A language for modeling enterprise contextual ontologies

Ma. Laura Caliusco[1], César Maidana[1], Ma. Rosa Galli[1,2], and Omar Chiotti[1,2]

[1] GIDSATD – UTN – FRSF, Lavaisse 610 – 3000 Santa Fe - Argentina
{mcaliusc, cmaidana}@frsf.utn.edu.ar
[2] INGAR-CONICET, Avellaneda 3657 – 3000 Santa Fe – Argentina
{mrgalli, chiotti}@ceride.gov.ar

Abstract. To achieve inter-enterprise software interoperability, the semantics of interchanged information by using electronic business documents, has to be explicitly modeled. A common approach to model semantics is to use an ontology. However, there is an emerging approach that combines an ontology with its context definition. So, the misunderstanding can be avoided if the context is explicitly defined. The resulting structure is called contextual ontology. To process a contextual ontology at run time, it has to be expressed in a machine procesable language. Recently, some languages have appeared. But, the main disadvantage of these languages is that they are mostly based on logic formalisms to support machine reasoning. Then, for the analysis and design phase of electronic business documents, a more appropriate contextual ontology modelling language is needed. To this aim, this paper presents a language for modelling explicit and formal contextual ontologies that assists business ontology designers in modelling contextual ontologies associated to electronic business documents.

Keywords. Contextual ontologies, inter-enterprise interoperability, modelling.

1 Introduction

Nowadays, enterprises confronts technological changes, such as Internet and mobile computing, and business changes, such as shorter product life cycles and global market; that makes business decisions complex and difficult. In this scenario, business managers need appropriate tools for conducting collaborative business over Internet. This way of doing business is called e-Collaboration. The e-Collaboration is supported through electronic business documents interchanges.

When information passes between companies, inconsistencies and misunderstandings occur very often, leading to wasted work efforts. To solve this problem XML-based specifications were defined to exchange business information between partners. These specifications propose to use the same vocabulary, that is, the same collection of terms. However, this is unrealistic since in the same department of different enterprises, people

could use the same term to define different concepts. This problem is known as semantic interoperability problem. In order to overcome this problem we have to explicitly define the meaning of the terms, its semantics [6].

A common approach to represent semantics is to use an ontology. However, it is well known that a human being does not reason without context. So, if we explicitly define the context, we improve the semantic interoperability. The resulting structure is called contextual ontology [2].

To process a contextual ontology at run time, it has to be expressed in a machine procesable language. Recently, some languages have appeared. However, from business perspective, the main disadvantage of these languages is that they are mostly based on logic formalisms to support machine reasoning. This makes the language syntax unfamiliar to business analysts who model the electronic business documents (EBD).

To overcome the gap between people involved in the definition of EBD and ontology specification languages, a contextual ontology modeling language is needed. A model consists of sets of elements that describe some physical, abstract, or hypothetical reality. Good models serve as means of communications [12].

The objective of this paper is to present a metamodel for modeling explicit and formal contextual ontologies, for human processing, associated to EBD. Firstly, we discuss some related works. Then, we present the proposed metamodel. Following, we analyze relationships between XML-based specifications and ontologies in order to add formal and explicit semantics to EBD. Finally, we present our conclusions.

2. Related works

The wide acceptance of the Unified Modeling Languages (UML), not only in academia but also in software development; makes it an ideal language to be used in order to create a critical mass of people able to build high quality models of information semantics. Cranefield (2001) proposed an ontology representation formalism based on a subset of the UML together with its associated Object Constraint Language (OCL) for agent software communication. However, UML itself does not satisfy needs for representation of ontology concepts that are borrowed from Descriptive Logic and that are included in ontology specification languages [8].

The Ontology Working Group is defining the Ontology Definition Metamodel (ODM) [13], which is a MOF2 (Meta Object Facilities) compliant metamodel. ODM allows a user to define ontology models using the same terminology and concepts as those defined in OWL, a semantic markup language for publishing and sharing ontologies on Internet [11]. In this metamodel the context definition is supported by using annotations in natural language which make the definition ambiguous. In addition, context features cannot be automatically transform into a machine procesable language.

The need for a dedicated contextual ontology modeling language stems from the observation that a contextual ontology cannot be sufficiently modeled in UML or ODM.

3 An overview of the proposal

In order to define the proposed modeling language, we have imported from the "UML 2.0: Infrastructure" specification [15] some elements of the Core::Abstractions, which contains a set of metaclasses to be specialized when defining new metamodels, and Core::PrimitiveTypes Packages, which simply contains a number of predefined data types.

The main design principles of the metamodel are: (1) easy to use in rapid development of contextual ontologies from electronic business documents, (2) modularity and (3) high independence degree of contextual ontology specification languages.

In order to fill the modularity design principles, the metamodel constructs were grouped into packages. The main package is the Kernel Package which imports the reused elements from Infrastructure::Core Package. All metamodel elements are derived from Kernel elements. Following we define the other packages of the proposed metamodel.

3.1 Modeling an Ontology

An ontology O_i is defined as a 4-tuple of the form $O_i = <T_i, P_i, R_i, A_i>$ where: i identifies the domain that an ontology is associated with; T_i is a set of terms $t_j \in O_i$;P_i is a set of properties of terms $t_j \in T_i$; R_i is a set of relations between t_j and $t_x \in T_i$ and A_i is a set of axioms.

Classes and associations, defined in the proposed modeling language for ontology modeling, are showed in Figure 1. The main component is *Ontology* class that includes definition of concepts used to describe a domain. This class is associated with the *OntologyElements* abstract metaclass, which groups the objects of an ontology metamodel. If an ontology is removed, so are the elements owned by it. The *imports* association represents that an ontology could contain definitions whose meanings are defined in other ontologies. The *prior_Version* association identifies the referred ontology as a prior version of one ontology. Each ontology element could be described by a comment, represented by the *Documentation* class. Finally, each ontology could be described by *Feature*, such as creation date, author, subject, title and so on.

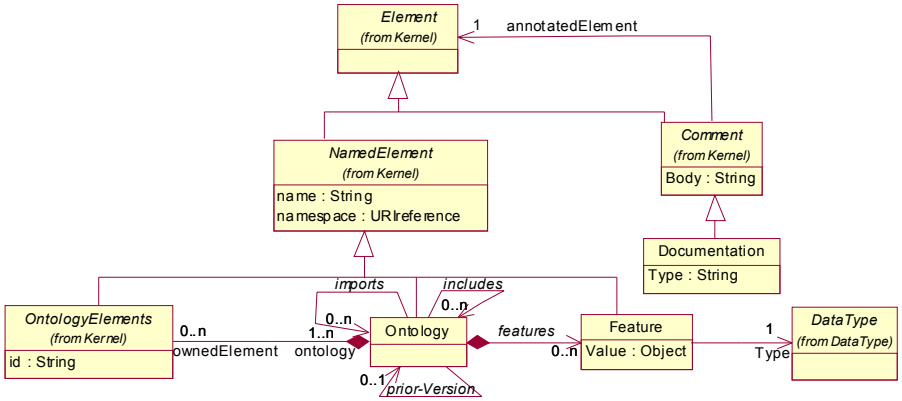

Figure 1. Elements defined in the *Ontology* package.

3.2 Modeling concepts

A term and their relations with other terms represent a concept. A term can be simple or complex. Simple terms have literal of some kind as their values. Complex terms are composed by simple or complex terms.

Properties describe the features of a term. For example, allowed values, the number of the values, and other features of the values that a simple or complex term could take.

The metamodel that represents the relation between *Property* and *Term* is showed in Figure 2. In the proposed metamodel, the class *Property* defines the features of a term so if a term is removed the properties owned by it have to be removed.

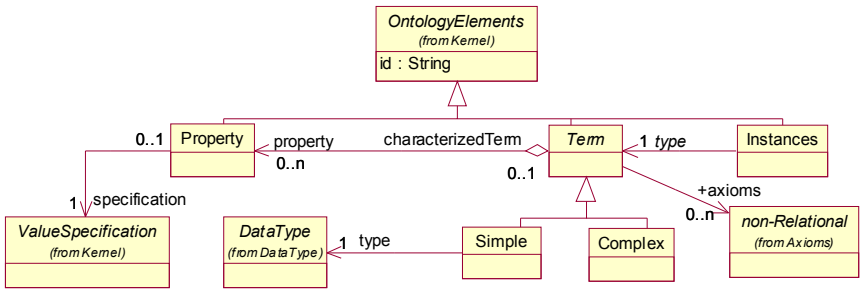

Figure 2. Elements defined in the *Properties and Terms* package.

3.3 Defining relations between terms

A set of relations R_i represents how terms belonging to O_i are related to. Relations can be divided into hierarchical relations (is-a, part-of), conceptual relations (synonym and antonym), particular relations (defined by the ontology modeler) and the relation that states that a term is an attribute of another term.

Term and *Relations* classes are associated via the *RelationEnd* class, as showed in Figure 3. An instance of *Relations* class has to be associated with two instances of *RelationEnd* class, the source and target. *RelationEnd* class is associated with one *Term* class and contains the information about cardinality and the role of terms. Furthermore, this class has the *Navigable* attribute to represent the direction of the relation.

The *Particular* class allows ontology modeler to model user-defined relations between terms. One important requirement for ontologies is the ability to structure the relations into hierarchies, i.e., to define sub-relations of a relation. Furthermore, it is suitable to define equivalent relations and inverse relations. *Subrelationof*, *inverseof* and *equivalentto* relations model these characteristics.

Each relation could be characterized by axioms and this is modeled by the *axiom:Relational* [0..n] association.

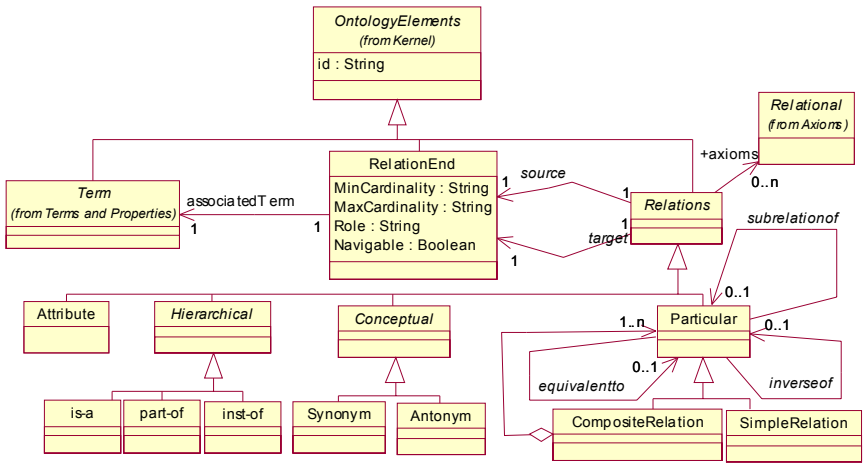

Figure 3. Elements defined in the *Relations* package.

3.4 Modeling axioms

Axioms are properties of relations and facts about concepts. Axioms help to constrain interpretation of concepts and provide guidelines for automated reasoning.

In the knowledge engineering area, axioms have been represented using logic languages. In the UML class diagram, axioms could be expressed by Object Constraint Language (OCL) [15]. However, describing such constraints may involve writing moderately complex OCL expressions that are not immediately understandable to a human reader. Furthermore, there may be several different expressions encoding the same constraint. So, an interesting issue is to represent axioms as objects [14] for modeling constraints.

Axioms can be divided into three subsets: the set of axioms for relational algebra, the set of general axioms and the set of axioms that states relations between attributes of a term. These sets of axioms are representing by *Relational*, *AParticular* and *non-Relational* classes, respectively, as showed in Figure 4.

The *Relational* class is an abstract class that groups the symmetric, transitive, antisymmetric, reflexive and functional axioms. The *AParticular* class is associated with the *Period* class indicating that an axiom could be true during a period of time.

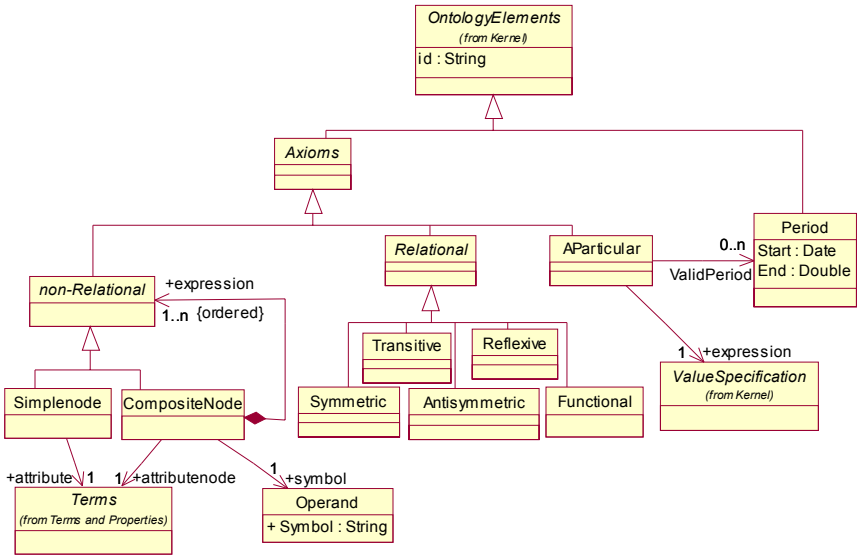

Figure 4. Elements defined in the *Axioms* package.

3.5 Contextualizing ontologies

There is no unique definition about what a context is. Different approaches use their own assumptions to define a context and use it for different purposes. In computer science, several context definitions have been defined in different areas, such as artificial intelligence, software development, databases, knowledge representation and semantics

web. However, all of these notions of context are very diverse and suffice for different purposes [9][3]. From business perspective, a context $C_j \ \forall j \in J$ can be defined as a 3-tuple <c, D_j, $O_{i,j}$>, where J is a set of indexes and c is the unique identifier of the context j, D_j is a set of assumptions about context j and O_i represents the ontology i within the context j.

In the class diagram of Figure 5, we associate the *Context* class with an *Ontology* class and with one or more *Assumptions* classes. In [1] the following components are defined as assumptions: the owner of the context, the group in which the context has been developed, the security information and information on how a context was generated. But, we preferred to define the class in general to allow a user to define their own *Assumptions*. In addition, a context could be derived from other context and this is modeled by the *derivedfrom* association.

Furthermore, a context could be a simple one or a complex one. That is, a context could be formed by other contexts. For example, the context of an enterprise could be composed by different contexts that represent different decision points, such as Forecasting, Planning, Scheduling, Product Design, and so on.

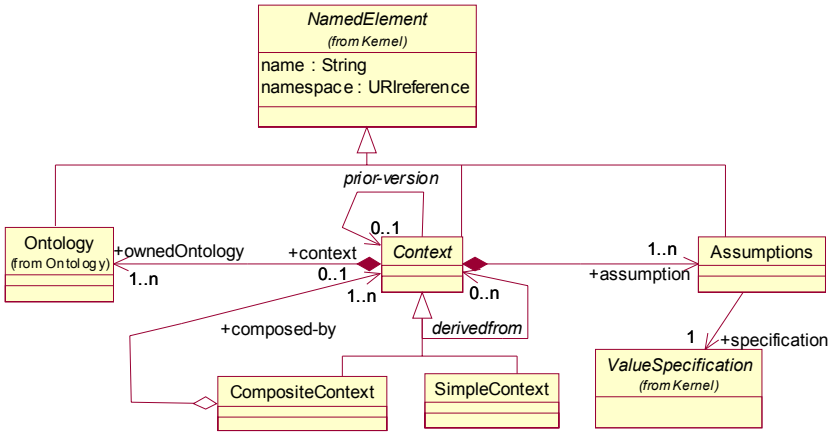

Figure 5. Elements defined in the *Context* package.

3.6 Modeling Context Mappings

A *context mapping* $M_{s,t}$ allows us to state that a certain property holds between elements defined in the source context *s* and the target context *t*.

Mappings are directional, i.e., $M_{s,t}$ is not the inverse of $M_{t,s}$. A mapping $M_{s,t}$ might be empty [2]. That means that there is no relation between both contexts.

A context mapping is defined by *bridge rules* as linking rules between contexts. A bridge rule from context *s* to context *t* is a statement of one of the following forms,

$c_s : e_i \xrightarrow{\equiv} c_t : e_j$ (1) $c_s : e_i \xrightarrow{\perp} c_t : e_j$ (2) $c_s : e_i \xrightarrow{*} c_t : e_j$ (3) $c_s : e_i \xrightarrow{\subseteq} c_t : e_j$ (4) $c_s : e_i \xrightarrow{\supseteq} c_t : e_j$ (5)

where e_i and e_j are elements of context c_s and c_t respectively.

Rule (1) means that e_i is similar to e_j. For example, Forecasting:Item $\xrightarrow{\equiv}$ Scheduling:Item.

Rule (2) means that both elements are disjointed. For example, Forecasting:Forecast $\xrightarrow{\perp}$ Scheduling:Employee.

Rule (3) means that e_i and e_j are compatible elements. For example, Forecasting:Forecast $\xrightarrow{*}$ Scheduling:Schedule (an Schedule derives from a Forecast).

Rule (4) means that e_i is less general than e_j and

Rule (5) means that e_i is more general than e_j. For example, Forecasting:Bucket $\xrightarrow{\supseteq}$ Schedudulling:Date (bucket: valid forecast time period).

Finally, we have to define the container of all the above defined elements. This container is the domain space concept. That is, the contexts and the mapping rules that relate the elements between contexts are contained in a domain.

Classes and associations to model context mappings, bridge rules and the domain space are showed in Figure 6. The *DomainSpace* class is associated with one or more *Context* and *Mapping* classes. A *Rules* class associated with the *BrigdeRules* class could be one of the five rules previously defined, *Equivalence*, *MoreGeneral*, *LessGeneral*, *Compatible* and *Disjunct*.

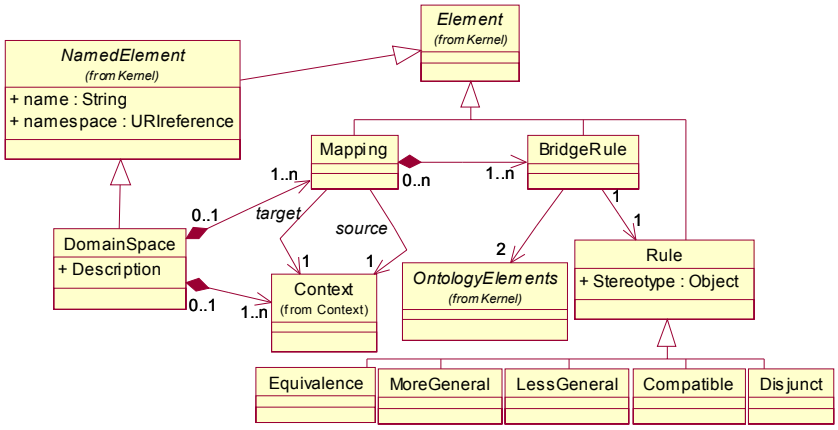

Figure 6. Elements defined in the *Context Mapping* package.

4 An example of ontology modeling

XML (eXtensible Markup Language) has became a standard for business information interchange in a e-Collaboration domain where the information is interchanged by using electronic business documents based on XML [6]. In order to integrate the information contained in the electronic business documents to their own information systems, both parties have to know what the information exactly means. For example, the meaning of the "requirement information" depends on the business collaborative process. That is, at Forecasting Process level "requirement information" means a demand forecast sent from a customer to a supplier. At Scheduling Process level, "requirement information" means a supply schedule sent from a supplier to a customer. To inform this information between a customer and a supplier the structure of the electronic business document could be the same, i.e. as shown in Figure 7; but the semantics changes.

First of all, trading partners have to define the context and the characteristics that make the context unique. Then, they decide about the schemas of business documents that will be interchanged during the e-Collaboration relationship. Finally, they have to model the semantics associated to these business documents.

```
1.  <xs:complexType name = "LocalAddress">
2.      <xs:complexContent>
3.          <xs:extension base = "Address">
4.              <xs:sequence>
5.                  <xs:element name= "zip" type = "xs:string"/>
6.              </xs:sequence>
7.          </xs:extension>
8.      </xs:complexContent>
9.  </xs:complexType>
10. <xs:complexType name = "Quantity">
11.     <xs:simpleContent>
12.         <xs:extension base = "xs:decimal">
13.             <xs:attribute name = "uom" use = "required" type = "UOM"/>
14.         </xs:extension>
15.     </xs:simpleContent>
16. </xs:complexType>
17. <xs:complexType name="PlanningSchedule">
18.     <xs:sequence>
19.         <xs:element name = "Date" type = "xs:date"/>
20.         <xs:element name = "Description" type = "Description"/>
21.         <xs:element name = "LocalAddress" type = "LocalAddress"/>
22.         <xs:element name = "Item" type = "Items"/>
23.         <xs:element  name="ItemQuantity" type="Quantity"  minOccurs="0"/>
24.     </xs:sequence>
25. </xs:complexType>
26. <xs:simpleType name = "Description">
27.     <xs:restriction base = "xs:string">
28.       <xs:maxLength value = "40"/>
29. </xs:simpleType>
30. <xs:element name = "PlanningSchedule" type="PlanningSchedule"/>
31. </xs:schema>
```

Figure 7. A fragment of the business document: Planning Schedule.

3.1 Rules defined for ontology modeling from XML document

To model an ontology from a document based on XML, we have defined some rules presented following:

1. The *xsd:complexType* element has to be modeled by the class *Complex*. For example, from the document defined in Figure 7, line 1 and line 17, we can model **LocalAddress** and **ReplenishmentOrder** terms as complex terms.
2. The *xsd:simpleType* element has to be modeled by the class *Simple*. For example, from the document presented in Figure 7, line 26, we can model **Description** element as a simple term.
3. The *xsd:element* element could be simple or complex depending on its type definition. That is, if they are defined as a base xsd type, they are simple terms. For example, on line 5 the **zip** element is defined as *xsd:string*. So, this element has to be modelled as a simple term. Then, on line 23, the term **ItemQuantity** element is defined as **Quantity**. Due to **Quantity** (line 10) is defined as *complexType*, **ItemQuantity** is a complex term.
4. The *xsd:restriction* element has to be modelled by *Properties* class. For example, on line 27 the **Description** element is restricted by using *xsd:restriction* definition so this characteristic has to be modelled as a property of the **Description** term.

After identifying the terms, the associations between them have to be defined. These relationships between concepts can be derived from an XML document as follows:

1. The *xsd:extension* element represents the *is-a* relationship between terms. For example, in Figure 7 line 18, the <xsd:extension base = "**Address**"> definition states that the previously defined element (**LocalAddress**) *is-a* **Address**.
2. The combination of *complexType* and *element* primitives represents the *part-of* relationship between terms. For example, all elements defined between the *complexType* primitives that define the **ReplenishmentOrder** term are related to it by the *part-of* relation. That is, **Date**, **Description**, **Items** and **LocalAddress** are *part-of* **ReplenishmentOrder**.
3. The *element* primitive represents the *Inst-of* relation between terms.
4. The *xs:attribute* represents the *Atributte* relation. For example, in line 13 the definition states that **uom** term is an attribute of **Quantity** term.

Figure 8 presents the model of the terms belonging to the XML-based document represented in Figure 7. On the one hand, at the left of the figure 8, we can see the syntactic model of the Planning Schedule document represented like a tree. On the other hand, at the right of the figure, we can see the semantic model. This model was obtained by applying the rules above defined. The graphical notation is based on [4].

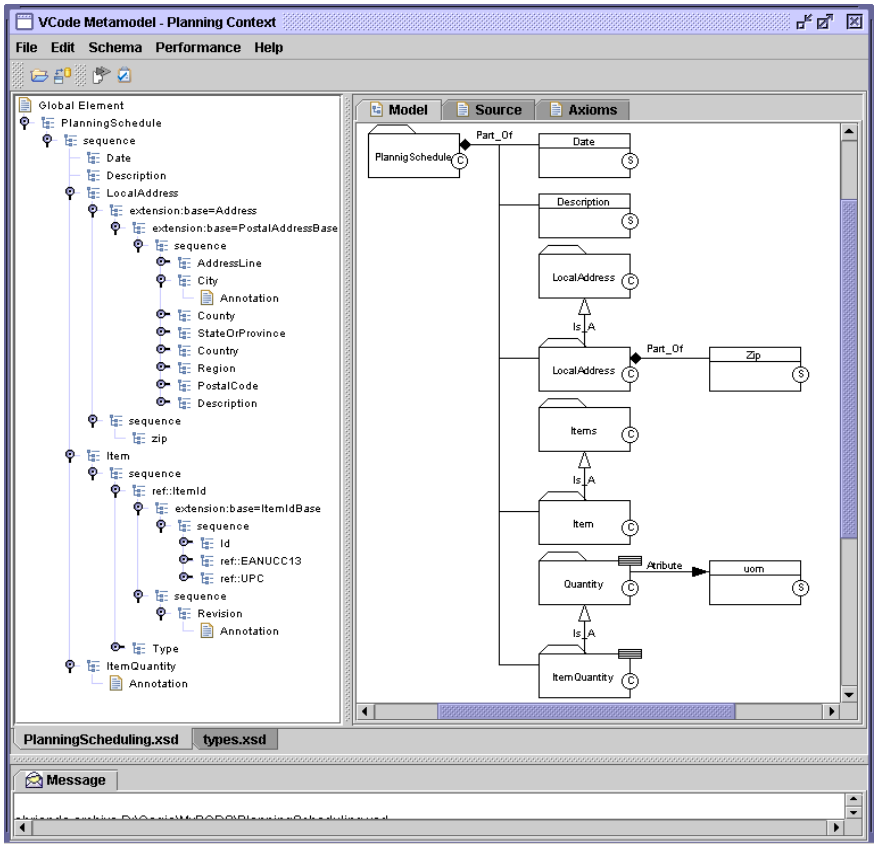

Figure 8. Syntactic and semantic models.

4 Conclusions

New information technologies provide new opportunities for allowing e-Collaboration which requires integration of the information and processes needed to conduct business in real time. XML is becoming widely used for defining electronic business documents within an e-Collaboration. However, these specifications need to be enriched using contextual ontologies due to XML does not express semantics by itself.

In order to fill the gap between people involved in the electronic business documents definition and contextual ontology specification languages we have defined a Contextual Ontology Definition Metamodel. This metamodel allows business analysts to explicitly model a contextual ontology. Furthermore, we have defined a set of rules to model the semantics implicit in XML-based business documents. These rules are based on the

elements that can be modeled automatically from an XML-based document. Other contextual ontology elements have to be defined by the ontology modeler.

Acknowledgment
This work has to be partially supported by Banco Rio S.A. and Universidad Tecnológica Nacional.

REFERENCES

[1] Bouquet, P, Dona, A, Serafini, L and Zanobini, S. Contextualized local ontologies specification via CTXML. AAAI-02 Workshop on Meaning Negotiation (MeaN-02) July 28, 2002, Edmonton, Alberta, Canada.
[2] Bouquet, P; Giunchiglia, F.; van Hamerlen, F.; Serafini, L and Stuckenshmidt. C-OWL: Contextualizing ontologies. Proc. of the 2nd Int. Semantic Web Conference, 2003. 164-179.
[3] Brézillon, P. (1999) Context in problem solving: A survey. The Knowledge Engineering Review, 14 (1), 1-34.
[4] Caliusco, Ma. Laura. Soporte para la definición semántica de Documentos de Negocio Electrónicos en relaciones de e-Colaboración. PhD Thesis. March 2005
[5] Caliusco, Ma. L.; Maidana, C.; Chiotti, O. and Galli, Ma. R. Propuesta de un Sistema Multiagente para Asistir al Modelado de Documentos de Negocio. Proceeding of Argentine Symposium on Information Systems (ASIS 2004).
[6] Caliusco, Ma. L., Galli, Ma. R. and Chiotti, O. Ontology and XML-based specifications for collaborative B2B relationships. Proceeding of III Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería de Conocimiento (JIISIC). Nov. 2003.
[7] Cranefield, S., Haustein, S. and Purvis, M.. UML as an ontology modelling language. Proceedings of the Workshop on Ontologies in Agent Systems, Canada, 2001.
[8] Duric, D.; Gasevic, D; Devedzic, V. A MDA-based Approach to the Ontology Definition Metamodel. Proceedings of a 4th Workshop On Computational Intelligence and Information Technologies. October 13, 2003, Faculty of Electronics, Niš, Serbia.
[9] Giunchiglia, F. and Bouquet, P. "Introduction to contextual reasoning. An Artificial Intelligence Perspective", in B. Kokinov (ed.), Perspectives on Cognitive Science, 3, NBU Press, Sofia (Bulgaria) 1997. http://dit.unitn.it/~bouquet/pers-publ-engl.html
[10] Kogut, P., Cranefield, S., Hart, L., Dutra, M., Baclawski, K., Kokar, M., Smith, J. UML for Ontology Development, Knowledge Engineering Review Journal Special Issue on Ontologies in Agent Systems, 2002 Vol. 17.
[11] McGuinness, D and van Harmelen, F. OWL Ontology Web Language - Overview. W3C Recomended Propesed. December 2003. http://www.w3.org/TR/owl-features/.
[12] Mellor, S; Scott, K; Uhl, A. and Weise, D. MDA Distelled – Principles of Model-Driven Architecture. ADDISON-WESLEY, 2004.
[13] Ontology Definition Metamodel (ODM). Initial Submission to OMG. August, 2003.
[14] Staab, S; Mädche, A.. Axioms are objects, too - Ontology Engineering Beyond the Modeling of Concepts and Relations. Proc. of the ECAI 2000 Workshop on Ontologies and Problem-Solving Methods. Berlin, August 21-22, 2000.
[15] UML 2.0 Infrastructure – Final Adopted Specification. September, 2003.