Ontologies and Object models in Object Oriented Software Engineering

Dr. Waralak V. Siricharoen
phone: 6601-696-6425; fax: 6602-882-3783
e-mail: lak_waralak@ yahoo.com, waralak_von@utcc.ac.th
University of the Thai Chamber of Commerce (UTCC), Bangkok, Thailand

Abstract—**This paper is to clarify ontologies in knowledge base compare with object models in object oriented software engineering. Ontology itself has the concept which is the foundation of knowledge base; on the other hand The object model is the center of object oriented software engineering. Because ontologies are closely related to modern object-oriented software design, it is natural to adapt existing object-oriented software development methodologies for the task of ontology development. Selected approaches originate from research in artificial intelligence; knowledge representation and object modeling are presented in this paper. Some issues mentioned in this paper are related with their connection; some are addressed directly into the similarities or differences point of view of both. This paper also presents the available tools, methods, procedures which show the corporation with object modeling and ontologies.**

Index Terms— **Software Engineering, Object Model, Object Oriented Development, knowledge base, Ontologies.**

I. INTRODUCTION

The object oriented paradigm is the framework in software engineering, influencing all effort in information science. It is one of the main objectives of the software engineering discipline[18]. Object models are different from other modeling techniques because they have merged the concept of variables and abstract data types into an abstract variable type: an object. Objects have identity, state, and behavior and object models are built out of systems of these objects. To make object modeling easier, there are concepts of type, inheritance, association, and possibly class [1]. Object modeling's focus on identity and behavior is completely different from the relational model's focus on information [14]. Ontology is well known as description of declaration and abstract way the domain information of the application, it involved with vocabulary and how to constrain the use of the data [6] and they are used widely in the semantic web approach, which requires a significant degree of structure. In the area of ontology the concept have been supplemented above which allow expressing the similarity of concept in ontology with object (atom) in object oriented.

Ontologies themselves are rising as an important tool for coping with very great, compound and various sources of information. It has also been known that ontologies are advantageous for software engineering. Ontology representations are little known outside AI research laboratories, In contrast, commercial interest has results in ideas from object oriented programming community maturing into industry standards and powerful tools for object oriented analysis design and implementation. And this maturing standards and tools can be used for ontology modeling [1]. Ontology is formally specified models of bodies of knowledge defining concepts used to describe a domain and the relationship that hold between them.

The central objective of this paper is to acknowledge the paper is structured as follows. Section 2 introduces the definition of ontology and object model. Section 3 will be present the modeling with their structure. Section 4 will show the languages used for object model and stand for ontology. Section 5 will be emphasize on possibility development processes and tools of them and the conclusion section will be point of the similarities and differences of them.

II. DEFINITION

Ontology is actually well known in philosophy research area since 1960s, in the artificial intelligence (AI) arena, has been focused on knowledge modeling. The term ontology is used to refer to "an explicit specification of a conceptualization [of a domain] is mentioned by Tom Gruber which we are already familiar with for quite sometimes. In other words, ontology refers to a formalization of the knowledge in the domain. Ontology is the concept which is separately identified by domain users, and used in a self-contained way to communicate information. Combination of concept is the knowledge base or knowledge network. Some of the reasons why someone want to develop an ontology are to share common understanding of the structure of information among people or software agents, to enable reuse of domain knowledge, to make domain assumptions explicit, to separate domain knowledge from the operational knowledge, to analyze domain knowledge [19]

While ontologies is formally specified models of bodies of knowledge defining concepts used to describe a domain and the relationship that hold between them [6]. Object model is the mechanism of object-oriented paradigm, which is used for software engineering. In particular, the general software engineering principle of parting of concerns combined with object-oriented modeling characteristics has turned out to be very useful. The basic idea of object-orientation is the consequent application of the abstract data type concept, combining data and functionality. The abstract data type concept is applied in the context of the architecture of any object-oriented model. The Objects model in object oriented analysis and design provide a more realistic representation,

which an end user can more readily understand.. An object oriented model uses functions to model relationships of objects and the attributes [19]. An ontology structure holds definitions of concepts, binary relationship between concepts and attributes. Relationships may be symmetric, transitive and have an inverse. A minimum and maximum cardinality constraint for relations and attributes may be specifies. Concepts and relationships can be arranged in two distinct generalization hierarchies [5]. Concepts, relationship types and attribute abstract from concrete objects or value and thus describe the schema (the ontology) on the other hand concrete objects populate the concepts, concrete values instantiate the attributes of these objects and concrete relationship instantiate relationships. Three types of relationship that may be used between classes: generalization, association, and aggregation.

III. MODELING

A. Object Modeling:

The object-oriented model is based on a collection of objects. An object contains values stored in instance variables within the object. Thus objects contain objects to an arbitrarily deep level of nesting. *Attributes/properties:* objects will have at least one attribute. Possible slot types are primitive types (integer, boolean, string etc.), references to other objects (modeling relationships) and sets of values of these types. An object also contains bodies of code that operate on the object. These bodies of code are called methods. *Method/Operations:* They are attached to classes or slots and contain meta information, such as comments, constraints and default values. *Relationship/relations:* they represent the relation between objects/classes from object model (KB). Major classes of relations exist: relations combining labels(the name we tend to give to things) and concepts(the things themselves) and concepts and relations combining concepts (the part-whole relation)[13].

Objects that contain the same types of values and the same methods are grouped into *classes*. A class may be viewed as a type definition for objects. Analogy: the programming language concept of an abstract data type. The only way in which one object can access the data of another object is by invoking the method of that other object. This is called sending a message to the object. Internal parts of the object, the instance variables and method code, are not visible externally or some researchers called it as black box.

The following Fig.1 shows a simple Banking System object Model, containing classes for Head-Office, Branch, Accounts held at that Branch, and the Customers who the Accounts belong to. Object/Class represent the tangible things. For example, an object representing a *bank account*. The object contains instance variables *number* and *balance*. The object contains a method *pay-interest* which adds interest to the balance. Under most data models, changing the interest rate entails changing code in application programs. In the object-oriented model, this only entails a change within the *pay-interest* method [10]. In commonly-known object-oriented data models attributes and associations are not defined with the class specification itself [14]. Instead, class properties are first-class primitive themselves [12].

One approach for implementing objects is to have a class, which defines the implementation for multiple objects. A class defines what types the objects will implement, how to perform the behavior required for the interface and how to remember state information. Each object will then only need to remember its individual state. Although using classes is by far the most common object approach, it is not the only approach (using prototypes is another approach) and is really peripheral to the core concepts of object-oriented modeling [19].

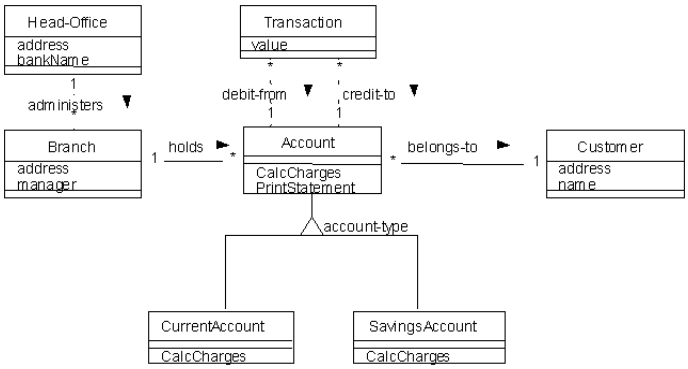

Fig. 1 Example of banking system object oriented model

Unified Modeling Languages (UML) is well known and widely used object modeling that consist of concepts /entitypes/classes in a specification hierarchy, the description of concepts by attributes which have range and relationship between concepts. UML defines several types of diagram that can be used to model the static and dynamic behaviors of a system. A UML object diagram does not define a standard set of primitive types for attributes and operation declarations; however, Object Constraint Languages (OCL) does and it is proposed that these be used for ontology modeling with UML Model an ontology as a static model consisting of a class diagram to depict the classes in the domain and their relationships, an object diagram to show particular named instances of those classes.

Conceptual (or Ontology) modeling deals with the question on how to describe in a declarative and abstract way the domain information of an application, its relevant vocabulary, and how to constrain the use of the data. Modeling languages like UML and Object Data Management Group (ODMG) have been developed for object oriented models in software engineering. Common to all of these newer models is the arrangement of concepts/entitytypes/classes in a specialization hierarchy, the description of concepts by attributes which have ranges and relationships between concepts. Concepts, relationshiptypes and attributes abstract from concrete objects or values and thus describe the schema (the ontology). On the other hand concrete objects populate the concepts, concrete values instantiate the attributes of these objects and concrete relations instantiate relationships [10].

B. Ontologies Modeling:

Ontology is a formal explicit description of concepts in a domain of discourse (*classes*/*concepts*). *Slots/properties/roles* Properties of each concept describing various features and

attributes of the concept. And *facets/role restrictions* means restrictions on slots. An ontology together with a set of individual *instances* of classes constitutes a *knowledge base*. In reality, there is a fine line where the ontology ends and the knowledge base begins.

Classes are the focus of most ontologies. Classes describe concepts in the domain [19]. For example, a class of *banking accounts* represents all *accounts*. Specific *accounts* are instances of this class. The *your own account (e.g. Waralak account)* is an instance of the class of *accounts*. A class can have *subclasses* that represent concepts that are more specific than the *superclass*. For example, we can divide the class of all *accounts* into *saving accounts*, and *checking accounts*.

Slots describe properties of classes and instances: *Saving Account* has a specific requirement; it is opened by the *bank branch*. We have slots describing the account in this example: the slot *branch* with the value *New York Fifth Avenue branch*. At the class level, we can say that instances of the class *Account* will have slots describing their *account number*, *name*, *address*, the *branch* of the account and so on.

All instances of the class account, and its subclass *Saving*, have a slot *branch* the value of which is an instance of the class *Branch*. All instances of the class *Branch* have a slot *open* that refers to all the *accounts* (instances of the class Account and its subclasses) that the branch opens an account for a particular customer.

IV. LANGUAGES AND STANDARDS

In particular, object-oriented languages like C++ or Java [14] have become the effectively standard for programming. The same holds for the analysis and design phases within a software development process, where object-oriented modeling approaches are becoming more and more the standard ones. Unified Modelling Language (UML) and Object Data Management Group (ODMG) have been developed for object oriented models in software engineering. Modeling language like UML and ODMG. The ODMG Object Model is intended to allow portability of applications among object database products. It provides a common model for these products by defining extensions to the OMG object model that support object database requirements. In particular, the ODMG model extends the OMG core to provide for persistent objects, object properties, more specific object types, queries and transactions. The basic concepts are objects, types, operations, properties, identity and subtyping. Objects have state (defined by the values of their properties), behavior (defined by operations) and identity. All objects of the same type have common behavior and properties. Types are objects so may have their own properties. A type has an interface and one or more implementations. All things are instances of some type and subtyping organizes these types in a lattice. A type definition can declare that an extent (set of all instances) be maintained for the type. Objects are instances of a type, and as such have state, behavior and identity [26].

There are lot of papers address about the object oriented standard to be used for ontology modeling. The large user community and commercial support for object oriented standards warrants the investigation of standard object modeling technique for ontology development.

For ontology standards, the most notably W3C recommends a number of semantic markup language standards as part of the semantic Web stack. Specifically, eXtensible Markup Language (XML) provides a surface syntax for structured documents but imposes no semantic constraints on the meaning of these documents. XML Schema is a language for restricting the structure of XML documents. Resource Description Framework (RDF) is a language for creating a data model for objects (or “resources”) and relations among them, providing a simple semantics for the data model. The data models are represented in an XML syntax. RDF Schema is a vocabulary for describing properties and classes of RDF resources, with semantics for generalization hierarchies of such properties and classes [25]. Finally, Web Ontology Language (OWL) adds more vocabulary for describing properties and classes. Among others, relations among classes, cardinality, equality, richer typing of properties and enumerated classes. Also now, the OWL is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema [24]. OWL succeeds the preceding effort of DAML+OIL in this area, and thus it goes beyond these languages in its ability to represent machine-readable content [28]. As following is the example of OWL used for describing banking system account register ontologies according to class and property hierarchy from DAML ontology library [22]:

Class Hierarchy

- *Category (description)*
- *Transaction (amount, category*, cleared, date, description, memo?, number?)*

Property Hierarchy

- *amount*, *category*, *cleared*, *date*, *description*, *memo*, *number*

```
<owl:Ontology rdf:about="">
  <owl:versionInfo>$Id: check-ont.daml,v 1.3 2001/07/21 22:34:26 mdean
    Exp $</owl:versionInfo>
  <rdfs:comment>checking account register</rdfs:comment>
</owl:Ontology>

<rdfs:Class rdf:ID="Transaction">
  <rdfs:comment>
    a deposit, check, or other item
  </rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#date"/>
      <owl:allValuesFrom
    rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
      <owl:cardinality>1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
…
<owl:DatatypeProperty rdf:ID="date"/>
<owl:DatatypeProperty rdf:ID="description"/>
<owl:DatatypeProperty rdf:ID="memo"/>
<owl:DatatypeProperty rdf:ID="amount">
  <rdfs:comment>in US dollars</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="number"/>
<owl:DatatypeProperty rdf:ID="category"/>
<owl:DatatypeProperty rdf:ID="cleared"/>
```

Another ontology language, with F-Logics provide a clearly defined syntax and semantics to ontologies and the representation of these knowledge models is based on a well-understood logical framework. F-Logic allows to describe ontologies, i.e. classes, the hierarchy of classes, their attributes and relationships between classes in an object oriented style way. Nowadays, the object-oriented paradigm has become the standard approach throughout the whole software development process.

Ontologies allow the specification of concepts in a domain as well as the terms used to markup content in a learning object. Shared ontologies allow for different systems to come to a common understanding of the semantics of a learning object. The present the required ontology model including the formal expression of ontology, object model, mapping to XML representation and the corresponding system architecture for binding web services. This can see that ontology model work well together almost the same with object model because it is the fundamental of system architecture [9].

V. DEVELOPMENT TOOLS

For several years ago, object-oriented development was an interesting concept. Every development tool on the shelf promotes itself as object oriented. Object orientation is a software development success story, and it is here to stay. Despite the tremendous success of objects, developers and development organizations are still low on the object oriented learning curve. Object oriented systems development is an extension of structured programming: Object oriented development emphasizes the benefits of modular and reusable computer code and modeling real-world objects, just as structured programming emphasizes the benefits of properly nested structures. In many cases, the tools are more advanced than the expertise of the developers who use them. Although most object oriented development tools promote object reuse, there is no standard reusable object among tools (for example, C++ objects under Smalltalk). To solve this problem, the Object Management Group (OMG) came up with the Common Object Request Broker Architecture (CORBA). As the object market expands and component development takes off, more object vendors will be interested in standard objects. IBM, for example, bases its object development on its System Object Model (SOM). Other examples of distributed objects include NeXT Computer's Portable Distributed Objects, and Sun's Distributed Objects Everywhere. Microsoft is pushing Object Linking and Embedding (OLE) and its Component Object Model as its standard object development strategy. Although not object oriented (they do not support inheritance), OLE objects work with a number of application development tools. It's not a question of whether object oriented development will remain popular, but how to use objects in your application development efforts. There are more choices of object oriented tools out there than ever before. These tools all implement objects in their own special way. Moreover, the industry is still very much in the learning phase, and it's going to take a few more years before it finally sees the optimal payback from object oriented development [17]. Object orientation is much more than a way to program. It can apply across every system development activity, including requirements analysis, design, testing, and business process reengineering. Developing an Object Oriented application requires even more thought about the design than developing in the traditional structured programming environment, because the focus on future reuse requires a longer-term view during analysis and design. However, a well-stocked library of reusable components reduces the need to perform original analysis and design. Nevertheless, many experienced object oriented developers know that their application is only as good as its design. Object oriented analysis and design methodologies and the use of CASE technology are extensive in the object oriented world. Object oriented languages such as C++ is a complex language; C++ is a low-level language that operates very close to the metal. Developers must learn how to do such things as manage memory directly, access physical disk storage, and use cryptic APIs. The potential for problems is enormous, and it takes about twice as long to develop a client/server application using C++ than it does using other 4GL-driven client/server development environments. Increased productivity with object technology methods is not just a theory. The tools that surround software development are built around both in reduced development efforts as well as fewer ongoing maintenance costs concept, as well; compilers, linkers, and even language features rely on source code files as input. Version control systems mirror the file system's structure, maintaining a copy of every version of each file monitored. For decades this has been acceptable and has even become standard. Languages such as C are built upon file references in the source code via include directives. The problems are in the mid-1990s, the idea of Object- oriented development finally began to gain momentum. Languages like C++, once formerly constrained to small research projects, now became the mainstream. With this fundamental shift, the file-based system of development was dragged along as a relic of old. C++, like its predecessor, also uses file includes to resolve dependencies. Despite new techniques and languages for modeling and designing software, such as UML, in the end source files still needed to be created and linked to each other. Developers reuse existing objects through the inheritance mechanism that most object oriented tools provide. Inheritance is a critical concept of object orientation; because it lets developers inherit the capabilities (methods and data) of existing objects. This lets developers maximize the use of application objects.

For knowledge engineering methodology for developing an ontology, there are some fundamental rules in ontology design. These rules may seem rather dogmatic. However, these rules can often help in making design decisions [28]:

- There is no one correct way to model a domain – there are always viable alternatives. The best solution almost always depends on the application that you have in mind and the extensions that you anticipate.
- Ontology development is necessarily an iterative process.
- Concepts in the ontology should be close to objects (physical or logical) and relationships in your domain of interest. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe your domain.

Deciding what we are going to use the ontology for, and how detailed or general the ontology is going to be, will guide many of the modeling decisions down the road. Among several viable alternatives, we will need to determine which one would work better for the projected task and which one would be more intuitive, more extensible, and more maintainable. We must also remember that ontology is a model of reality and the concepts in the ontology must reflect reality. After we define an initial version of the ontology, we can evaluate and debug it by using it in applications or problem-solving methods or by discussing it with experts in the field, or both. As a result, we will almost certainly need to revise the initial ontology. In practical terms, developing an ontology includes[28]:

- Defining classes in the ontology,
- Arranging the classes in a taxonomic (subclass–superclass) hierarchy,
- Defining slots and describing allowed values for these slots,
- Filling in the values for slots for instances.
- Creating a knowledge base by defining individual instances of these classes, filling in specific property value information and additional property restrictions.

Then we can create a knowledge base by defining individual instances of these classes filling in specific slot value information and additional slot restrictions. The idea of ontology has been welcomed by visionaries and early adopters. For example, ontology has been used in medical informatics studies, and the community produced popular tools such as Protégé ontology editor. However, it has failed to appeal to the majority users of the mainstream, at least until recently. It is said that the idea was too arcane for ordinary folks to understand. There is no standard way to do things with ontology, but so many different proprietary ways. There were not enough tools for programming ontologies and managing various aspects of the life cycle of ontologies. Recently, however, the semantic Web initiative lead by W3C has changed the ontology landscape completely. Through the initiative, researchers and developers join forces to provide standard semantics markup languages based on XML, ontology management systems, and other useful tools. An ontology-development methodology have been described for declarative frame-based systems. However, ontology development is different from designing classes and relations in object-oriented programming. Object oriented programming centers primarily around methods on classes—a programmer makes design decisions based on the *operational* properties of a class, whereas an ontology designer makes these decisions based on the *structural* properties of a class. As a result, a class structure and relations among classes in an ontology are different from the structure for a similar domain in an object-oriented program. Concepts in the ontology should be close to objects (physical or logical) and relationships in your domain of interest. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe your domain. The modern development environment has not yet fully caught up to the object-oriented shift. All of the tools still rely on a file to be the container of source code. While some contain modeling capabilities as well, the models exist as different entities than the file and its code. The tools need to evolve to support modern software development; they need to merge the model view and the code view into a single entity. While this may seem like a radical shift, and it is, the new tools will be natural to developers. Developers already expect their classes to be organized by namespace; they do not really care about which file or directory contains what code [16].

VI. CONCLUSIONS

Object technology support the ability to build applications by selecting and assembling objects from libraries. If a developer must create a missing object to meet the application's requirements, that new object may be placed in a library for reuse in future applications. For many simple systems, the developer may use the available objects to form the entire application instead of writing code. More complex development efforts require the developer to modify the objects to meet specific requirements. Ontologies are promised to bright future. In this paper we propose that as ontologies are closely related to modern object-oriented software engineering, it is natural to adapt existing object-oriented software development methodologies for the task of ontology development. This is some part of similarity between descriptive ontologies and database schemas, conceptual data models in object oriented are good applicant for ontology modeling, however; the difference between constructs in object models and in current ontology proposals which are object structure, object identity, generalization hierarchy, defined constructs, views, and derivations. We can view ontology design as an extension of logical database design, which mean that the training object data modelers could be a promising approach. An ontology use the equivalent of database schema But ontology represent a much more richer information model than normal database schema, and also a richer information model compared to UML class/object model.

Ontology is different from object-oriented modeling (represented in UML) in several ways. First, the most profound difference is that the ontology technology is theoretically found on logic. While ontology allows automated reasoning or inference, object-oriented modeling does not. Another difference is the treatment of properties; while the ontology technology treats properties as the first-class citizen, the object-oriented modeling does not. That is, while the ontology technology allows inheritance of properties, the object-oriented modeling does not. While the ontology technology allows arbitrary user-defined relationships among classes (a type property), the object-oriented modeling limits the relationship types to the subclass-superclass hierarchical relationship. While the ontology technology allows adding properties to relationships such as symmetry, transitivity, and inversion so that they are used in reasoning, the object-oriented modeling does not. While the ontology technology allows multiple inheritances among classes and also among properties, the object-oriented modeling allows only single inheritances. Despite theses differences, object-oriented modeling and UML are accepted as a practical ontology specification, mostly because of their

wide-spread use in industry and the multitude of existing models in UML. There is an on-going effort to add logic capability to object-oriented modeling, represented by OCL (Object Constraint Language).It is very make sense to attempt to highlight similarities as well as significant differences in the approach. Ontologies are meant to describe and explain the world, while object model (database) are meant to describe that part of the world whose representation has to be managed for some application purpose. Overcoming differences is a meaningful way to benefit one domain with results from the other domain. The bottom line is that object-oriented software development methodologies show promise as a basis for ontologies methodologies.

Ontology is not meant to replace various software technologies in the procedural computation category, such as Java, SQL, data mining, statistics, etc. Instead, ontology brings most value when it is used in combination with such procedural technologies. For example, ontology cannot replace data-mining algorithms based on pattern matching or statistical techniques. However, it can help make data-mining procedures more efficient, adaptive, and smart by externalizing and organizing domain knowledge the data-mining algorithms use in ontological models. For another example, ontology cannot replace software-engineering technology using object-oriented analysis and modeling. However, it can help software-engineering tools validate the generated models by externalizing and organizing metadata of the models in ontological models. For yet another example, ontology is not meant to replace database technology for storing large-scale data sets. However, it can be used with databases to provide a conceptual view of various data sources scattered in a number of databases with an ontological model, and virtually integrate (federate) the data sources without replicating data instances.

VII. REFERENCES

[1] W. Vongdoiwang, .D. N. Batanov. (2004). *Similarities and Differences between Ontologies and Object Model*. CCCT'05 proceeding 2004. Austin, Texas.

[2] S. Cranefield, M. Purvis. (1999). *UML as an Ontology Modeling Language*. Proceeding of the IJCAI-99 Workshop on Intelligent Information Integration, Department of Information Science, University of Otago, New Zealand.

[3] O. R. Zaï ane (1995). *The Object-Oriented Model*. [Online]. Available: http://www.cs.sfu.ca/CC/354/zaiane/material/notes/Chapter1/node8.html

[4] *Object Oriented Database*. [Online]. Available: http://www.profc.udec.cl/~gabriel/tutoriales/giswb/vol1/cp4/cp4-6.htm

[5] [Online]. Available: http://www.cs.vu.nl/~mcaklein/papers/oil-xmls.pdf

[6] J. Angele, S.Staab, H. Schurr, *Object Oriented Logics for Ontologies. Draft Whitepaper Series*, Karlsruhe, Germany, 2003.

[7] N. Cullot, C. Parent, S. Spaccapietra and et. *Ontologies : A contribution to the DL/DB debat.* to appear.

[8] R. Volz, D. Oberle, R. Studer. (1999). *Views for light-weight web ontologies.* Proceeding of SAC 2003, Melbourne, Florida, USA.

[9] B. Wouters, D. Deridder, E. V. Paesschen. (2000). *The use of Ontologies as a backbone for use case management* ", This research was partially supported by Wang Global and the Brusseles Capital Region (CCOOS Project), Belgium, to appear.

[10] D. E. Jenz. (2003). *It is High Time for Pursuing the Ontology-Centric Approach*

[11] J. Heflin, M. N. Huhns. (2003). *The Zen of the Web*. Guest Editors' Introduction, IEEE Internet Computing, pp. 30-33, September-October 2003.

[12] S. Strom. (2000). *Building a Large-Scale Generic Object Model:Applying the CYC Upper Ontology to Object Database Development in Java*. [Online]. Available: www.techtrader.com.

[13] P. Mohan, C. Brooks. (2004). *Learning Objects on the Semantic Web.*

[14] ChiMu Corporation, " Object Modeling ". Foundations of O-R Mapping, [Online]. Available: http://www.chimu.com/publications/-objectRelational/part0003.html.

[15] B. Morgan. *Java and the Component Object Model.* [Online]. Available: http://docs.rinet.ru/ZhPP/ch16.htm#TypeLibrariesand-ObjectDescriptionLanguage

[16] J. Greenfield. (2004). *The Case for Software Factories*. JOURNAL3: Microsoft Architects Journal, Issue 3, July 2004. [Online]. Available: http://msdn.microsoft.com/library/default.asp?url=/library/enus/dnmaj/html/nexgen.asp

[17] Netmation Inc. (2005). *Object Oriented developments* [Online]. Available: http://netmation.com/exp0028.htm

[18] G. Engels., L. Groenewegen. (2000). *Object-Oriented Modeling: A Roadmap* [Online]. Available: www.cs.ucl.ac.uk/staff/ A.Finkelstein-/fose/finalengels.pdf

[19] N. F. Noy., D. L. McGuinness. (2001). *Ontology Development 101: A Guide to Creating Your First Ontology* [Online]. Available: protege.stanford.edu/publications/ontology_development/ontology101.pdf

[20] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. (1991). *Object-oriented modeling and design.* Englewood Cliffs, New Jersey: Prentice Hall.

[21] DAML *Ontologies by Keyword (account register).* [Online]. Available: http://www.daml.org/cgibin/hyperdaml?http://www.daml.org/2001/06/expenses/check-ont

[22] *Object Oriented Analysis and Design Using UML* Mark Collins-Cope Objective view software development Mangazine Issue 9; Ryby, Rails, Ajax, AspectJ [Online]. Available: http://ratio.co.uk/W1.html

[23] N. F. Noy, and D. L. McGuinness. mentioned in that ontologies can build on the experience using Protégé-2000 (Protege 2000), Ontolingua (Ontolingua 1997), and Chimaera (Chimaera 2000) as ontology-editing environments.

[24] G. Wilkie. (2001). *Object-Oriented Software Engineering - The professional Developer's Guide*(*on OMG's OOA/OOD proposal*) [Online]. Available: www.idi.ntnu.no/grupper/su/courses/dif8901-/presentations2001/a01-wilkie.ppt

[25] D. L. McGuinness, F. V. Harmelen. (2004). *OWL Web Ontology Language Overview.* [Online]. Available: http://www.w3.org/TR/owl-features/

[26] M. Denny. *Language Suitability: Ontology Tools Survey, Revisited.* [Online]. Available: http://www.xml.com/pub/a/2004/07/14/onto.html?-page=2

[27] R.G.G. Cattell. (1994). *The Object Database Standard: ODMG-93, Release 1.1*, Morgan Kaufmann Publishers, San Francisco [Online]. Available:http://www.objs.com/x3h7/odmg.htm.

[28] Alphawork. *What is ontology? Frequently asked questions.* [Online]. Available: http://www.alphaworks.ibm.com/contentnr/semanticsfaqs