. Proceedings of the IBERAMIA'98, Lisbon, Portugal, 1998.

[5] Falbo, R.A., et al.; *“Using Ontologies to Improve Knowledge Integration in Software Engineering Environments”*, Proceedings of SCI’98/ISAS’98, Orlando, USA, July, 1998.

[6] Gruninger, M., and Fox, M.S., *"The Role of Competency Questions in Enterprise Engineering", Proceedings of the IFIP WG5.7 Workshop on Benchmarking - Theory and Practice,* Trondheim, Norway, 1994.

[7] Ellis G; Callaghan S; *"A specification of a Set Class in Peirce"*, online: http://citeseer.nj.nec.com/29926.html, Oct, 1995.

[8] Munday C.; Lukose D; *"Object-Oriented Design of Conceptual Graph Processor"*, Proceedings of the Fourth International Workshop on Peirce: A Conceptual Graph Workbench University of Maryland, Maryland, USA, 1994.

[9] Southey, F. and Linders, J. G., *"Notio - A Java API for Conceptual Graphs"*, in Proc. of the 7th International Conference on Conceptual Structures (ICCS'99), Springer-Verlag, 1999

[10] Fonseca, F. Egenhofer M. *"Knowledge Sharing in Geographic Information System"*, In: P. Scheuerman, (Ed.) The Third IEEE International Knowledge and Data Engineering Exchange Workshop, Chicago, 1999.

[11] Fonseca F. T. et al. *"Ontologies and Knowledge Sharing in Urban GIS"*, CEUS- Computer,Environment and Urban Systems, 2000.

[12] Knublauch H.; Sedlmayr M.; Rose T., *"Design Patterns for the Implementation of Constraints on JavaBeans"*, NetObjectDays2000,Erfurt, Germany, 2000.

[13] Knublauch H., *"Three Patterns for the Implementation of Ontologies in Java "*, OOPSLA'99 Metadata and Active Object-Model Pattern Mining Workshop, Denver, CO, USA, 1999.

[14] Grosso W. et al. *"Knowledge Modeling at the Milennium (The Design and Evolution of Protégé-2000)"*, Knowledge Aquisition Workshop, Banff, Canada, 1999.

[15] Woodfield S.N. *"The impedance Mismatch between Conceptual Models and Implementation Environments"*, International Conference on Conceptual Modeling (ER'97), Workshop on Behavioral Models and Design Transformations, UCLA, Los Angeles, California, Nov, 1997.

[16] Guarino N. *"The Ontological Level"*. In R. Casati, B. Smith and G. White (eds.), Philosophy and the Cognitive Sciences, Vienna, Hölder-Pichler-Tempsky 1994.

[17] Spivey, J. M. *"Understanding Z: A specification language and its formal semantics"*, Cambridge University Press, 1988.

[18] Roitman J. *"Introduction to modern set theory"*, Wiley-Interscience, New York, 1990.

[19] Guizzardi, G. *"A methodological approach for reuse-oriented software development, based on formal domain ontologies"* (in portuguese), Federal University of Espírito Santo, Master Thesis, 2000.

[20] Guizzardi, G.; Falbo, R.; Gonçalves J. *"Using Framewors and Patterns to Implement Domain Ontologies"*, Proceedings of the XVI Brazilian Symposium on Software Engineering, Rio de Janeiro, Brazil, Oct, 2001.

[21] Hayes P. "The Naive Physics Manifesto", Expert Systems in Microeletronics age", D. Ritchie Ed., Edinburgh University Press, 1978, pp 242-270.