An Operational Approach for Capturing and Tracing the Ontology Development Process

Marcela Vegetti[1], Luciana Roldán[1], Silvio Gonnet[1], Gabriela Henning[2] and Horacio Leone[1]

[1] Ingar (CONICET/UTN)
Avellaneda 3657 – S3002GJC – Santa Fe – Santa Fe – Argentina

[2] Intec (CONICET/UNL)
Güemes 3450 – S3000GLN– Santa Fe – Santa Fe – Argentina

{mvegetti,lroldan,sgonnet,hleone}@santafe-conicet.gov.ar,
ghenning@intec.unl.edu.ar

Abstract. *The history of an ontology development project, including its intermediate products, together with the executed activities, and the decisions made, might be of great importance in other future ontology developments. However, current tools supporting this kind of projects do not capture such information; thus, the process trace is lost, and any new ontology development project would start from scratch. This paper presents a framework meant to do overcome these deficiencies, allowing the capture and trace of such projects.*

1. Introduction

Until the mid-90s, ontologies were developed without addressing systematic procedures. Therefore, the ontology development process was an art rather than an engineering activity [Fernández-López et al., 1999]. In the last decade, many ontology development processes have changed from the traditional ones, performed by isolated knowledge engineers or domain experts, into collaborative processes executed by mixed teams [Bernaras et al. 1996]. In such teams, experts in knowledge acquisition and modeling, domain specialists, and experts in implementation languages collaborate to build ontologies, according to well-established methodologies. The expertise of each team member, as well as the executed activities, and the decisions made during the development process might be of great importance in future projects. However, current tools supporting ontology development processes do not capture such information; thus, the process trace is lost, and any new project would start from scratch. In fact, once a given ontology development process is finished, the things that remain are mainly isolated design products (e.g., requirement specifications, competency questions, class diagrams, specific language implementations, etc.), without an explicit representation of how these products were obtained, and with no capture of the rationale behind the process. In addition, ontology building is turning into a more professional engineering activity that needs to be managed and measured in order to obtain high quality results; and such management requires an explicit representation of the development process. The issues pointed out before constitute essential challenges that need to be addressed.

In order to tackle them, this contribution proposes ONTOTracED, a framework to represent, capture and trace ontology development processes. This paper is organized

as follows: after discussing some issues about ontology development processes in Section 2, the framework components are presented in detail in Section 3. Finally, Section 4 concludes the paper and offers paths to future work.

2. Ontology development processes

Ontology Engineering (OE) is a relatively new field concerning ontology development processes, the ontology life cycle, the methods and methodologies for building ontologies, and the tool suites and languages that support them. A series of methodologies have been reported in the literature in the last two decades. An extensive state-of-the-art overview of these methodologies can be found in Gómez-Pérez et al. (2004). In addition, Cristani and Cuel (2005) have proposed a framework to compare ontology engineering methodologies and evaluated the established ones accordingly. The first contributions in the field, which are due to several authors [Gruber 1993], [Grüninger and Fox 1995], [Uschold et al. 1998], [Uschold and Gruninger 1996], set the grounds for many subsequent proposals. Gruber's work [Gruber 1993] discussed some basic ontology design criteria associated with the quality of the developed ontology, as well as related to the methodology used to build it. Gruninger and Fox (1995) provided a building methodology based on Competency Questions. Methontology [Fernández-López et al. 1999] which is an ontology development process, proposed an ontology lifecycle based on evolving prototypes and specific techniques to address each activity of the approach. With emphasis on knowledge management, Staab et al. (2001) proposed On-To-Knowledge. Other approaches, related to industry or research projects, include the methods used for building CyC, SENSUS [Swartout et al. 1997] and Neon [Suárez-Figueroa et al. 2012]. These works report different principles, design criteria, and stages of the development process. However, no one is yet emerging as a clear reference [De Nicola et al. 2009]. Despite recent advances, there are few computational tools supporting the above mentioned methodologies. Neon Toolkit supports the Neon methodology and allows scheduling the stages that will be included in the design of a specific ontology. However, such tool neither captures the operations actually executed when adding a concept, a relation among concepts, etc., nor the rationale behind such operations. Consequently, there is still room for improvement in the OE field.

3. A framework to capture and trace the ontology development process

Generally, at the end of an ontology development process the things that remain are mainly unconnected design products (e.g. the requirements specification, competency questions, ontology class diagrams, the ontology implementation in a specific language, etc.), without an explicit representation of how they were obtained, and with no capture of the history and rationale behind the project. More specifically, there is no trace of the activities that have led to any of the products, the requirements imposed at each stage of the process, the actors that have performed each of the activities, and the underlying rationale behind each decision that was made. To overcome these weaknesses, this work proposes a comprehensive framework to represent, capture and trace the ontology development process, along with its associated products and their evolution.

Fig. 1 shows the main components of the proposed framework, that includes: (i) a *Conceptual Model*, which is able to represent generic design processes; (ii) an *Ontological Engineering Domain Model (OEDM)* that specifies the concepts that are

required to describe ontology development processes, and (iii) a support computational environment, named *TracOED (Tracking Ontology Engineering Designs)*, that implements both the conceptual model and the OEDM to enable the capture of specific ontology design processes, along with their associated products.

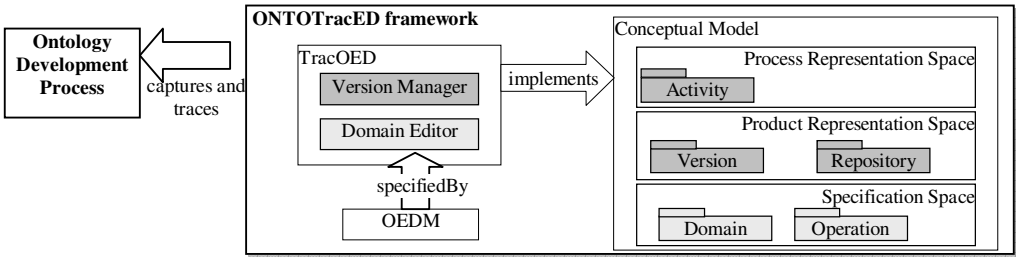

Figure 1. Components of the proposed framework.

The supporting *Conceptual Model* is based on an operational-oriented approach that envisions the ontology development project as a sequence of activities that operates on the products of the development process. The proposal defines two representation spaces to model generic design process concepts: the *Process* and *Product* spaces. In addition, a third component (the *Specification Space* in Fig. 1) is included to fully specify a flexible model that is able to represent and capture design processes pertaining to specific engineering fields.

The *Ontological Engineering Domain Model* component can represent and capture particular ontology development projects, based on building-blocks that define the products obtained, as well as the activities carried out during this type of processes. This representation includes those modeling elements that are most commonly used in the methodologies that nowadays guide ontology development processes. Among these modeling elements are: the competency question, concept, and relation concepts, etc. In order to show how this proposal may be applied when ontologists want to stick to specific methodologies and/or approaches, the ontological categories proposed by the Unified Foundational Ontology (UFO) [Guizzardi 2005] have been added to the *Ontological Engineering Domain Model*. UFO is a language to build domain ontologies that preserves the ontological commitment of the domain being modeled. It distinguishes between conceptual entities called *universals* and *individuals*. In particular, due to space limitations, this work focuses on the subsumption hierarchy of sortal universals.

TracOED is the computational environment that implements the conceptual model and incorporates the OEDM. It is based on TracED [Roldán et al. 2010], which was conceived for capturing and tracing engineering designs. The major components of TracOED are the *Domain Editor* and *Versions Manager*. By using the *Domain Editor*, the OEDM has been specified in *TracOED*. Furthermore, the editor allows this model to be further specialized, if required. On the other hand, the *Versions Manager* keeps track of the execution of a design project, as will be shown in the following sections.

3.1 Conceptual Model

The *Conceptual Model* component provides the framework foundations. This component is organized in *Process Representation*, *Product Representation* and *Specification spaces*, which are explained in this section. The *Process representation*

space models the activities being performed during an ontology development process and it is specified by the *Activity* package (Fig. 2). In particular, when tackling the development of an ontology, typical tasks are: adding concepts and relations into the ontology, defining constraints on a specific concept, analyzing whether a group of concepts, relationships and constraints satisfies a formal competency question, evaluating the ontology, deciding on alternative concepts and relations, etc. As Fig. 2 shows, such activities are represented in the model with the *BasicActivity* or *CompositeActivity* classes, depending on whether the task is atomic or it can be decomposed into a set of subactivities.

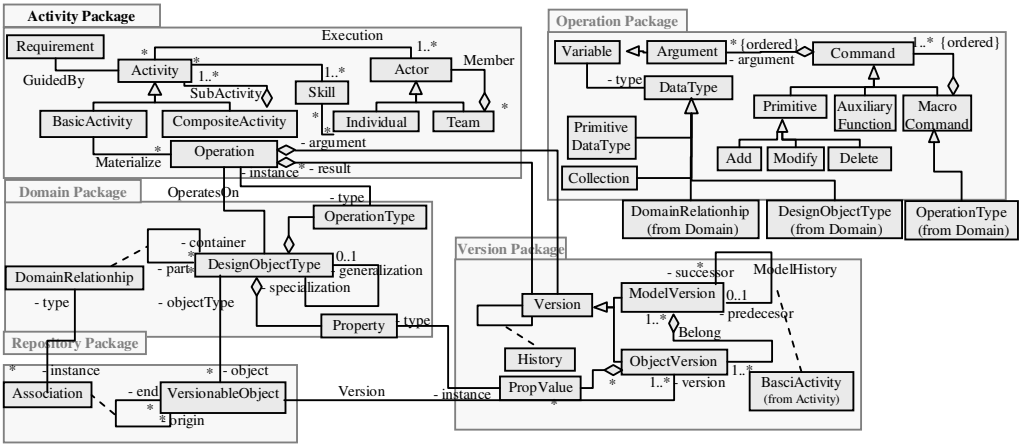

Figure 2. Conceptual Model.

In the proposed model, the execution of an activity is guided by one or more *requirements,* which specify the functional and non-functional characteristics that a development product must satisfy (e.g., in the ontology development domain, the concepts have to preserve the ontological commitment of the domain being modeled). Therefore, the ontology development process is interpreted as a series of activities led by *requirements* that are performed by *Actors.* An *Actor* may be either an *Individual* (a human being or a computational program) or a *Team.* Teams allow representing composite skills that are needed for carrying out activities. Each basic activity performed by an actor during an ontology development process is represented by the execution of a sequence of *operations*, which transforms the design objects. The operations that can be applied are domain dependent. So, it is necessary to define the allowed types of operations, as well as the modeling elements, for each specific domain.

As it was previously introduced, activities operate on the outcomes or products of the ontology design process, called *design objects* (Fig. 2). Design objects represent the various products of the development activities. Typical design objects are models of the artifact being conceived (e.g., in the ontology development domain: class diagrams, implementations in specific ontology languages, etc.), specifications to be met (i.e. competency questions, quality attributes, etc.). Design Objects may relate among themselves by domain specific relationships (*DomainRelationship* association class in Fig. 2), and can be organized in generalization-specialization hierarchies. Design objects types are described by a set of properties. Moreover, each design object type is related to a set of operation types that may be used to transform such design object.

In this proposal, the execution of an activity (materialized through a sequence of operations) transforms a design object, which thus may evolve into multiple versions. In order to represent this evolution, each *design object* is specified at two levels: the *Repository* and the *Version* packages (Fig. 2), which constitute the *Product Representation Space*. The *Repository* keeps a unique entity for each *design object* that has been created and/or modified due to the natural progress that takes place during a development project. Any entity kept in the repository is regarded as a *versionable object*. Furthermore, relationships among the different versionable objects are also maintained in the repository (*Association* class in Fig. 2). On the other hand, the *Version* level keeps the different versions resulting from the evolution of each design object, which are called *object versions*. The relationship between a *versionable object* and any of its *object versions* is captured by the *Version* association. Therefore, for a given design object, a unique instance is kept in the repository, and all the versions it assumes along the design process belong to the versions level. Fig. 2 also includes the *Design object type* class, which allows representing the various kinds of modeling elements pertaining to particular domains.

The versions package also includes the *ModelVersion* concept, which represents a set of design objects within the context in which a given design activity is carried out. Its aim is to provide a snapshot description of the state of a certain design process at a given moment. According to the proposed representation, a new *model version* m_n is generated when a *basic activity* is executed. Since each *basic activity* is *materialized* by a sequence of operations, named ϕ, the new *model version* m_n is the result of applying such sequence to the components of the previous *model version* m_p. This *predecessor model version* m_p corresponds to the context where the activity was performed and the successor one (m_n) represents the resulting context.

The *Specification Space* is defined by the *Domain* and *Operation* packages (see Fig. 2), which allow specifying the building blocks and operations of particular engineering design domains. In the context of the OntoTracED framework, this space has allowed specifying the ontological engineering domain model. The *Operation* package enables the specification of operation types and their implementations in a computational environment (TracOED in this case). This package defines the primitive operations *add*, *delete* and *modify* and also enables the specification of other operations that are applicable into the specific design domains (the ontology development domain in this work). When an *operation* is specified, it is necessary to define both its *arguments* and *body*. The *body* is comprised by some already defined commands that are available for being used in other operation specifications. They can be primitive (such as *add*, *delete*, or *modify*), *auxiliary function* commands, or previously defined operations.

3.2 Ontological Engineering Domain Model

As it was mentioned in section 3.1, the *Domain* and *Operation* packages (Specification space) of the underlying conceptual model let specify modeling elements and operations that are suitable for particular domains. This section presents the use of these packages in the specification of the *Ontological Engineering Domain Model*. Figure 3 (part a) presents a partial view of the resulting model.

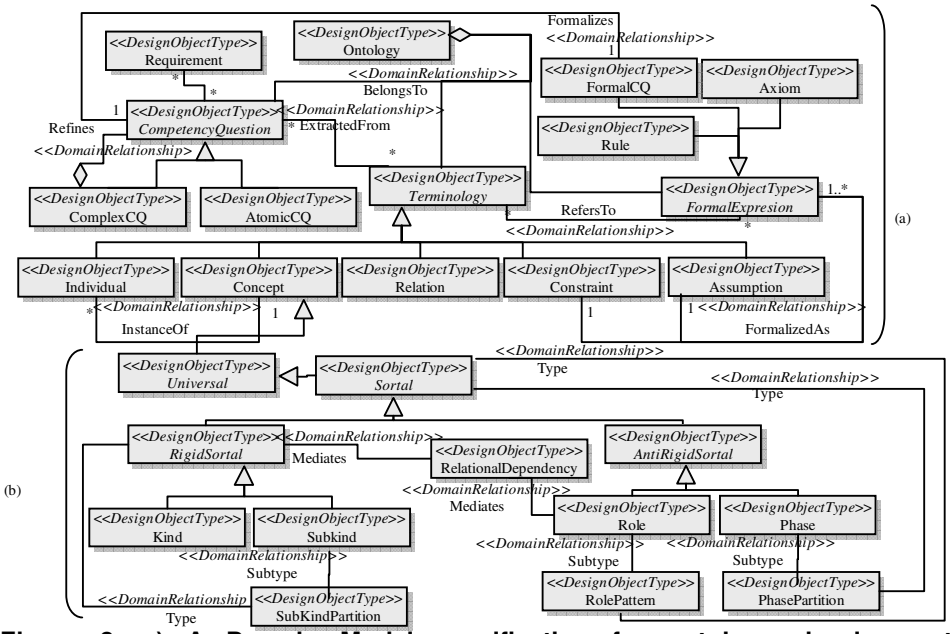

Figure 3. a) A Domain Model specification for ontology development processes. b) Design objects proposed for the development of ontologies using UFO.

There are several methodologies for building ontologies and no one is yet emerging as a clear reference. In spite of their diversity, most methodologies share structural similarities and have comparable modeling elements. In this proposal, the following components are considered to be part of the proposed domain model:

- **Competency questions** play the role of a type of requirement specification against which a given ontology can be evaluated [Gómez-Pérez et al. 2004]. They can be split off into more specific ones (*AtomicCQ* in Fig. 3), and complex competency questions (*ComplexCQ* in Fig. 3), which can be expressed in terms of simpler ones. Competency questions participate in most methodologies and they are the starting point in the identification of the ontology terminology.
- **Concepts** represent a collection of entities that share a common set of characteristics. Certain languages call them classes or frames. Concepts can be hierarchically organized by means of subsumption relationships.
- **Relations** symbolize interrelations between classes. Different languages call them properties, slots, roles, or associations.
- **Individuals** are entities that belong to a particular class. They are also called instances or members of such class.
- **Assumption** and **Constraints** represent natural language expressions that restrict the interpretation of concepts and relationships.

It is possible to distinguish between ontologies that are mainly taxonomies from the ones that model the domain in a deeper and formal way and provide more restrictions on the domain semantics. In order to represent this type of formalization it is necessary to incorporate additional design objects and operations. Therefore, the following elements have been added into the domain model:

- **Formal Competency Questions** are specification in a formal language of informal competency questions that were initially identified.
- **Axioms and rules** represent formal expressions that allow ontologists to (i) explicitly define the semantics of an ontological concept by imposing constraints on its value and/or its interactions with other concepts; (ii) verify the consistency of the knowledge represented in the ontology, and/or (iii) infer new knowledge from the explicitly stated facts.

Fig. 4 presents the functional specifications of some of the operations included in the OEDM. They give an outline of how these operations can be stated in the computational environment. From an implementation point of view, these specifications are instances of the entities defined in *Operation* Package (Fig. 2).

```
addConcept(o,cname)
   cversion:= add(cname,Concept)
   addRelationship(o,cversion, BelongsTo)
end

addInformalCQ(o,ICQname,exp)
  icqversion:= add(ICQname,AtomicCQ)
  modify(icqversion,exp)
  addRelationship(o,icqversion,BelongsTo)
end

addFormalCQ(o,exp)
  CQversion:= add(exp, FormalCQ)
  addRelationship(o,CQversion,BelongsTo)
end

toKind(o,cversion)
 n:= getname(cversion)
 kversion:=addKind(n)
 addRelationship(o,kversion,BelongsTo)
 delete(cversion)
end

addRole(o,rname,relDep,aSortal)
 rversion:= add(rname,Role)
 addRelationship(o, rversion,BelongsTo)
 rdvers:= add(relDep,RelationalDependecy)
 addRelationship(o,rdvers,BelongsTo)
 addRelationship(rversion,rdvers,Mediates)
 addRelationship(rdvers,aSortal,Mediates)
end

addSubKind(o,skname)
  skversion:=add(skname,SubKind)
  addRelationship(o,skversion,BelongsTo)
end
```

```
deriveConcept(o, cqversion,l_con)
  for each cname in l_con
       cversion:= addConcept(o,cname)
       addRelationship(cqversion,cversion,
                       ExtractedFrom)
  end for
end

formalizeCQ(ICQversion, fexp)
  o:= get(ICQversion,Ontology)
  f:= addFormalCQ(o, fexp)
  addRelationship(ICQversion,f, Formalizes)
end

applyRolePattern(o,pname,c,rname,rel,sv)
 rpversion:= add(pname,RolePatern)
 addRelationship(o,rpversion,BelongsTo)
 tversion:= type?(c)
 addRelationship(rpversion,tversion,Type)
 rversion:= addRole(o, rname,rel,sv)
 addRelationship(rpversion,rversion,Subtype)
end

applyPhasePartition(o,pname, kversion, l_con)
 ppversion:= add(pName,PhasePartition)
 addRelationship(o,ppversion,BelongsTo)
 addRelationship(ppversion,kversion,Type)
 for each cname in l_con
       phversion:= addPhase(cname)
   addRelationship(phversion,ppversion,Subtype)
end for
end
```

Figure 4. Specification of some operations belonging to the proposed model.

Fig. 4 shows some simple operations (*addConcept*, *addInformalCQ*, *addFormalCQ*) that allow adding design objects while developing a given ontology. It also shows that more complex ontological operations can as well be implemented. This is the case of the operation *formalizeCQ*, which allows formalizing a competency question, and the *deriveConcept* one. In particular, the *deriveConcept* operation allows adding into an ontology a list of new concepts that are identified from an informal competency question. The competency question object version (*cqversion*) and the list of concepts to be added (l_{con}), are the input parameters of this operation. As seen, all the proposed operations are defined in terms of primitive ones (*add*, *modify*, *delete*), auxiliary functions (*getDescription*, *getOntology*, *attachAffectedTerm*, among others), and/or operations (*addFormalCQ*, *addRelationship*).

As it was previously mentioned, the proposed OEDM defines design objects and operations to be able to handle the UFO ontological categories during the development of an ontology. Fig. 3 (part b) introduces a partial view of the resulting domain model

showing these new design objects. Table 1 presents the meanings of the concrete object types *Kind*, *SubKind*, *Phase* and *Role* and the list of applicable operations.

UFO is considered as a Pattern Language; i.e., in this language the choice of a particular design object type causes a whole pattern to be manifested [Guizzardi et al. 2011]. For example, a phase is always defined as part of a partition; a role is always played in relation to another sortal. Therefore, the adopted domain model also includes the following design patterns proposed by UFO: *SubKindPartition*, *PhasePartition* and *RolePattern* [Guizzardi et al. 2011].

Table 1. UFO Sortal Universals. Adapted from Guizzardi (2005)

UFO Ontological Categories	
Kind	A *Kind* represents rigid, relationally independent object universals that supply a principle of identity for their instances. Examples include instances of Natural Kinds (such as Person, Dog, Tree) and of artifacts (Chair, Car, Television).
SubKind	A *SubKind* is a rigid, relationally independent restriction of a substance sortal that carries the principle of identity supplied by it. An example could be the SubKind MalePerson of the Kind Person.
Phase	A *Phase* represents anti-rigid and relationally independent universals defined as part of a partition of a sortal. For instance, [Child, Teenager, Adult] is a partition of the kind Person. A Phase is always defined as part of a partition.
Role	A *Role* represents an anti-rigid and relationally dependent universal. For instance, the role student is played by an instance of the kind Person.

Proposed Operations			
Basic		**Pattern related**	
addKind	toRole	addPhasePartition	addPhase2Partition
addSubKind	toPhase	addRolePattern	addSubkind2Partition
addPhase	remKind	addSubkindPartition	remPhaseFromPartition
addRole	remSubKind	remPhasePartition	remRoleFromPartition
toKind	remPhase	remRolePattern	remSubkindFromPartition
toSubKind	remRole	remSubkindPartition	

Table 1 also presents the operations required to capture and manage the UFO-related design objects (Fig. 3 part b). It includes two groups of operations: basic ones, which comprise operations to add, delete or modify simple design objects, and pattern-related ones. These last operations are associated with the addition of the new set of design objects that follows the application of a given UFO pattern. Fig. 4 also presents the functional specification of some of these operations. As seen, the *toKind(o, cversion)* operation adds into a given ontology (*o*) a *Kind* design object (*kversion*), which is a refinement of a previously included concept (*cversion*). This operation also deletes the *cversion* concept from the current model version. Similarly, the *addRole* and *addSubKind* operations allow adding the *Role* and *SubKind* design objects to a given ontology *o*. Fig. 4 also presents the *applyRolePattern* and apply *PhasePartition* operations, which add a Role pattern and a *Phase* partition into a certain ontology, respectively. The rest of the operations are defined in a similar way by means of primitive operations (*Primitive* in Fig. 2), such as *add(skname,SubKind)*, and non-primitive ones, like *addPhase(cname)*.

3.3. TracOED

TracOED is the computational environment that implements the conceptual model and incorporates the OEDM, thus materializing the ONTOTracED framework. In order to

illustrate its features a case study is presented in this section. It is based on the development of the well known travel ontology.

As already mentioned, the *Versions Manager* enables the execution of each ontology development project, and captures its evolution based on *operations* that are accomplished and the instantiation of those *design object types* that have been specified in the Ontological Engineering Domain Model by means of the *Domain Editor* tool.

The development of the ontology starts with the definition of competency questions from which the requirements of the ontology and some initial concepts are identified. For instance, from the CQ1 competency question, which is shown below, one of the ontologists recognized the concepts *Person*, *Traveler* and *Destination*, among others.

CQ1: *Given the preferences of a traveler, the age and some constraints (economical or about the travel itself), which destinations are the most suitable?*

The identification of all the concepts from suitable competency questions marks the end of the first stage of the ontology development process. In the following stage the ontologist has to assign UFO ontological categories to the identified concepts, as well as he/she has to define new concepts falling into these categories. In this case study, the ontologist working on this part of the project considered that each of the *Person* and *Destination* concepts should be represented as a *Kind*. This decision caused the creation of a new ontology version where the *Person* and *Destination* concepts were replaced by their corresponding kinds. In addition, during this stage the ontologist gathered more domain knowledge, which allowed him/her to specify the ontology in more detail. In particular, he/she identified that a person plays the role of *Traveler* related to a *Travel Agency*. Moreover, considering the age of travelers, the involved ontologist distinguished among young, adults and old travelers. Therefore, he/she applied a phase pattern to represent this situation.

Fig. 5 presents a schema that exemplifies how the development process is captured by the *Version Manager*. The upper part of Fig. 5 shows the two ontology versions that were described above and that are inferred from the captured knowledge. In fact, the project evolves from a *Root Model Version*, which is empty, to *Model Version1* by applying the $\phi 1$ sequence of operations, which in turn is captured by the tool from the operations that were performed by the ontologist during the first stage of the process (definition of competency questions and derivation of concepts from them). Then, the evolution from *ModelVersion1* to *ModelVersion2* is caused by the operations included in $\phi 2$. These operations capture the activities carried out by the ontology developer when he/she applied the role and phase partition UFO patterns.

The first operations sequence, ϕ_1, includes the *addOntoloy, addInformalCQ* and *deriveConcept* operations that are responsible for creating the *CQ1*, *cPerson and cDestination* versionable objects at the repository level, and their first corresponding object versions ($CQ1_{v1}$, $cPerson_{v1}$ *and* $cDestination_{v1}$) at the version level. In turn, ϕ_2 comprises the *tokind, addSubKind, applyRolePattern* and *applyPhase Partition* operations. The execution of these operations has the following impact in *ModelVersion2*: (i) the addition of *kPerson* (*Kind*), (ii) the incorporation of a *RolePattern*, which comprises *kPerson*, *TravelAgency* (*SubKind)*, *rTraveler* (*Role)* and

the *clientOf* (*RelationalDependency*), (iii) the inclusion of the *byAgePh* phase partition having the *YoungTraveler*, *AdultTraveler* and *OldTraveler* phases, and (iv) the removal of the *cPerson* and *cTraveler* concepts from the current model version.

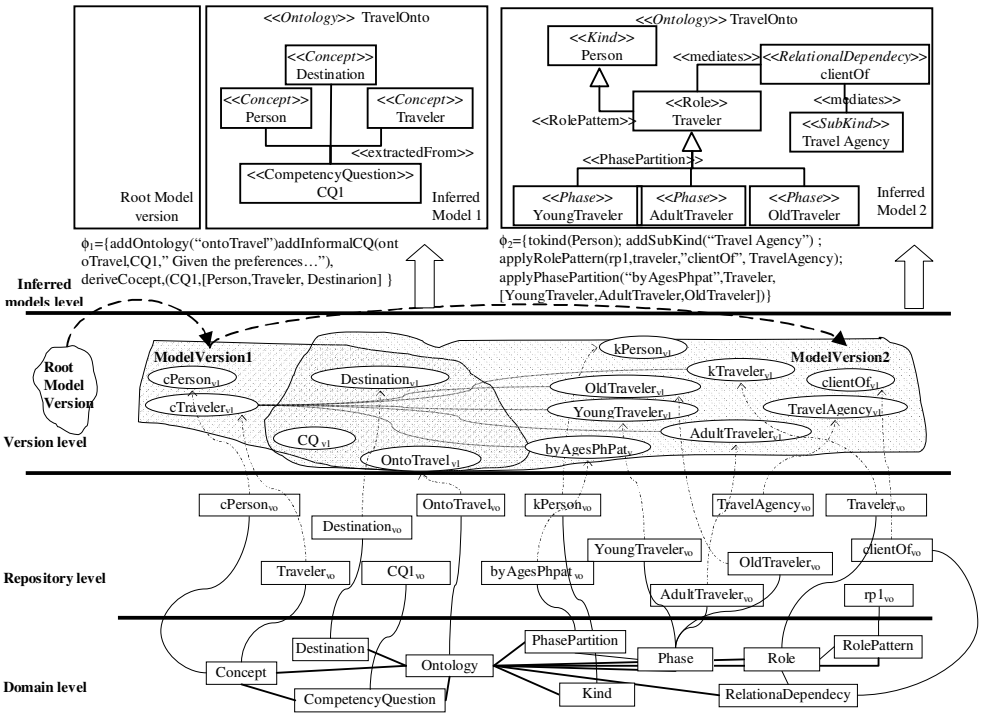

Figure 5. Specification of some operations belonging to the proposed model.

For each executed operation a version history link is created. For clarity reasons Fig. 5 only shows the version history links that relate *cTraveler* (*ObjectVersion)* in *ModelVersion1* with *kTraveler*, *OldTraveler*, *AdultTraveler*, *YoungTraveler* and *byAgesPhPart* (*ObjectVersion)* in *ModelVersion2*. By means of the history links it is possible to reconstruct the history of a given model version starting from the root one. The *Version Manager* presents such information in the so called *History Window,* which is illustrated in Fig. 6. In this pane it can be seen that TracOED allows keeping information about the development evolution of the *ontoTravel* ontology. From this knowledge it is possible to identify which are: (i) the predecessor and successors of *ModelVersion1*; (ii) the history links saving traces of the applied operation sequences, ϕ_1 and ϕ_2, which originated *ModelVersion1* and *ModelVersion2*, respectively; (iii) the set of object versions (*byAgePh*, *YoungTraveler*, *AdultTraveler* and *OldTraveler)* that appeared as a result of a given operation execution (*applyPhasePartition*).

Moreover, on the *Version Manager* History Window (Fig. 6) it is possible to see detailed data about each applied operation. For instance, this pane presents information about the time point at which a given operation was applied, who the involved actor was, and the identification of the successor object versions. In this example, the history window shows that an *applyRolePattern* operation was executed at *ModelVersion2* by mvegetti at time 11:40 -14/03/2012. It is possible to see that the execution of this operation also implied the addition of both, the *Traveler* role and the *clientOf* relational dependency.

It is important to remark that TracOED was developed with the aim of proving the proposed ideas and materializing the ONTOTracED framework. Therefore, this tool is not meant to replace traditional support environments. On the contrary, in the future TracOED should be integrated with existing ontology development tools, such as the OntoUML editor. In this way, TracOED would perform the capture of all the applied operations by working in a background mode, without being noticed by ontologists.

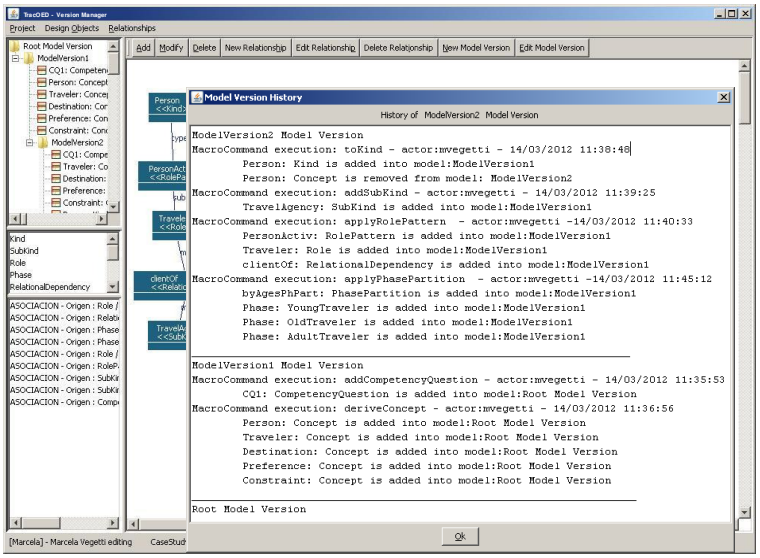

Figure 6. TracOED history window.

4. Conclusions

This contribution presents ONTOTracED, which is a framework aimed at capturing and tracing ontology development processes. The framework is based on a conceptual model of generic engineering design projects, an Ontological Engineering Domain Model, which specifies design objects and operations that are specific to ontology development processes, and a computational environment, named TracOED, which implements these models. The capabilities of TracOED have been presented and afterwards illustrated by means of a case study. The example shows that it is possible to keep track of the ontology development process along with its associated products, to store its history, allowing for the future retrieval of knowledge and experience. The proposal is flexible enough to be used in the development of ontologies that rely on particular methodologies and/or approaches, or that address particular fields. If needed, the TracOED domain editor can be used to extend the proposed Ontological Engineering Domain Model or to create a new one. To further validate the proposal, future work will be oriented to integrate TracOED with existing ontology development tools, like Protégé, the Neon Toolkit or the ontoUML editor, in such a way that its execution would take place in a background mode.

Acknowledgments

The authors wish to acknowledge the financial support received from ANPCyT (PAE-PICT-2315 and PAE-PICT-51), CONICET (PIP2754), UTN(PID 25-O117 and PID 25-0118), and UNL (CAI+D R4 N12).

References

Bernaras, A., Laresgoiti, I., Corera, J. (1996). "Building and Reusing Ontologies for Electrical Network Applications". In: the European Conference on Artificial Intelligence (ECAI'96), p. 298-302.

Cristani, M., Cuel, R. (2005). "A Survey on Ontology Creation Methodologies", *International Journal on Semantic Web and Information System*, 1, p. 49-69.

De Nicola, A., Missikoff, M., Navigli R. (2009). "A Software Engineering Approach to Ontology Building", Information Systems. 34, p. 258-275.

Fernández-López, M., Gómez-Pérez, A., Sierra, J. P. Sierra, A. P. (1999). Building a Chemical Ontology Using Methontology and the Ontology Design Environment, Intelligent Systems, 14, p. 37-46.

Gómez-Pérez, A., Fernandez-López, M., Corcho, O.: (2004). Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web, Springer, 2nd edition.

Guizzardi, G. (2005). Ontological Foundations for Structural Conceptual Models. PhD with Cum Laude, Telematica Instituut Fundamental Research Series, 015, Enschede, The Netherlands.

Guizzardi, G., Pinheiro das Graça, A., Guizzardi, R. (2011). "Design Patterns and Inductive Modelling Rules to Support the Construction of Ontologically Well-Founded Conceptual Models in OntoUML". In: 3rd International Workshop on Ontology-Driven Information Systems (ODISE 2011).

Gruber, T.R. (1993). "A Translation Approach to Portable Ontology Specification", Knowledge Acquisition. 5, p. 199-220.

Grüninger, M, Fox, M. (1995). "Methodology for the Design and Evaluation of Ontologies." In: Skuce D (ed). IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing, p. 258-269.

Roldán M. L, Gonnet S, Leone H. (2010). "TracED: A Tool for Capturing and Tracing Engineering Design Processes", Advances in Engineering Software, 41, p. 1087-1109.

Staab, S., Schnurr, H. P., Studer, R., Sure, Y. (2001)." Knowledge Process and Ontologies", IEEE Intelligent Systems. 16, p. 26-34.

Suárez-Figueroa, M. C., Gómez-Pérez, A., Motta, E., Gangemi, A. (2012). Ontology Engineering in a Networked World, Springer, First Edition.

Swartout, B., Ramesh, P., Knight, K., Russ, T. (1997). "Toward Distributed Use of Large Scale Ontologies". In: Symposium on Ontological Engineering of AAAI .

Uschold, M., Gruninger, M. (1996). "Ontologies: Principles, Methods and Applications", Knowledge Engineering Review, 11, p- 93–155.

Uschold, M., King, M., Moralee, S., Zorgios, Y. (1998). "The Enterprise Ontology", Knowledge Engineering Review, 13, p. 31–89.