IBIMA Publishing
Communications of the IBIMA
http://www.ibimapublishing.com/journals/CIBIMA/cibima.html
Vol. 2010 (2010), Article ID 749128, 16 pages
DOI: 10.5171/2010.749128

Elaboration of A Process for Mapping between Ontology Concepts and Objects Model Concepts

Hatem BEN STA
University Of Tunis at El Manar, Tunis, Tunisia

Abstract:

The principal idea of the framework in this paper is to use ontologies to convert a problem domain text description into an object model. The object model of a system consists of objects, identified from the text description and structural linkages corresponding to existing or established relationships. The ontologies provide metadata schemas, offering a controlled vocabulary of concepts. At the center of both object models and ontologies are objects within a given problem domain. The difference is that while the object model should contain explicitly shown structural dependencies between objects in a system, including their properties, relationships, events and processes, the ontologies are based on related terms only. On the other hand, the object model refers to the collections of concepts used to describe the generic characteristics of objects in object-oriented languages. Because ontology is accepted as a formal, explicit specification of a shared conceptualization, we can naturally link ontologies with object models, which represent a system-oriented map of related objects, described as Abstract Data Types (ADTs). This paper addresses ontologies as a basis of a complete methodology for object modeling, including available tools, particularly CORPORUM OntoExtract and VisualText, which can help the conversion process. This paper describes how the developers can use this framework and implement it on the base of an illustrative example.

Keywords: object model, ontologies, class model, ADT.

1. Introduction:

Ontology is a specification of a representational vocabulary for a shared domain of discourse: definitions of classes, relations, functions, and other objects as mentioned by Gruber (1993) or, more generally, a specification of conceptualization and also mentioned by Gruber (1994). Semantic Web uses ontologies as a tool for easy integration and usage of content by building a semi-structured data model. To solve the problem of heterogeneity in developing software applications, there is a need for specific descriptions of all kinds of concepts, for example, classes (general things), the relationships that can exist among them, and their properties (or attributes) in the draft of Helfin et al. (2002). Ontologies described syntactically on the basis of languages such as eXtensible Markup Language (XML), XML Schema, Resource Description Framework (RDF), and RDF Schema (RDFS) can be successfully used for this purpose.

Object orientation is a commonly accepted paradigm in software engineering for the last few decades. At the initial analysis phase, identifying the right objects, which are vital to the system's functionality, seems to be the most difficult task in the whole development process, from both theoretical and practical point of view. Object-oriented software development is well supported by a huge number of working methods, techniques, and tools, except for this starting point - object identification and building the related system object model. Framework for converting the text description of system problem domain and respective functional requirement specifications into an object model is usually left to the intuition and experience of developers (system analysts). One commonly accepted rule is, "If an object fits within the context of the system's responsibilities, then include it in the system." However, since the members of a development team are likely to have different views on many points, serious communication problems may occur during the later phases of the software development process. Recently there has been great research interest in applying ontologies for solving this "language ambiguity problem" as either an ontology-driven or ontology-based approach in the paper of Deridder et al. (1999). This is especially true for object-oriented software engineering, mainly because of the similarity in the principles of the two paradigms. More over, the object systems similar to ontologies, which represent conceptualized analysis of a given domain, can be easily reused for different applications as cited by Swartout (1999).

Representation of objects as Abstract Data Types (ADTs) is of primary importance in developing object-oriented software because it is actually a process of software implementation of ADTs. Any ADT is a named set of attributes, which show the characteristics of and formalize the relationships between objects and methods (operations, functions) for putting into effect the behavior of objects, making the system functional enough to be of practical use. Building an accurate, correct and objectively well-defined object model containing objects, represented as ADTs, is the basis for successful development of an object-oriented software system mentioned by Weiss (1993) and Manola (1999). The basic idea is that the implementation of ADTs as a code allows all working objects (instances of classes) to have one and the same behavior, which can be changed dynamically in a centralized manner for higher efficiency and effectiveness. Objects are transformed during the software development process from "real things" to concepts, and finally to Abstract Data Types, as shown in Figure 1.

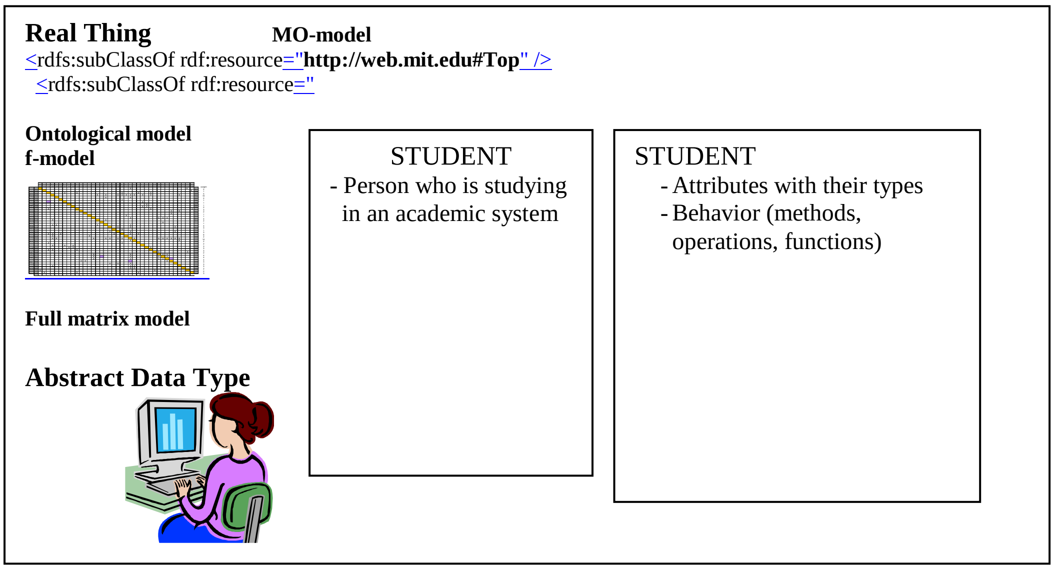

Real Thing
MO-model
<rdfs:subClassOf rdf:resource="http://web.mit.edu#Top" />
<rdfs:subClassOf rdf:resource="
Ontological model
f-model
Full matrix model
Abstract Data Type
STUDENT
- Person who is studying in an academic system
STUDENT
- Attributes with their types
- Behavior (methods, operations, functions)

Figure 1 Conceptualization and ADTs

The framework described in this paper, is based on eight different models, only two of which, namely the text description model (T-model) and class (object) model (C-model), are included in the classical object-oriented software development process. The rest of the models used represent specific analysis work, which the developers should do, to get benefit from using ontologies for semi-formal identification of objects, which are to be responsible for the system functionality, and their respective ADTs.

The basic idea is to ensure suitable transformation of the models from one to another using respective procedures and tools, which can be considered as potential elements for integrating ontologies into CASE tools for object-oriented systems. The paper is structured as follows: Section 2 introduces the models in general and describes the overall procedure for their transformation; Section 3 is dedicated to a more detailed description of the models as well as to discussion on the techniques and tools, which can be practically used for model transformation. An illustrative example of a part of the information system for the domain of academic management is used throughout the paper to support the explanations; finally, section 3 summarizes the proposed approach and highlights direction for future work.

2. Overview of the Framework:

Our framework is based on transformation of models. Models are inseparable and one of the most significant parts of any methodology. They help developers to better understand complex tasks and represent in a simpler way the work they should do to solve those tasks. Object-oriented analysis of a system under development is a good example of such a complex task. The complexity stems from the fact that in object-oriented development everything is based on objects but their identification in a given problem domain is completely left to the intuition of the developer. All that he/she has as a starting point is the text description of the problem domain, which is itself an extended model of the usually very general and ambiguous initial user requirements. Following the existing practice we accept this text description (T-model) as the available model, which serves as a starting point of our transformation process. According to the object-oriented software development methodology the analysis work on the T-model leads to two major deliverables: functional specification of the system, expressed as either text or graphically as Use Case diagrams and the object (class) model (we call it C-model). The ultimate goal of the developer's efforts is actually the creation of the C-model. This is so because the objects included the C-model should contain the complete information necessary for the next phases of design and implementation of the software system.

In other words the objects should be represented as ADTs - ready for design and implementation software modules. It is clear now the already mentioned problem with "language ambiguity" - different interpretations of the T-model, without any formal support of the choice of participating objects, would lead to creating C-models, which are quite probably inconsistent, incomplete or inefficient for the further steps of design and implementation. We believe that using ontology as a tool of conceptualization working on the T-model can make (if not fully formal at least) semi-formal the process of creating the C-model and in this way help developers in this complex and imprecise task. This is the major motivation of our work described briefly in this paper

.

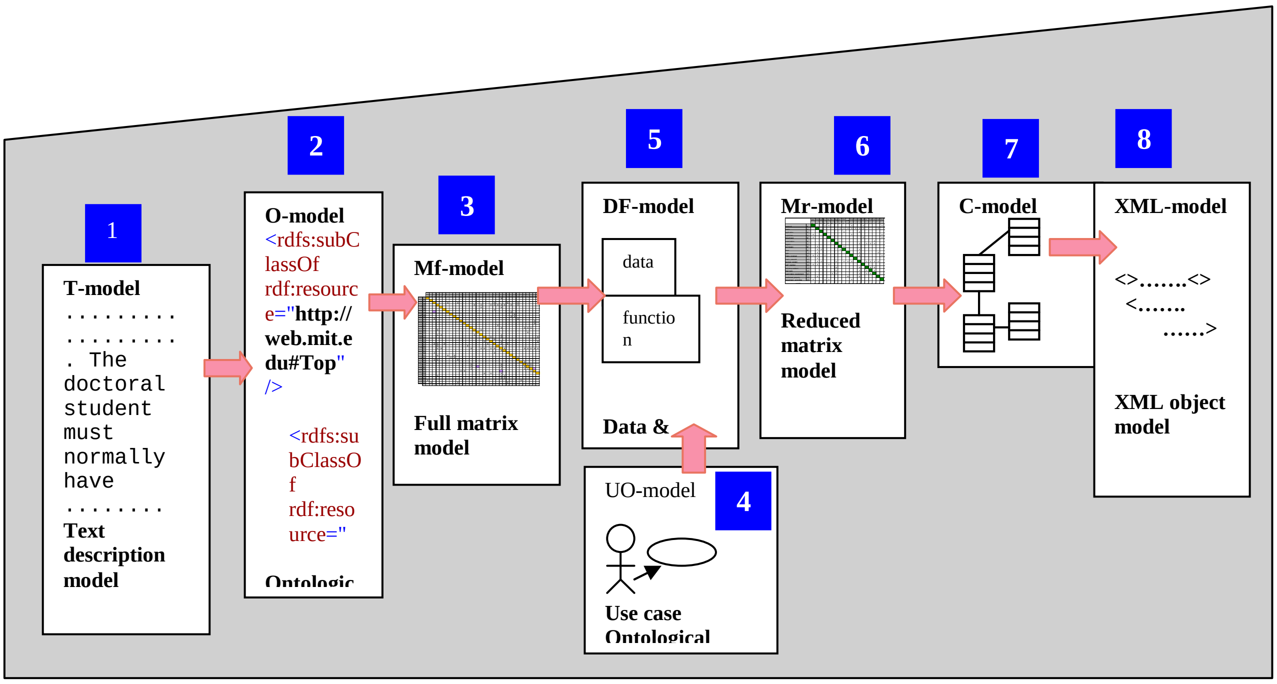


Figure 2 Framework Models for converting a text description into an object model

Figure 2 shows the basic idea of the proposed framework, models used and transformation process on them. The starting point of the transformation is the T-model, which represents a concise description of the problem domain, where the software system under development will work, written in a natural language, in our case English. If not available the T-model is a deliverable from a system analyst's work on the general user requirements for the system functionality. The presumption is that this problem domain description contains the main objects, which will participate in ensuring that functionality. Of course, at this level the objects are represented by their natural names only and as such are very far from the form we need to reach - represented as ADTs. To help this process we refer to a tool of conceptualization - an ontological engine, which applied on the T-model generates an ontological description (O-model) of the problem domain at hand.

We use the fact here that any ontology is a systematic description of concepts (objects) in a given domain of interest along with expressed relationships between all or part of them. This is actually the crossroads between the object-oriented and ontology-based paradigms. The O-model is a straightforward and practically useful source of information for identifying the participating objects. We use this information to build a so called Full Matrix model (Mf-model), which represents in a simple form those objects as well as the linkages (relationships) between them. However, it is worth noting that the processing of the Mf-model is semi-formal in nature. This means that at this phase the developer should take important decisions about which objects could be considered as basic ADTs and which, and where, could play a role of attributes of other ADTs. The idea is simple but not very easy for implementation - to reduce the full object matrix to a matrix (we call this model Mr-Model), which contains only the basic objects represented later as ADTs containing other ADTs as attributes. As was mentioned, the implementation is not very easy because we need more information here, which relates to expected functionality of participating objects. This information, however, is available or can be extracted from the Use Case model of the system under development. Also mentioned was that the Use Case model is another basic deliverable from the system analyst's work on the user requirements, which practically consists of a number of Use Cases decomposing the main Use Case of the system. Note that at this phase we can also use the already generated problem domain ontology. Along with showing the concepts hierarchy (possible objects in the system) the ontologies also analyze the verbs linking those concepts, which can be considered as functions (operations) belonging to respective objects.

We actually use the text descriptions of different Use Cases to extract different functionality of the system by the ontological engine and as a result we get the so called Use Case Ontological model (UO-model). We show later in section 3 that the functionality, expressed by the UO-model, can be used successfully at this particular phase along with the ontological information about the objects in the Mf-model to create a Data and Function model (DF-model). As a matter of principle DF-model can be used for each of the objects in the Df-model but this would lead to a high degree of redundancy and quite complicated matrix presentation even for relatively simple T-models. To avoid this we propose using so called business object patterns, which can be a result from ontology-based analysis. The idea is to use ontological libraries existing recently for a great number of application domains and to rely on the ontological description of the concepts (objects), which according to the developer's decision have the highest degree of likelihood of being, selected as basic objects in the system. This will allow for significant reduction of the number of possible objects in the Df-model, or we can transform it to the Mr-model.

We assume that this model contains all the necessary information for building the C-model, which is actually the goal of this first phase of analysis object-oriented software systems. The representation of the C-model is significantly different from Mr-model however, as far as the former shows not only the object hierarchy but the objects' structure as well. In other words, the C-model is a model representing ADTs. The last model, the XML-model is optional but can be very important in practice because it allows the C-model to be published on the Web in a unified (XML-based) format supporting in this way the collaborative work, which is a commonly accepted technology nowadays.

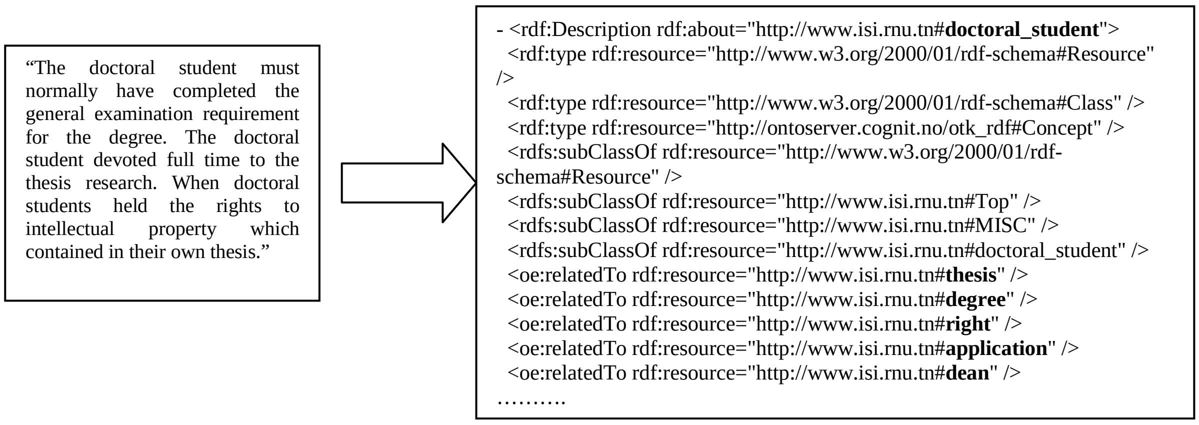
Finally, an interesting question may arise here. Do the models proposed in this framework replace or ignore the well known and widely used in practice models applied to the analysis of object-oriented systems? The answer is certainly not. All models, such as the information model, state model, process model, functional model, etc,, along with their accompanying methods, techniques and tools (for example those included in Objecteering CASE tools) remain absolutely necessary for completing the phase of object-oriented analysis. More over, all of them are created to be applied on the object model of the system under development and therefore, they will use the basic deliverable of the transformation process, shown above. What we have proposed is a semi-formal procedure for converting a text description of a given problem domain into an object model, which should be considered as a basis for further analysis work. Identification of objects and representing them as ADTs using ontologies is the major objective and achievement of the proposed approach.

3. The models used:

In this section we will briefly show the foundation, role and structure of the models used in the transformation process, explained generally in section 2. In addition, we will show some of the tools, mainly the ontological ones, which can be used for implementing the models. One and the same example - a part of a university information system regarding PhD students - is used as an illustration where needed.

3.1 T-model: Text description model:

The T-model or text description of a problem domain model that we were working on is an English text description of a part of a specific problem domain, shown on the left side of Figure 3.



"The doctoral student must normally have completed the general examination requirement for the degree. The doctoral student devoted full time to the thesis research. When doctoral students held the rights to intellectual property which contained in their own thesis."

```
- <rdf:Description rdf:about="http://www.isi.rnu.tn#doctoral_student">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"
/>
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class" />
  <rdf:type rdf:resource="http://ontoserver.cognit.no/otk_rdf#Concept" />
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Resource" />
  <rdfs:subClassOf rdf:resource="http://www.isi.rnu.tn#Top" />
  <rdfs:subClassOf rdf:resource="http://www.isi.rnu.tn#MISC" />
  <rdfs:subClassOf rdf:resource="http://www.isi.rnu.tn#doctoral_student" />
  <oe:relatedTo rdf:resource="http://www.isi.rnu.tn#thesis" />
  <oe:relatedTo rdf:resource="http://www.isi.rnu.tn#degree" />
  <oe:relatedTo rdf:resource="http://www.isi.rnu.tn#right" />
  <oe:relatedTo rdf:resource="http://www.isi.rnu.tn#application" />
  <oe:relatedTo rdf:resource="http://www.isi.rnu.tn#dean" />
……….
```

Figure 3 Text description model

This text is represented as an ontology description after processing by an ontological engine tool in our case CORPORUM OntoExtract as mentioned by Engles (2001). It is a Web-based version of CORPORUM, which is able to extract ontologies and represent them in XML/RDF/OIL (default in RDF schema) and also to communicate with and negotiate the final format of the to-be-submitted ontology extracted from a specific text as cited by Engles (2001). This tool can interpret text, in the sense that it builds ontologies that reflect world concepts as the user of the system sees and expresses them. So at this point in the process, the text is automatically processed and converted into ontologies, which can be done on line.

3.2 O-model: Ontological model

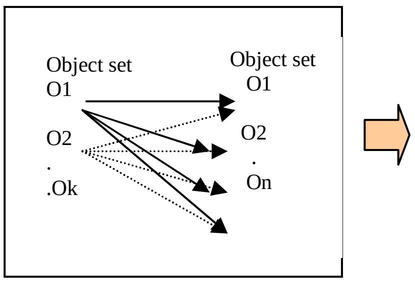
The ontology described in RDFS defines the names and relations of the extracted concepts, or object names. RDFS provides a mechanism to define domain-specific properties and classes of resources to which developers may apply those properties as cited by Klein (2001). More specifically, an ontology description is recognizable as an ontology language. Classes are specified with <rdfs: class>. Subclasses and subproperties are specified using <rdfs: subClassOf> and <rdfs: subPropertyOf> (the top class defined in the schema is "Resource") respectively. When a class is a subclass of several superclasses, this is interpreted as a conjunction of superclasses in a research study by Gil and Ratnakar (2002). CORPORUM OntoExtract basically generates taxonomies that represent classes, subclasses, and instances. A class described in the text may also be defined as a subclass of the universal "rdf: resource" if no more information about the class can be found. A class may also be defined as a subclass of other classes if evidence is found that the class is indeed a subclass. A subclass relationship found by this tool is based on information about the term in a research study by Engles and Bremdal and Jones (2001).

An important category that is exported by the CORPORUM OntoExtract engine is the *cross-taxonomic* relations. While a typical ontology often represents taxonomy, <isRelated> refers to cross-taxonomic links that may exist within a domain and, if represented, can make a difference in finding needed information based on context descriptions. In short, it can identify the possible relations between objects. For example, in the box on the right side of Figure 3, the class "doctoral_student" has certain relations with other classes, for instance, "thesis", "degree", "right", etc.

3.3 Mf-model: Full matrix model

The O-model describes only the type (object) name and provides relations between possible objects. However, this model is in a form difficult to understand and work with identified objects. That is why we use the RDFS description as an input to create a simple matrix-based model, which can serve as an intermediate model. It contains all identified objects, approved by the developer and allows easy manipulation on this full set of objects. This is the reason to call this mode a full matrix model (Mf-model).

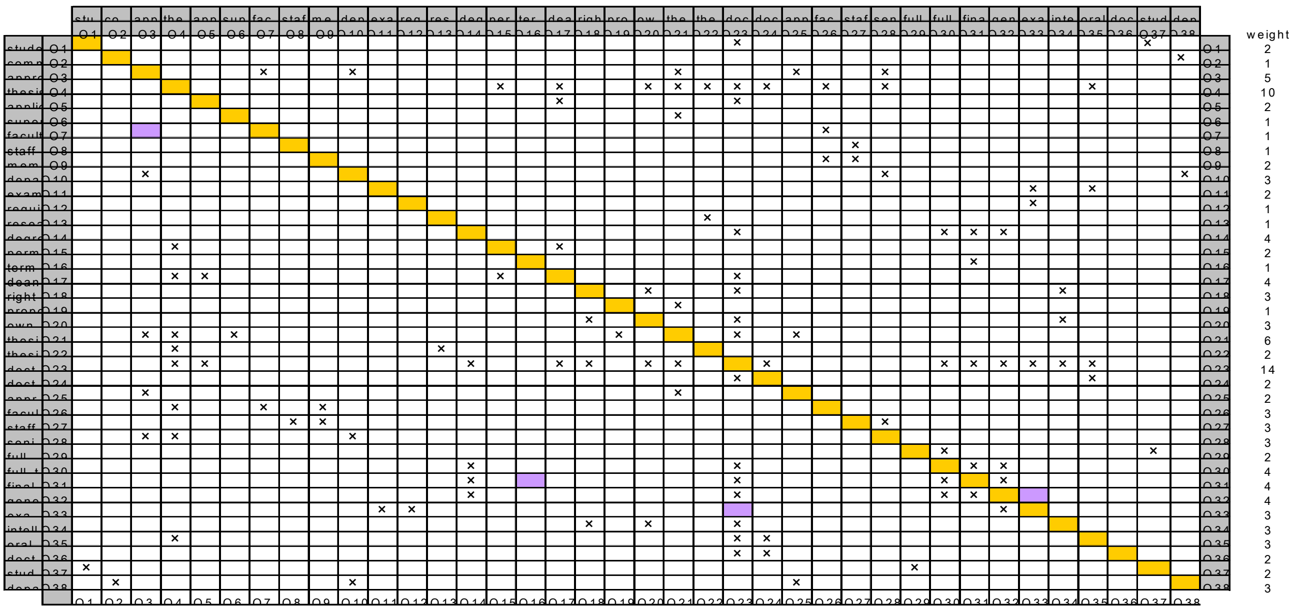
The relationships between objects in the system can be represented as simple mapping as shown in Figure 4 below. Generally speaking, we can always define two sets of k and n objects (k ≠ n in the common case) in a system between elements where relationships exist or can be established. If the objects are numbered differently each X in the table of Figure4 will represent those relationships.



	O1	O2	..	Ok
O1	X	X		X
O2	X	X		X
..			X	
On	X			X

Figure 4 Example of relationships between objects

Based on the above general considerations, we can build a full matrix as depicted in Figure 5 to show every relationship that occurs between already identified objects.



		stu	co	app	the	app	sup	fac	staf	me	dep	exa	req	res	deg	per	ter	dea	righ	pro	ow	the	the	doc	doc	app	fac	staf	sen	full	full	fina	gen	exa	inte	oral	doc	stud	dep		weight
		O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11	O12	O13	O14	O15	O16	O17	O18	O19	O20	O21	O22	O23	O24	O25	O26	O27	O28	O29	O30	O31	O32	O33	O34	O35	O36	O37	O38		
stude	O1	🟨																						×														×		O1	2
comm	O2		🟨																																				×	O2	1
appro	O3			🟨				×			×											×				×			×											O3	5
thesis	O4				🟨											×		×			×	×	×	×	×		×		×							×				O4	10
applic	O5					🟨												×						×																O5	2
super	O6						🟨															×																		O6	1
facult	O7			🟪				🟨																			×													O7	1
staff	O8								🟨																			×												O8	1
mem	O9									🟨																	×	×												O9	2
depa	O10			×							🟨																		×										×	O10	3
exam	O11											🟨																						×		×				O11	2
requi	O12												🟨																					×						O12	1
resea	O13													🟨									×																	O13	1
degre	O14														🟨									×							×	×	×							O14	4
perm	O15				×											🟨		×																						O15	2
term	O16																🟨															×								O16	1
dean	O17				×	×										×		🟨						×																O17	4
right	O18																		🟨		×			×											×					O18	3
propo	O19																			🟨		×																		O19	1
own	O20																		×		🟨			×											×					O20	3
thesi	O21			×	×		×													×		🟨		×		×														O21	6
thesi	O22				×									×									🟨																	O22	2
doct	O23				×	×									×			×	×		×	×		🟨	×						×	×	×	×	×	×				O23	14
doct	O24																							×	🟨											×				O24	2
appr	O25			×																		×				🟨														O25	2
facul	O26				×			×		×																	🟨													O26	3
staff	O27								×	×																		🟨	×											O27	3
seni	O28			×	×						×																		🟨											O28	3
full	O29																													🟨	×							×		O29	2
full t	O30														×									×							🟨	×	×							O30	4
final	O31														×		🟪							×							×	🟨	×							O31	4
gene	O32														×									×							×	×	🟨	🟪						O32	4
exa	O33											×	×											🟪									×	🟨						O33	3
intell	O34																		×		×			×											🟨					O34	3
oral	O35				×																			×	×											🟨				O35	3
doct	O36																							×	×												🟨			O36	2
stud	O37	×																												×								🟨		O37	2
depa	O38		×								×															×													🟨	O38	3
		O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11	O12	O13	O14	O15	O16	O17	O18	O19	O20	O21	O22	O23	O24	O25	O26	O27	O28	O29	O30	O31	O32	O33	O34	O35	O36	O37	O38		

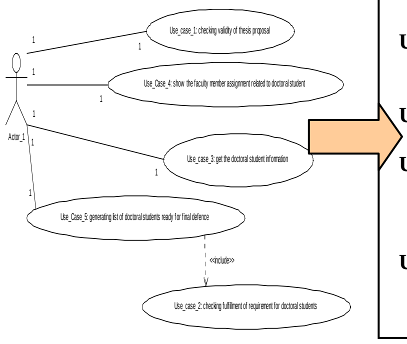
Figure 5 Full matrix model (Mf-model)

The total number of relationships an object has with other objects is called the weight of that particular object. It determines how many relationships one object has to other objects participating in this problem domain. One may infer that higher is the weight the higher the relevance of that object in the domain or, in other words, the higher the likelihood is that this particular object can be considered as a separate ADT in the software system. Following heuristics from previous experience we can define here some quantitative characteristics of the weight as a parameter, for example it's minimum, from which an object may be considered as a separate one. This can significantly help the developer to identify the basic objects in the system, although his/her decision making is still necessary. This is actually the semi-formal nature of the approach proposed in this paper.

3.4 UO-model: Use Case Ontological model

It was mentioned already in section 2 that the information from Mf-model, although useful, is perhaps redundant and certainly far from complete. Thanks to the ontological analysis the system analyst may have information about the possible objects in terms of names and partially as their attributes (other objects) but has no any information about the system behavior of objects. This means that at this phase we cannot talk about ADTs. Obviously, additional information is necessary related to system functionality, in which different objects are involved. Such information is of vital importance for identifying the complete contents of objects as data and behavior (objects' functions, operations), which are fundamental elements of the object model in a research study by Batanov and Arch-int (2003). Moreover, considering system functionality at this early stage of analysis may help the system analyst to define more precisely the basic objects in the system, to add new objects or to remove/replace already identified objects, which are not important for any of the system functions. This is the place where we should turn our attention to the Use Case modeling.

Use Case Modeling is the process of identifying and modeling business events, who/what initiates them and how the system responds to them. Use cases capture requirements from the perspective of how the actor will actually use the system, in other words each of them describes a given functionality of the system as mentioned by Bennett et al. (1999). Any Use Case can be represented either graphically (as a Use Case diagram) or as a text description. We use the functionality text description in order to apply the same ontology-based approach for creating the O-model (see Figure6 for clarifying the difference between Use Case diagram, Use Case text description and functionality text description). In this situation, however, another ontological engine, VisualText is used as a tool.



Use_case_1: checking validity of thesis proposal
Use_Case_4: show the faculty member assignment related to doctoral student
Use_case_3: get the doctoral student information
Use_Case_5: generating list of doctoral students ready for final defence
<<include>>
Use_case_2: checking fulfillment of requirement for doctoral students
Actor_1
1

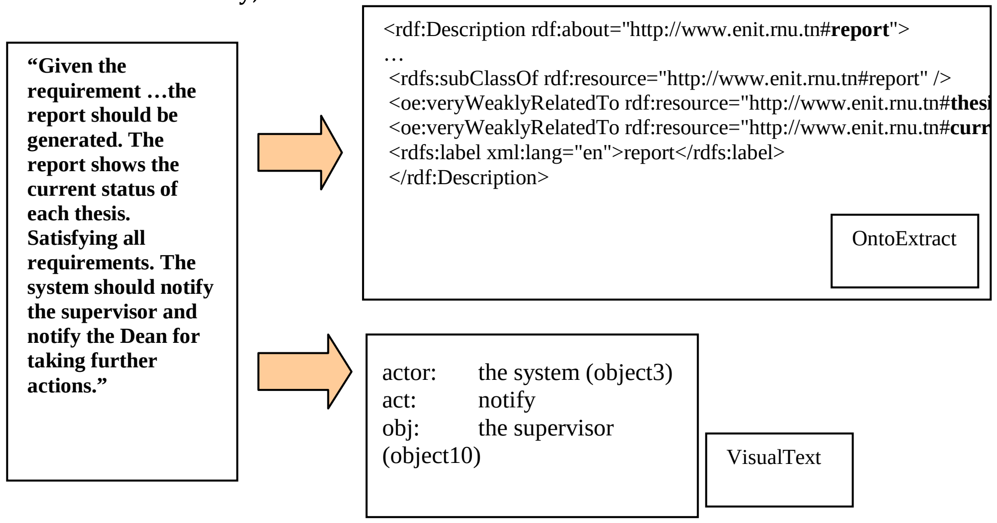
Use case1: Check validity of thesis proposal
Use case2: Check fulfillment of requirements for doctoral students
Use case3: Get the doctoral student's information
Use case4: Show the faculty member assignment related to doctoral students information
Use case5: generating list of doctoral students ready for final defense.

"Given the requirement for the degree, the system should check periodically the fulfillment the doctoral student for the doctoral thesis. The report should be generated. The report shows the current status of each thesis. Satisfying all requirements the system should notify the supervisor and notify the Dean for taking further actions."

Figure 6 Use case diagram, use case description, and functionality text description

VisualText is a tool for information extraction, natural language processing and text analysis systems. It makes it possible to find out the function within an event or action assigned to particular actors and/or objects in the system. Thus, the goal of UO-model is to analyze the functionality description and as a result to add functions/operations to respective objects.

As illustrated in Figure 6, several use cases may be used to describe one well-defined functionality of the system to be built within the problem domain. The ontological analysis of such a functionality description helps the system analyst to identify more precisely the real objects, which will play a substantial role in implementing the respective system functions. This can be done comparing (matching) the objects, already identified in the O-model. Obviously, if we have more than one functionality description to analyze, respective objects will be defined for each of them. It becomes easier now for the developer to decide which object should be considered as a separate ADT and which as an element of another ADT. For example, if a new object appears as a result of the ontological analysis of a functionality description but is not identified as a separate object from the O-model, it must be considered as an additional separate object now. Figure 7 illustrates how the two tools OntoExtract and VisualText can help determining which functions are relevant to the working objects in our problem domain description. The figure also shows that it is possible for new relationships to appear between the objects generated by the two tools, which means that they should be formalized in respective new attributes.


"Given the requirement …the report should be generated. The report shows the current status of each thesis. Satisfying all requirements. The system should notify the supervisor and notify the Dean for taking further actions."

```
<rdf:Description rdf:about="http://www.enit.rnu.tn#report">
…
 <rdfs:subClassOf rdf:resource="http://www.enit.rnu.tn#report" />
 <oe:veryWeaklyRelatedTo rdf:resource="http://www.enit.rnu.tn#thesi
 <oe:veryWeaklyRelatedTo rdf:resource="http://www.enit.rnu.tn#curr
 <rdfs:label xml:lang="en">report</rdfs:label>
 </rdf:Description>
```

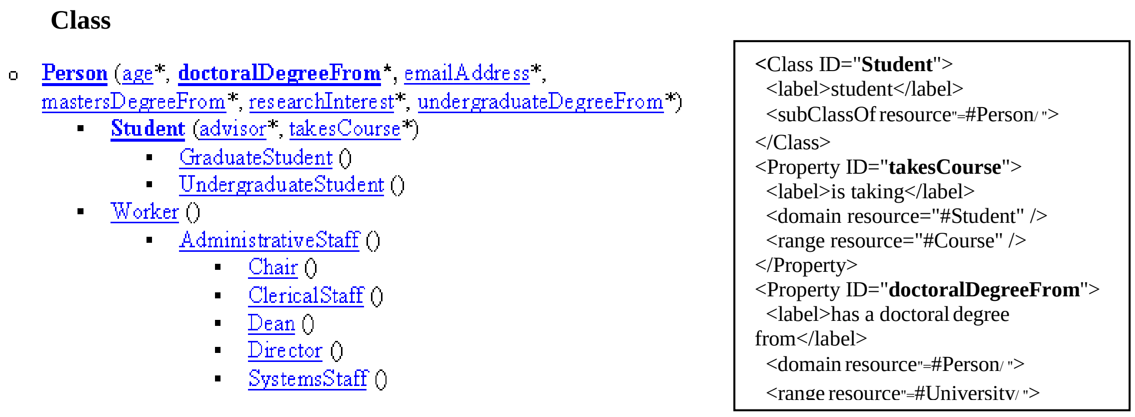
```
actor:    the system (object3)
act:      notify
obj:      the supervisor
(object10)
```

Figure 7 Output from a functionality text description

3.5 DF-model: Data and function model

Data (attributes) and functions (methods, operations) are the two fundamental parts of any object, represented as ADT. Each of the models introduced already has its own contribution to creating one or another element of those two parts. We can continue in this way relying on the decision making abilities of the developer to the final acceptable object model of the system containing ADTs. However, because of the requirement for decision making this process can still be characterized as subjective or even intuitive, which was the main reason to propose our approach. To avoid this situation we can recall the most powerful feature of both object and ontology orientation - they allow for a high degree of reusability of their artifacts in different application domains. The idea is very simple - if something is defined already and checked successfully and has been used in practice, perhaps with some adjustments it can be used for another developer's needs. This idea is implemented and used broadly in object-oriented software engineering through business objects and related patterns, shown in more detail for example in Batanov and Arch-int, 2003. We propose here an extension of this idea introducing the notion of Ontological Business Object Pattern (OBOP). An OBOP is an ontology-based description of a business object that presumably will be included as a working object in the object-oriented software system. We actually rely on the fact that there are a great number of ontological descriptions of concepts (objects) in different problem domains, existing already in a research study by Johansson (1998) and available from ontology library systems such as WebOnto, Ontolingua, DARPA Agent Markup Language (DAML), SHOE (Simple HTML Ontology Extensions), etc.

We use the DAML ontology library and SHOEntity libraryin our work, more specifically their catalogs of ontologies, which are available in XML, HTML and DAML formats. Here classes are called categories and these categories constitute a simple "is-a" hierarchy while the slots are binary relations. The relations between instances or between instances and data are allowed to have any number of arguments as cited by Noy et al. (2000). What the developer should do at this phase is to select the suitable ontology for the respective problem domain. Figure 8 shows an example of how available ontological description for our particular problem domain can be considered as OBOP.



Class

- **Person** (age*, **doctoralDegreeFrom***, emailAddress*, mastersDegreeFrom*, researchInterest*, undergraduateDegreeFrom*)
  - **Student** (advisor*, takesCourse*)
    - GraduateStudent ()
    - UndergraduateStudent ()
  - Worker ()
    - AdministrativeStaff ()
      - Chair ()
      - ClericalStaff ()
      - Dean ()
      - Director ()
      - SystemsStaff ()

```
<Class ID="Student">
  <label>student</label>
  <subClassOf resource"=#Person/ ">
</Class>
<Property ID="takesCourse">
  <label>is taking</label>
  <domain resource="#Student" />
  <range resource="#Course" />
</Property>
<Property ID="doctoralDegreeFrom">
  <label>has a doctoral degree
from</label>
  <domain resource"=#Person/ ">
  <range resource"=#University/ ">
```

Figure 8 Ontological class hierarchy used as a pattern

Representation of ontology specifications is standardized in a form of object description and this provides a great advantage for software developers. For example, the ontological description shown in Figure 8 is found in the ontology library and has a structure, which can be used by the developer directly as not only class hierarchy but as a structured content of respective classes. Therefore, this description can be considered as OBOP. Within this pattern the concept (object) "student" possesses exactly the properties (attributes) necessary for the system under development. We can say the same for the root concept (object, class) "person". Moreover, in the ontology the attributes themselves are treated as concepts (objects) just like in object orientation, which means that we can follow and extract the description of all objects which we are interested in within the class hierarchy. More specifically, the relationships are formalized through the arguments (attributes), which are either types (Atomic ADTs) or categories (objects, classes). If the argument is a category, any subcategory of that category is also valid in the ontology. In addition, the relationship between any two concepts (objects) is a commitment and all commitments are specific to objects and phenomena in one particular domain as mentioned by Chandrasekaran et al. (1999). Figure 9 shows that if a relationship exists between two concepts (objects), they are both objects in our problem domain (for example, “takesCourse” has a relationship with argument1 “Student” and argument2 “Course”, which should be considered as working objects). The phenomenon “age” is related to argument1 “Person” and argument2”NUMBER” (type or Atomic ADT), which is different from the first relation ("takesCourse"), so in this case, we should consider the “age” only as an attribute of “Person”.It is clear, however, that this attribute “age” will be valid also for objects “Student” and “GraduateStudent” because of the generalization/specialization relationship.

Relation	Argument 1	Argument 2
takesCourse	Student	Course
Age	Person	.NUMBER
Email Address	Person	.STRING
Head	Organization	Person
UndergraduateDegreeFrom	Person	University
MastersDegreeFrom	Person	University
DoctoralDegreeFrom	Person	University
advisor	Student	Professor

Figure 9 the relations pattern

3.6 Mr-model: Reduced matrix model

In order to emphasize the necessity of this model we will review what information the developer has up to this point working with the models described above:

1. Set of objects in the problem domain PD = {O_1, O_2, O_3,.., O_a} with their names and relationships, extracted from the T-model by an ontological engine (in our case CORPORUM OntoExtract). The result is represented in the Mf-model.
2. Set of objects FOE = {O_1, O_2, O_3,.., O_b} with their names and relationships as a result of applying an ontological engine (in our case OntoExtract) on a Use Case-based system functionality. The result is represented in a part of the UO-model.
3. Set of objects FVT = {O_1, O_2, O_3,.., O_c} with their names, relationships and functions as a result of applying an ontological engine (in our case VisualText) on a Use Case-based system functionality. The result is represented in the other part of the UO-model.
4. Set of objects BOP = {O_1, O_2, O_3,.., O_d}with their names, relationships (including hierarchical information) and functions as a result of searching for OBOPs in ontology libraries (in our case DAML and SHOEntity). The result is represented in the DF-model. Figure 10 shows in graphical form, although far from precise, the existing situation. Without a doubt all objects are within the system problem domain but on one hand their number is still large (this is true even for relatively simple systems) and they are defined from different perspectives (different models are used).

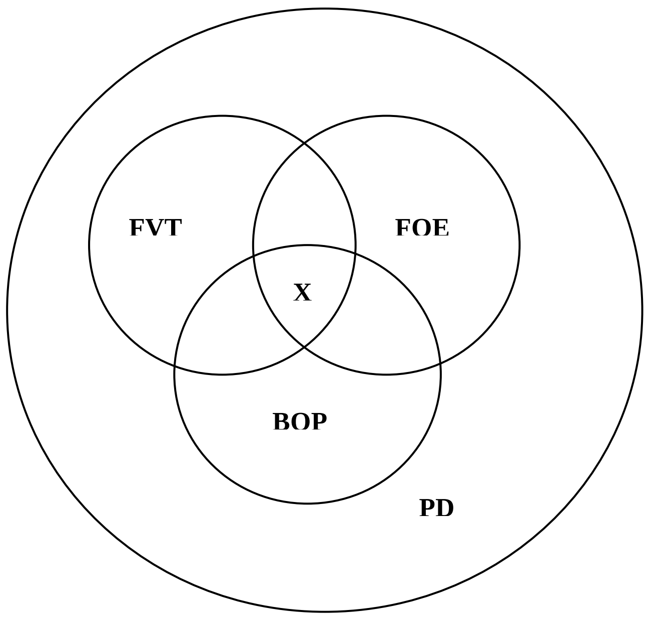
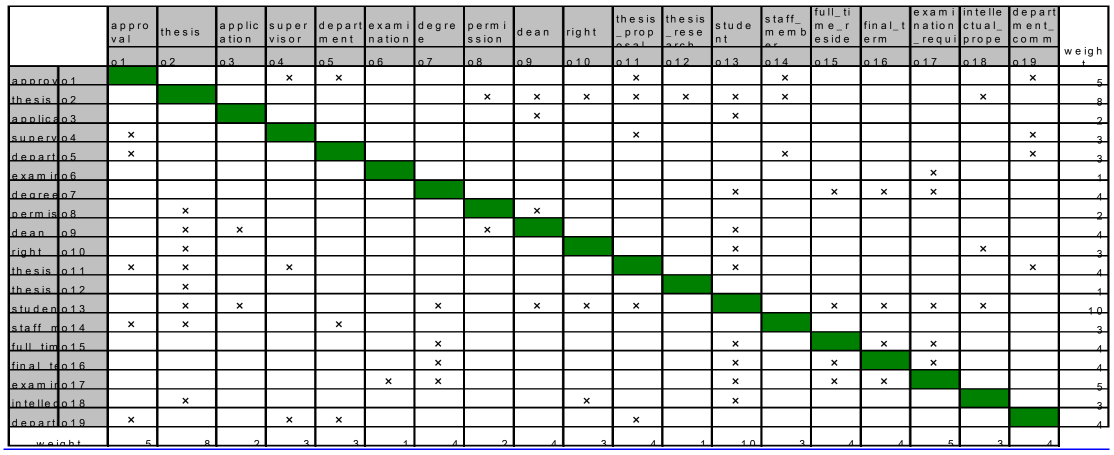


Figure 10 Integration procedure

Our presumption, based on a number of experiments, is that the basic objects, which will play a substantial role in ensuring the system functionality, will appear in all of the above models regardless of the perspective. This practically means that we can apply a simple integration procedure - intersection of the above sets - to identify those objects In Figure10 the resulting area is X, or

$$X = PD \cap FOE \cap FVT \cap BOP$$

Applying the above procedure the developer has the opportunity reduce the number of objects, which he/she is interested in, or to transform the full matrix model (Mf-model) to reduced matrix model (Mr-model). Along with this, the developer can use another quantitative technique for reducing the number of objects using the already mentioned parameter weight, assigned to each object during the process of creating the Mf-model. This technique is based on a simple assumption, which is well supported by our experiments – an object with higher weight would play a significant role in the system and, therefore, can be identified as a separate object (ADT). At this stage of research, to determine the degree of weight as low or high we refer to our experiments, which qualitatively show that the border is somewhere about 4 or 5 and a value above 10 should be definitely considered as high weight. For objects with low weight, there are two options, either to consider them as complementary objects, to be included as attributes or references in other objects, or to rename and consider them as separate objects. The final decision should be taken by the developer. The resulting Mr-model will look like the matrix shown in Figure 11.



		approval	thesis	application	supervisor	department	examination	degree	permission	dean	right	thesis_proposal	thesis_research	student	staff_member	full_time_reside	final_term	examination_requi	intellectual_prope	department_comm	weight
		o1	o2	o3	o4	o5	o6	o7	o8	o9	o10	o11	o12	o13	o14	o15	o16	o17	o18	o19	
approv	o1	🟩			×	×						×			×					×	5
thesis	o2		🟩						×	×	×	×	×	×	×				×		8
applica	o3			🟩						×				×							2
superv	o4	×			🟩							×								×	3
depart	o5	×				🟩									×					×	3
exam in	o6						🟩											×			1
degree	o7							🟩						×		×	×	×			4
permis	o8		×						🟩	×											2
dean	o9		×	×					×	🟩				×							4
right	o10		×								🟩			×					×		3
thesis	o11	×	×		×							🟩		×						×	4
thesis	o12		×										🟩								1
studen	o13		×	×				×		×	×	×		🟩		×	×	×	×		10
staff m	o14	×	×			×									🟩						3
full tim	o15							×						×		🟩	×	×			4
final te	o16							×						×		×	🟩	×			4
exam in	o17						×	×						×		×	×	🟩			5
intellec	o18		×								×			×					🟩		3
depart	o19	×			×	×						×								🟩	4
weight		5	8	2	3	3	1	4	2	4	3	4	1	10	3	4	4	5	3	4	

Figure 11 Mr-model matrix

3.7. C-model: Class model

The C-model is the goal of preliminary analysis of object-oriented systems. This is the well-known class hierarchy representation, including some initial but significant relationships for the system functionality contents of objects – data and behavior (functions, operations). We stress on the word initial here to emphasize the fact that the analysis is far from over yet. The developer should continue applying the conventional analysis models, methods and techniques on the C-model, which can lead to substantial changes, including adding new objects, deleting some objects, adding or removing some elements of the included objects, etc. The C-model can be represented graphically using different tools such as Rational Rose (class diagrams), textually using either some natural language or pseudo programming language, and finally using some highly structured tag-based language.

3.8. XML-model: XML object model

This model is optional but extremely useful for exchanging analysis and design information through the Web for supporting collaborative work. It represents the C-model using the third option mentioned above and, more specifically XML (eXtensible Markup Language) as a language-specification for computer-readable documents or a metalanguage, which can be used as a mechanism for representing other languages in a standardized way as mentioned by Klein (2001). In our case we use W3C XML Schema, which allows the highest flexibility in describing all necessary elements of any object hierarchy on one hand and the details of object model on the other, Figure 12 illustrates a part of the XML-based description of the object "student" as an ADT.

```
    <elementtype name="student">
       <empty/>
       <attdef name="student name"
    datatype="string"/>
       <attdef name="degree">
          <enumeration
    datatype="NMTOKEN">
             <option>Bachelor</option>
              <option>Master</option>
             <option>Doctoral</option>
          </enumeration>
       <funcdef name="getter">
       <funcdef name="setter">
          <required/>
           </funcdef>
        </attdef>
</elementtype>
```

Figure 12 Example of XML object model

4. Conclusion:

We believe that merging ontologies with existing methods, techniques, and tools used during the analysis phase of complex object-oriented software systems can contribute significantly to reaching better decisions, with a positive effect on all the subsequent phases of the development process. This paper describes a methodology for supporting the high-level analysis phase of object-oriented software engineering using ontologies for identification of system objects. Eight models are introduced and briefly described in the paper as a part of this methodology. We believe that these models and the process of their transformation can help developers of complex object-oriented software systems to: (a) transform user requirements (represented as text description) into an object model of the system under development based on the use of ontologies; (b) improve the existing methods and techniques for creating a specific ontology from a text description of the system problem domain, which would serve as a source for identifying the objects and their respective ADTs; (c) work out implementation techniques and tools for semi-automated or automated generating and editing of ADTs for object-oriented application software development, and (d) improve the effectiveness and efficiency of the existing methodology for high-level system analysis in object-oriented software engineering.

The research work for improving the proposed methodology is however not completed yet. A lot of work is still ahead mainly in regard to the formalization of the methods and techniques introduced so far in order to make them a part of CASE. Identification of objects and related ADTs is based on ontology analysis but if for a given problem domain such ontology still does not exist, the developers should be ready to create this ontology themselves including a description of well selected ontological business object patterns. In any case we strongly believe that using ontologies has a great potential for analysis and design of complex object-oriented software systems.

References:

Gruber, T. R (1993) ‘A translation approach to portable ontology specifications. Knowledge Acquisition 5, 199-220.

Gruber, T. R. (1994) Towards Principles for the Design of Ontologies Use for Knowledge Sharing. In Proceedings of IJHCS-1994, 5 (6) (1994), 907-928.

Helfin, J. and Volz, R. Volz and Dale, J. (2002), “Requirements for a Web Ontology Language. W3C Working Draft”.

Deridder, D. Deridder, and Wouters, B. (1999) “The Use of Ontologies as a Backbone for Software Engineering Tools”, Programming Technology Lab, Vrije Universiteit Brussel, Brussels, Belgium.

Swartout, W. (1999) “Ontologies. IEEE Intelligent Systems January/February”, pp. 18-25.

Weiss, M. A. Weiss, (1993) Data Structures and Algorithm Analysis in C. Benjamin/Cummings Publishing Company, Florida International University, Redwood City, CA.

Manola, F. (1999) Technologies for a web object model. IEEE Internet Computing January-February, pp. 38-47.

Engles, R. (2001) Del 6: CORPORUM – OntoExtract ontology extraction tool, On-To-Knowledge: Content-driven knowledge management tools through evolving ontologies. IST project IST-1999-1032, On-To-Knowledge.

Engles, R. H. P. And Bremdal, B. A. Jones, and R. Jones, (2001) CORPORUM: a workbench for the semantic web. EXML/PKDD workshop, CognIT a.s.

Gil, Y. and Ratnakar, V. (2002), A comparison of (semantic) markup languages. Proceedings of the 15th International FLAIRS Conference, Special Track on Semantic Web, Pensacola, FL.

Batanov, D. N. and Arch-int, S. (2003) towards construction of business components: an approach to development of web-based application systems, In: Peckham J and Lloyd SJ (eds) Practicing Software Engineering in the 21st Century. IRM Press, pp. 178-194.

Benett, S. and McRobb. S. and Farmer.R. (1999) Object-Oriented System Analysis and Design Using UML. McGraw-Hill, International Editions 2000, London.

Johanson, I. (1998) Pattern as an ontological category, In: Guarino N (ed), Formal Ontology in Information Systems. IOS Press, Amsterdam, Netherlands, pp. 86-94.

Noy, N.F. and Sintek, M. and Decker. S et al., (2001) Creating semantic web contents with Protégé-2000. IEEE Intelligent Systems March/April, pp. 60-61.

Chandrasekaran, B. and Josephson. J. R. and Benjamin, V. R. (1999) what are ontologies, and why do we need them? IEEE Intelligent Systems, 14(1): 20-26.

Klein, M. (2001) XML, RDF and relatives. IEEE Intelligent Systems March/April, 2001, pp. 26-28.