# 1   Ontology Engineering Environments

R. Mizoguchi

The Institute of Scientific and Industrial Research, Osaka University
8-1 Mihogaoka, Ibaraki, Osaka, 567-0047 Japan

## 1.1  Introduction

In order to discuss ontology engineering environments, we first need to clarify what we mean by ontology engineering. Ontology engineering is a successor of knowledge engineering which has been considered as a key technology for building knowledge-intensive systems. Although knowledge engineering has contributed to eliciting expertise, organizing it into a computational structure, and building knowledge bases, AI researchers have noticed the necessity of a more robust and theoretically sound engineering which enables knowledge sharing/reuse and formulation of the problem solving process itself. Knowledge engineering technology has thus developed into "ontology engineering" where "ontology" is the key concept to investigate.

There is another story concerning the importance of ontology engineering. It is the semantic web movement. Semantic web strongly requires semantic interoperability among metadata which are made using semantic tags defined in an ontology. The issue here is to build good ontologies to come up with meaningful sets of tags which are made interoperable by ontology alignment.

Although the importance of ontology is well-understood, it is also known that building a good ontology is a hard task. This is why there have been developed some methodologies for ontology development and have been built a number of ontology representation and editing tools. Among many tools [1,7,8,11,15,16,17,18,19], because of the space limitation, this chapter takes up OntoEdit[15,16], WebODE[1], Protégé[11] and Hozo[7,8] which cover a wide rage of ontology development process rather than being single-purpose tools which are covered elsewhere.

## 1.2  Factors of an ontology engineering environment

A comprehensive evaluation of ontology engineering tools is found in [22][23] in which the major focus is put on static characteristics of tools. The evaluation in this chapter is done focusing on dynamic aspects of the tools, that is, we here concentrate on characteristics of the ontology engineering process supported by the four environments. Let us enumerate factors by which an environment should be characterized.

− Development methodology
  The first key task of ontology engineering is ontology building which requires a sophisticated development methodology. However, a methodology itself is not sufficient. Developers need an integrated environment which helps them build an ontology in every phase of the building process. In other words, a computer system should navigate developers in the ontology building process according to a methodology.

  − Development process and its management
    Ontology building process is divided into several phases such as requirement specification, knowledge acquisition, conceptualization, implementation, evaluation, etc. An environment is required to manage these processes based on a sophisticated methodology.
  − Collaborative development
    Building an ontology is often done with collaboration of multiple developers who need help in orchestration of the collaborative activities.
  − Compliance with an ontology theory
    An ontology is not just a set of concepts but at least a "well-organized" set of concepts. An environment is expected to guide users to a well-organized ontology which largely depends on the environment's discipline of what an ontology should be rather than an ad-hoc classification of concepts or a frame representation. This is why an environment needs to be compliant with a sophisticated theory of ontology.

− Use of an ontology
  Ontology use is the other key task of ontology engineering. Users need also effective support in how to share ontology with others, how to use/reuse an ontology and how to build an instance model based on an ontology.

  − Compliance with WWW standard
    There are many languages standardized by W3C: XML, RDF(S), DAML+OIL and OWL, etc. The environment is required to be compliant with these.
  − Ontology/Model(instance) server
    Ontologies and instance models should be available through internet.
  − Instance definition
    Instance model building is crucial to ontology applications to real-world problems.
  − Inference service
    An inference engine is used to check the consistency of ontologies/models.

− Software level issue
  − Usability
    GUI as well as functionality is essential to the usability of the environment.
  − Architecture of the environment
    An environment should be designed in an advanced and sophisticated architecture to make it usable.
  − Extensibility
    It is good if users easily extend the environment.

Description of the four environments is done having these factors in mind in the following sections.

## 1.3  OntoEdit

OntoEdit[15, 16], professional version, is an ontology engineering environment to support the development and maintenance of ontologies. Ontology development process in OntoEdit is based on their own methodology, On-To-Knowledge [14][20] which is originally based on Common KADS[13] methodology and consists of major three steps such as requirement specification, refinement and evaluation processes. The requirement specification consists of description of the domain and the goal of the ontology, design guidelines, available knowledge sources, potential users and use cases, and applications supported by the ontology. The output of this phase is refined into a formal description in the next phase. Refinement is done usually collaboratively. In the evaluation phase, competency questions are used to evaluate if the ontology built can answer these questions.
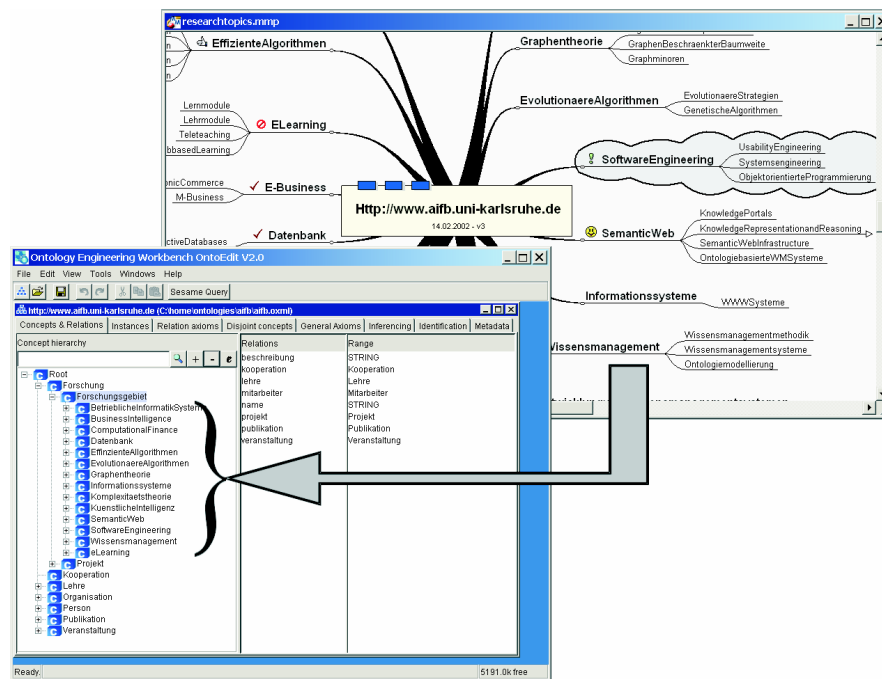


Fig. 1   Mind2Onto and translation the result to *is-a* hierarchy[16].

### 1.3.1  Requirement specification phase

Two tools, OntoKick and Mind2Onto, are prepared for supporting this phase of ontology capture. OntoKick is designed for computer engineers who are familiar with software development process and tries to build relevant structures for building informal ontology description by obtaining competency questions proposed in [4] which the resulting ontology and ontology-based applications have to answer. Examples of competency questions made by OntoKick include "which research groups exist at the institute?", "which teaching courses are offered by the institute?", etc. Mind2Onto is a graphical tool for capturing informal relations between concepts. It is easy to use because it has a good visual interface and allows loose identification of relations between concepts. However, it is necessary to convert the map into a more formal organization to generate an ontology. Fig. 1 depicts the conversion process of Mind2Onto from mind maps$^{TM}$, which is a plug-in module, to an ontology.

### 1.3.2  Refinement phase[16]

This phase is for developers to use the editor to refine the ontological structure and the definition of concepts and relations. Like most of other tools, OntoEdit em-
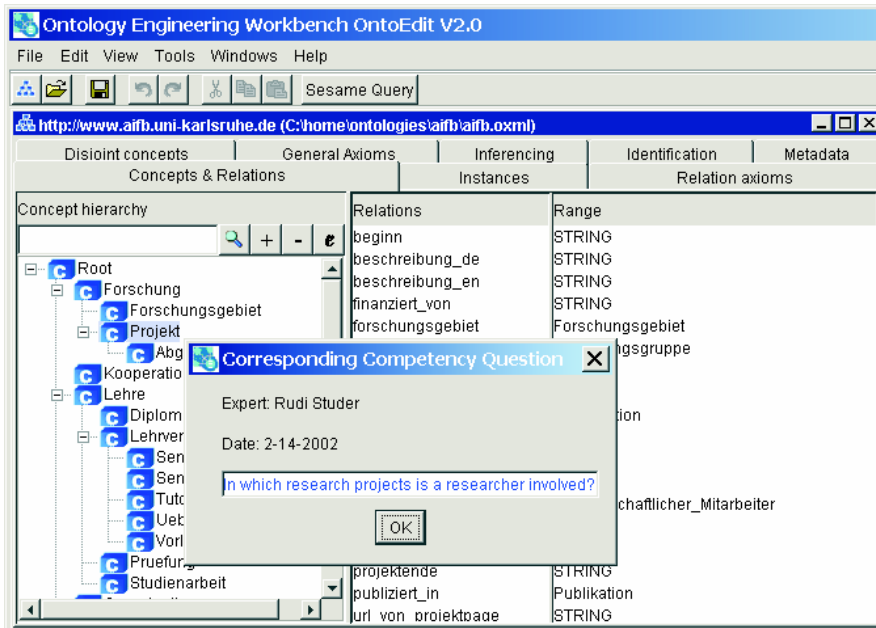


Fig. 2    Dependency management between competency questions
and concepts[16].

ploys the client/server architecture where Ontologies are managed in a server and multiple clients access and modify it. A sophisticated transaction control is introduced to enable concurrent development of an ontology in a collaborative manner. Because OntoEdit allows multiple users to edit the same class in an ontology at the same time, it needs a powerful lock mechanism of each class and devises Strict two Phase Locking protocol: S2PL to support arbitrary nested transactions.

### 1.3.3  Evaluation phase

The key process in this phase is use of competency questions obtained in the first phase to see if the designed ontology satisfies the requirements. To do this, OntoEdit provides users with a function to form a set of instances and axioms used as a test set for evaluating the ontology against the competency questions. It also provides users with debugging tools for ease of identify and correct incorrect part of the ontology. It maintains the dependency between competency questions and concepts derived from them to facilitate the debugging process(Fig. 2). This allows users to trace back to the origins of each concept. Another unique feature of this phase is that collaborative evaluation is also supported by introducing the name space so that the inference engine can process each of test sets given by multiple users. Further, it enables local evaluation corresponding to respective test sets followed by global evaluation using the combined test. Like WebODE, OntoEdit supports Ontoclean methodology to build a better *is-a* hierarchy.
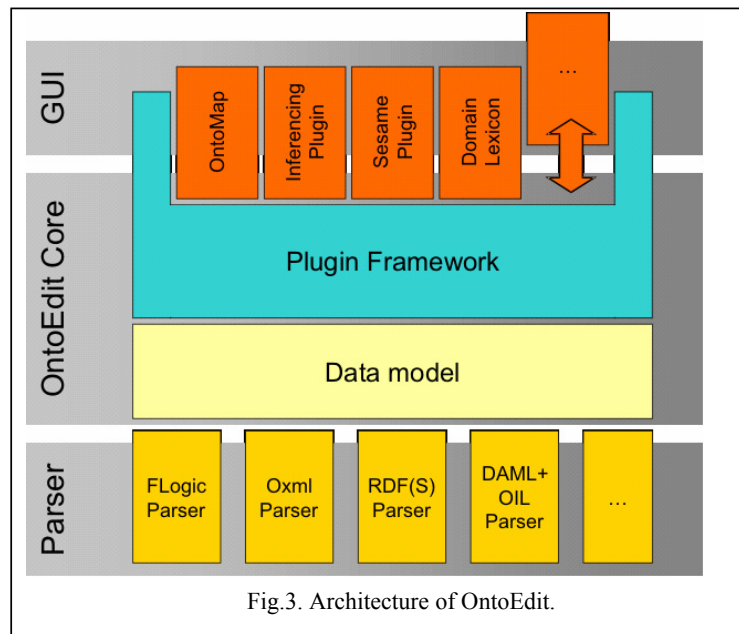


Fig.3. Architecture of OntoEdit.

### 1.3.4  Inference

OntoEdit employs Ontobroker[21] and F-Logic[6] as its inference engine. It is used to process axioms in the refinement and evaluation phases. Especially, it plays an important role in the evaluation phase because it processes competency questions to the ontology to prove that it satisfies them. It exploits the strength of F-logic in that it can express arbitrary powerful rules which quantify over the set of classes which Description logics cannot.

### 1.3.5 Architecture

Fig. 3 shows the architecture of OntoEdit consisting of three layers: GUI, OntoEdit core and Parser. Like Protégé, it employs plug-in architecture to make it easily extensible and customizable by the users. It is compliant with XML family standard in import and export the ontology.

## 1.4 Hozo

"Hozo[1]" is an integrated ontology engineering environment for building/using task ontology and domain ontology based on fundamental ontological theories[7,8]. "Hozo" is composed of "Ontology Editor", "Onto-Studio" and "Ontology Server"(Fig.4). The ontology and the resulting model are available in different formats (Lisp, Text, XML/DTD, DAML+OIL) that make it portable and reusable. One of the most remarkable features of Hozo is that it can treat the concept of **Role**. When an ontology is seriously used to model the real world by generating instances and then connecting them, users have to be careful not to confuse the **Role** such as teacher, mother, fuel, etc. with other basic concepts such as human, water, oil, etc. The former is a role played by the latter. For example, if one builds an ontology including "Mr. A is *instance-of* teacher" and "teacher *is-a* human", then when he quits the teacher job, he cannot be an instance of the class of teacher, and hence he cannot be an instance of the class human, which means he must die. This difficulty is caused by making an instance of **Role** which cannot have an instance in theory. In Hozo, three different classes are introduced to deal with the concept of role appropriately.

   **Role-concept:** A concept representing a role dependent on a context(e.g., teacher role)
   **Basic concept:** A concept which does not need other concepts for being defined(e.g., human)
   **Role holder:** An entity of a *basic concept* which is holding the role(e.g., teacher)

---

[1]  "Ho" is a Japanese word and means unchanged truth, laws or rules in Japanese, and we represent "ontologies" by the word. "Zo" means to build in Japanese.
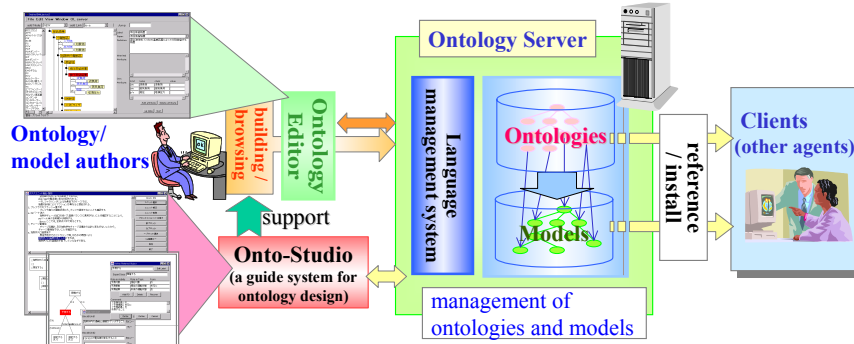
Fig.4    The architecture of Hozo

A basic concept is used as the *class constraint.* Then an instance that satisfies the *class constraint* plays the role and becomes a *role holder*. Hozo supports to define such a role concept as well as a basic concept.

### 1.4.1 Ontology development phase

**Onto-Studio** is based on a method of building task&domain ontologies, named AFM (Activity-First Method) [9]. It helps users design a domain ontology through building a task ontology from technical documents. One of the key ideas here is that task ontology provides users with the set of **Roles** played in the task context by the domain concepts which should be organized according to the roles identified by designing task ontology. Fig. 5 shows the skeletal building process of task and domain ontologies using Onto-Studio. It consists of 4 phases and 12 steps. The following outlines these 4 phases.

1. *Extraction of task-units*: In this phase, users extract **task-units** which contain only one process(action) from the technical documents.
(1) Divide the text in the technical documents into small **blocks** to extract vocabulary easier.
(2) Extract **task-units** which contain only one process(action) from these blocks.
(3) Make a flow chart called a **concrete task-flow** by combining task-units.

2. *Organization of task-activities*: In this phase, users specify the input/output of task-activities and organize the task-activities.
(4) Conceptualize **task-activities** from verbs in the task-units.
(5) Organize the task-activities in an *is-a* hierarchy.
(6) Define role-concepts, called **task-activity roles**, which appear in the input/output of these task-activities.

3. *Analysis of task-structure*: In this phase, users analyze the flow of  the task-activities, specify the flow of the objects from input to output, and define the task-context-roles.

(7) Generalize the concrete task-flows to obtain **general task-flows**.

(8) Describe the **object-flows**, which clearly express relations between inputs and outputs of the task-activities, in the general task-flows obtained above.

(9) Define the **task-context roles** on the basis of these object-flows. By task-context roles, we mean the role-concepts dependent on the whole process of a task.

(10) Extract the **domain terms** which play a task-context role.

4. *Organization of domain concepts*: In this phase, users organize domain concepts extracted in phase 3.

(11) Discriminate between the roles dependent on the domain concepts and the basic concepts.

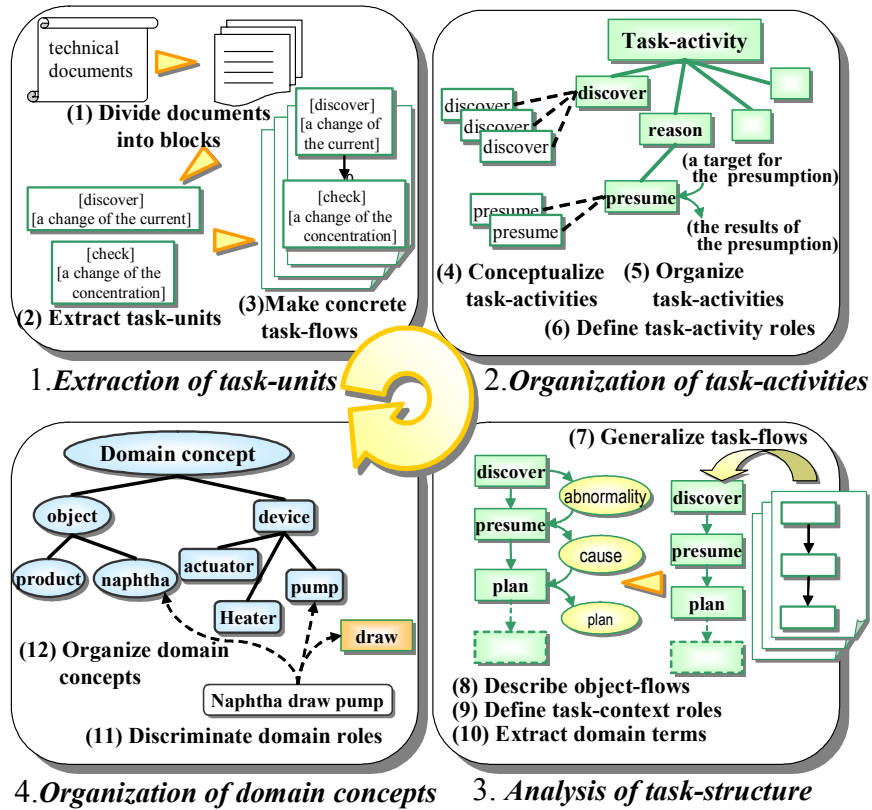(12) Organize the domain concepts in an *is-a* hierarchy.



Fig. 5    The building process of ontologies using Onto-Studio.

In practice, these steps are not done in a waterfall manner. Users can go back and forth during the process. In each step Onto-Studio provides users with graphical interfaces to help them perform the suggested procedures. The output of Onto-Studio is a rather informal representation of ontology which is in turn translated by the system into the Ontology editor representation to enable users to define ontology more rigorously.

Thanks to an Onto-Studio functionality for dependency management of many of the critical decisions made during ontology building from the technical documents, it allows developers to trace back to the previous decisions down to the original words in a sentence in the document. The system provides users with a graphical interface for tracing back the dependency chain concerning the four kinds of dependencies: original text & terms extracted, verbs(task activities) & task roles, task activity roles & task-context roles and domain concepts & task/domain roles.

### 1.4.2 Ontology definition and refinement phase

Like other editors, Ontology Editor in Hozo provides users with a graphical interface through which they can browse and modify ontologies by simple mouse operations. Users do not have to worry about so-called coding to develop an ontology. The internal representation of the ontology editor, which is hidden from users, is XML and it generates DAML+OIL code to export the ontology and instance. It
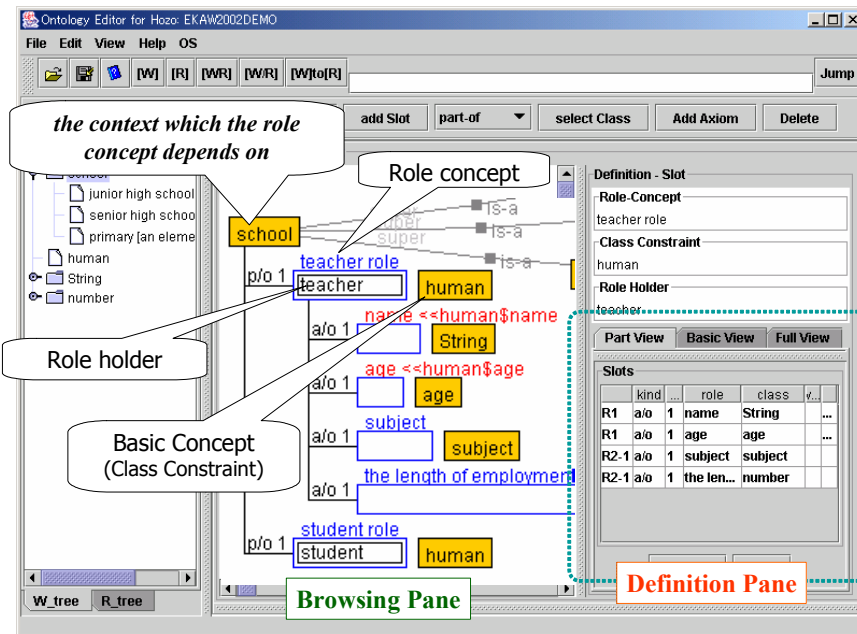


Fig. 6 GUI of Ontology Editor.

treats "role concept" and "relation" on the basis of fundamental consideration discussed in [7]. This interface consists of the following four parts(Fig. 6):

1. **Is-a hierarchy browser** displays the ontology in a hierarchical structure according to only *is-a* relation between concepts.
2. **Edit panel** is composed of a *browsing panel* and a *definition panel*. The former displays the concept graphically, and the latter allows users to define the selected concept in the *is-a* hierarchy browser.
3. **Menu bar** is used for selecting tools
4. **Tool bar** is used for selecting commands

Collaborative development of an ontology is supported in the Ontology Editor. At the primitive level, the ontology server allows users to read and copy all the ontologies and instances, but do not allow modification of them by users other than the original developer of them. Thus, unlike OntoEdit, Hozo does not allow multiple users to edit the same concept. Instead, Ontology Editor allows users to divide an ontology into several components and manages the dependency between them to enable the concurrent development of an ontology. The dependency between the component ontologies are three fold: super-sub relation(*is-a* relation), referred-to relation(class constraint) and task-domain relation. In the current implementation, the first two are taken into account. The system observes every change in each component and notifies it to the appropriate users who are editing the ontology which might be influenced by the change. The notification is done based on the 16 patterns of influence propagation analyzed beforehand. The notified users can select a countermeasure among the three alternatives: (1)to adapt his/her ontology to the change, (2)not to adapt to the change but stay compliant with the last version of the changed ontology and (3)neglect the change by copying the last version into his/her ontology. The timing of the notification is selected by the users among the two: when the editing task has been initiated and he/she requested. Fig. 7 shows a snapshot of the collaboration window.
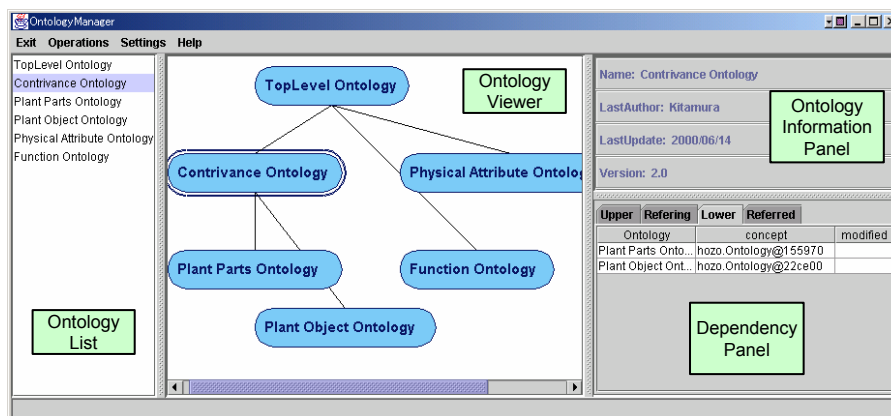


Fig. 7    Collaboration support window.

### 1.4.3 Ontology use phase

Functionality and GUI of Hozo's instance editor is the same as the one for ontology. The consistency of all the instances with the ontology is automatically guaranteed, since a user is given valid classes and their slot value restrictions by the editor when he/she creates an instance. Hozo has an experience in modeling of a real-scale Oil-refinery plant with about 2000 instances including even pipes and their topological configuration which is consistent with the Oil-refinery plant ontology developed with domain experts[10]. The model as well as the ontology are served by the ontology server and can answer questions on the topological structure of the plant, the name of each device, etc. Any ontology can have multiple sets of instances which are independent of one another.

The ontology server stores ontologies and instance models in an XML format and serves them to clients through API compliant with OKBC protocol. Ontology editor is also a client of the ontology server. Inference mechanism of Hozo is not very sophisticated. Axioms are defined for each class but it works as semantic constraint checker like WebODE.

## 1.5 WebODE

WebODE[1] is a scalable and integrated workbench for ontology engineering and
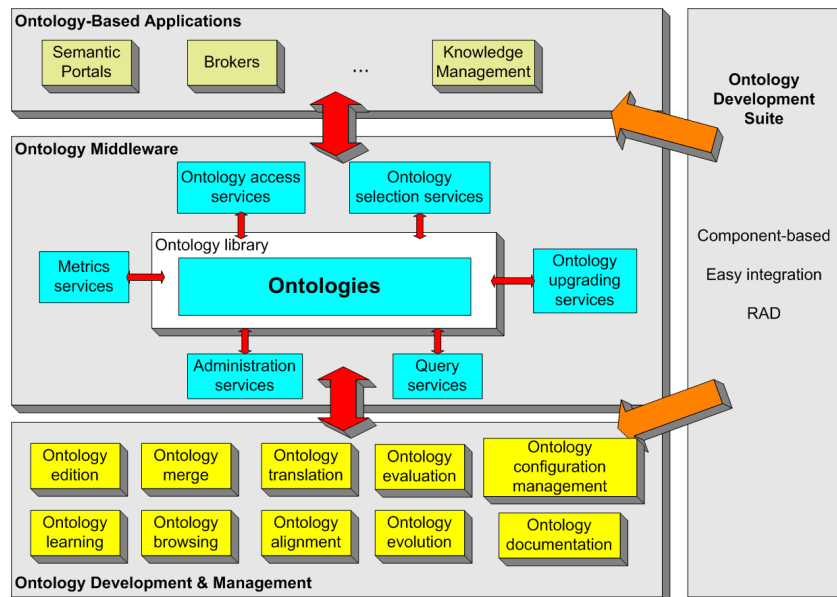


Fig. 8    Arcdhitecture of WebODE[1].

is considered as a Web evolution of ODE(Ontology Development Environment[2]). It supports building an ontology at the knowledge level, and translates it into different ontology languages. WebODE is designed on the basis of a general architecture shown in Fig. 8 and to cover most of the processes appearing in the ontology lifecycle. While Protégé and OntoEdit are based on plug-in architecture, WebODE is based on a client-server architecture which provides high extensibility and usability by allowing the addition of new services and the use of existing services. Ontology is stored in an SQL database to attain high performance in the case of a large ontology.

It has export and import services from and into XML, and its translation services into and from various ontology specification languages such as RDF(S), OIL, DAML+OIL, X-CARIN, Jess and F-Logic. Like OntoEdit, WebODE's ontology editor allows the collaborative edition of ontologies. One of the most characteristic features of WebODE is that it is based on an ontology development methodology named METHONTOLOGY[2]. Although WebODE is an integrated tool sets covering most of the activities in ontology lifecycle, it has no explicit stepwise guidance function unlike Hozo.

### 1.5.1 Ontology development phase

WebODE has ontology editing service, WAB: WebODE Axiom Builder service, inference engine service, interoperability service and ontology documentation service in this phase. The ontology editor provides users with form based and graphical user interfaces, WAB provides an easy graphical interface for defining axioms. It enables users to define an axiom by using templates given by the tool with simple mouse operations. Axioms are translated into Prolog. The inference engine is based on Prolog and OKBC protocol to make it implementation independent. Interoperability services provided by WebODE are of variety. It includes ontology access API, ontology export/import in XML-family languages, translation of classes into Java beans to enable Jess system to read them and OKBC compliance.

**ODEClean[3]**
Like OntoEdit, WebODE supports Ontoclean methodology to build a more convincing *is-a* hierarchy. The tool is called ODEClean whose architecture and its basic steps are shown in Fig. 9. Ontology for Ontoclean is composed of top level universal ontology developed by Guarino[5], a set of meta-properties and Ontoclean axioms which are translated into Prolog to be interpreted by WebODE inference engine. It is given to the ODEClean which works on the basis of it.

The collaborative editing of an ontology is supported by a mechanism that allows users to establish the type of access of the ontologies developed through the notion of groups of users. Synchronization mechanism is also introduced to enable several users to safely edit the same ontology. Ontologies are automatically

documented in different formats such as HTML tables with Methontology's intermediate representations, HTML concept taxonomies and XML.

### 1.5.2 Ontology use phase

To support the use process of ontology, WebODE has several functionalities. Like Hozo, it allows users to have multiple sets of instances for an ontology by introducing instance sets depending on different scenarios, and conceptual views from the same conceptual model, which allows creating and storing different parts of the ontology, highlighting and/or customizing the visualization of the ontology for each user. WebPicker is a set of wrappers to enable users to bring classification of products in the e-Commerce world into WebODE ontology. ODEMerge is a module for merging ontologies with the help of correspondence information given by the user. Methontology and ODE have been used for building many ontologies including chemical ontology[2].
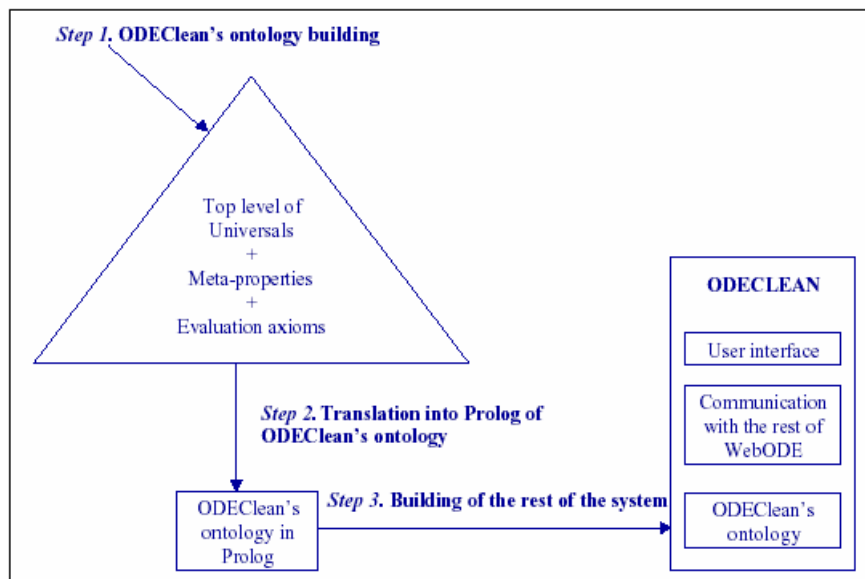


Fig. 9 Architecture of ODEClearn and its Steps[3].

## 1.6 Protégé-2000

Protégé-2000[11] whose architecture is shown in Fig. 10 is strong in the use phase of ontology: Use for knowledge acquisition, merging and alignment of existing ontologies, and plug-in new functional modules to augment its usability. It has been used for many years for knowledge acquisition of domain knowledge and for domain ontology building in recent years. Its main features include:

(1) Extensible knowledge model to enable users to redefine the representational primitives.
(2) A customizable output file format to adapt any formal language
(3) A customizable user interface
(4) Powerful plug-in architecture to enable integration with other applications

These features make Protégé-2000 a meta-tool for domain model building, since a user can easily adapt it to his/her own instance acquisition tool together with the customized interface. It is highly extensible thanks to its very sophisticated plugin architecture. Unlike the other three, Protégé-2000 assumes local installation rather than use through internet using client/server architecture. Its knowledge model is based on frame similar to other environments. Especially, the fact that Protégé-2000 generates its output in many ontology languages and its powerful customizability make it easy for users to change it to an editor of a specific language. Fig. 11 shows a snapshot of the definition of a class of RDFS which is defined as a subclass of standard class of Protégé. This "meta-tuning" can be easily done
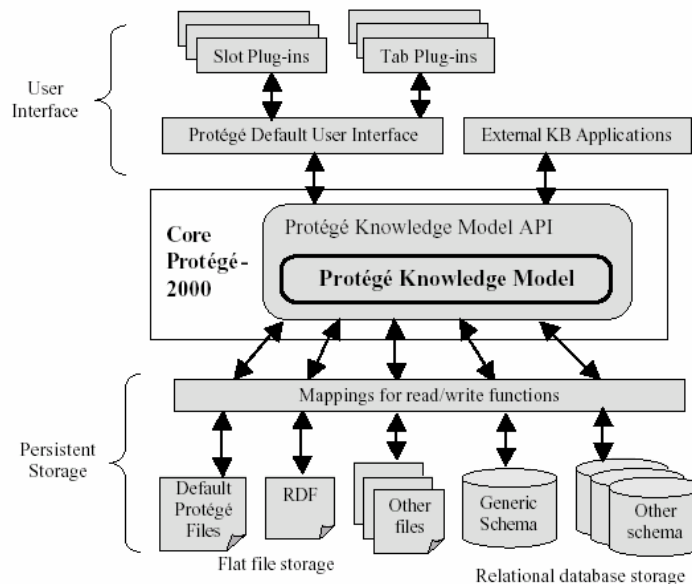


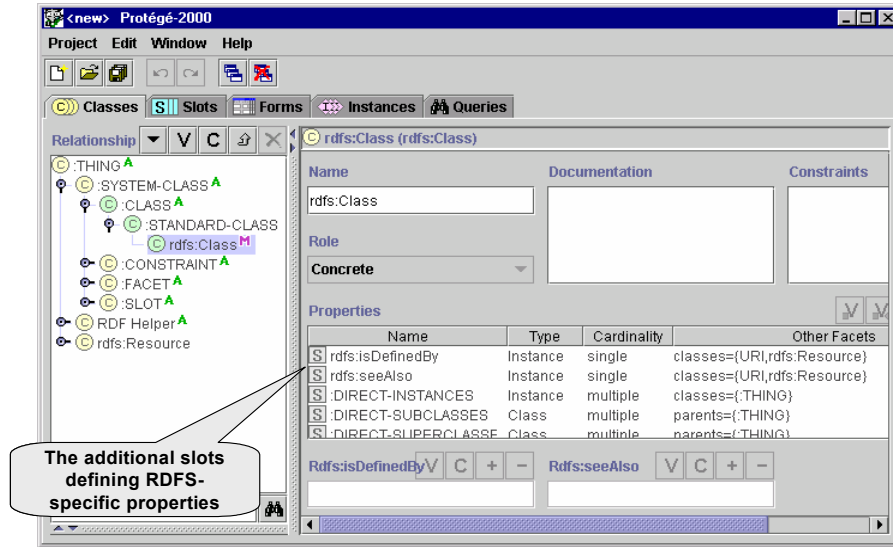Fig. 10    Architecture of Protege2000[11].

Fig. 11 Metaclass definition window[11].

thanks to Protégé's declarative definition of all the meta-classes which play a role of a template of a class.

Protégé has a semi-automatic tool for ontology merging and alignment named PROMPT[12]. It performs some tasks automatically and guides the user in performing other tasks. PROMPT also detects possible inconsistencies in the ontology, which result from the user's actions, and suggests ways to remedy them(Fig. 12).
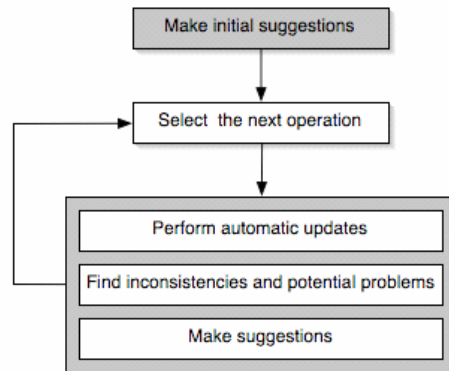


Fig. 12 Control flow of PROMPT[12]

## 1.7 Other environments

Let us briefly discuss other environments. KAON: Karlsruhe Ontology and Semantic Web framework is a sophisticated plug-in framework with API and provides services for ontology and metadata management for E-Services[17]. Its main focus is put on the enterprise application in the semantic web age. WebOnto[18] is a support tool for ontology browsing, creation and editing. When used with Tadzebao[18], a support system on discussions on ontologies, its utility is maximized because they collectively realize a collaborative development of an ontology. OilEd[19] is also an ontology editor mainly for showing how reasoning functionality is used to check the consistency of an ontology.

## 1.8 Comparison and discussion

The four environments are compared according to the factors presented above. Table 1 summarizes the comparison.

(1) **Development process management**

Philosophy of supporting ontology development is partly based on viewing an ontology as a software product. Common features of OntoEdit and WebODE include management of the well-known steps in software development process, that is, requirement specification, conceptual design, implementation and evaluation. While WebODE is mainly based on the software development process, OntoEdit on the On-To-Knowledge methodology. Protégé has no such a methodology.

The above approach is characterized as macroscopic in the sense that it manages steps of a rather large grain size in the developmental process. On the other hand, Hozo's process management is microscopic in the sense that it tries to manage finer grained activities such as what types of concepts should be focused on in a phase in the total process. It does not take care of macroscopic management of ontology development process as software development process assuming the first half of such a process has been done when a technical document has been selected. This is meaningful because Onto-Studio in Hozo supports development of task and domain ontologies from a technical document such as an operation manual of a machine and hence, in most cases, the document itself specifies the scope and goal of ontology.

The Common feature of OntoEdit and Hozo includes management of dependencies between intermediate data to enable users to retrospect the development history. This function is crucial in the case of large ontology development, since it often happens that developers need to trace back to the original data on which a resulting conceptualization, organization, or attribution depends. What OntoEdit does is to manage the dependency between competency questions and ontology. Because competency questions explicitly represent the requirement to the ontology, it can correctly navigate users back to the origin of each concept in the ontology. Especially, data-dependency management in Hozo is thorough. It allows users to trace back a long chain of dependencies between many kinds of intermediate decisions from the final ontology to the words in a sentence in the original source document following the derivation dependency chain. Onto-Studio manages four kinds of dependencies as described above.

The other two environments have no such function.

(2) **Collaboration**

Collaboration occurs in two different ways: (1)Multiple persons are involved in building a module of ontology and (2)Collaboration in building multiple modules of ontologies assuming the merge of all the modules into one ontol-

ogy later on. In the case of (1), because multiple persons might modify the same class at the same time, transaction control is one of the main issues in supporting collaboration. OntoEdit and WebODE take this approach. On the other hand, Hozo takes (2) in which the system only has to take care of the dependencies between the partial ontologies because each of the modules is taken care of by one person. Transaction control is not an issue in the case of (2). When building a large ontology, (2) is very useful because it allows users the concurrent development of an ontology. To make the latter approach feasible, however, the system does need to provide developers with relevant information of changes done in other ontologies developed by others which might influence on the ontology they are developing. Hozo is designed to cope with all the possible situations developers encounter by analyzing possible patterns of influences propagated to each partial ontology in a different module according to the type of the change. Both approaches look different but they are complementary. The former can be incorporated in the latter. Protégé-2000 has no function of supporting collaboration.

(3) **Theory-awareness**

Ontology building is not easy. This is partly because a good guideline is not available which people badly need when articulating the target world and organizing a taxonomic hierarchy of concepts. An ontology engineering environment has to be helpful also in this respect. WebODE and OntoEdit support Guarino's Ontoclean method[5]. Guarino and his group have been investigating basic theories for ontology for several years and have come up with a sophisticated methodology which identifies inappropriate organization of *is-a* hierarchy of concepts. Developers who develop an ontology based on their intuition tend to misuse of *is-a* relation and to use it in more situations than are valid, which Guarino called "*is-a* overloading". Ontoclean is based on the idea of meta-property which contributes to proper categorization of concepts at the meta-level and hence to appropriate organization of *is-a* hierarchy.

OntoEdit and WebODE way of ontology cleaning can be said that post-processing way. On the contrary, Hozo tries to incorporate the fruits of ontological theories during the development process. One of the major causes of producing an inappropriate *is-a* hierarchy from Guarino's theory is lack of the concept of **Role** such as teacher, mother, food, etc. which has different characteristics from so-called basic concepts like *human, tree, fish, etc*. Ontology editor in Hozo incorporates a way of representing the concept of *Role* and Onto-Studio guides developers to identify role concepts correctly and not to confuse them with basic concepts.

Use of multiple inheritance is another source of producing inappropriate *is-a* hierarchy. It is harmful especially when a model instantiated by the ontology is seriously used. According to the proper definition of *is-a* link, it should propagate downward essential property of the thing to represent its identity which is necessarily unique. This apparently forbids any concept to have multiple inheritance paths. Ontology editor in Hozo does not allow mul-

tiple inheritance. Hozo thus realizes ontology theory- compliance during the development process. Protégé- 2000 is not ontology theory aware in this sense.

(4)  **Architecture**

While WebODE and Hozo employ standardized API to the main ontology base, OntoEdit and Protégé a plug-in architecture. Both enable a module can easily added or deleted to make the environment extensible. WebODE, OntoEdit and Hozo are web-based, while Protégé is not. All the four are based on frame-like data model and have sophisticated GUI to make users free from coding using a complicated language.

(5)  **Interoperability**

OntoEdit, WebODE and Protégé have import and export functionalities from and to many of the XML-family ontology languages. Hozo only can export its ontology and model in XML and DAML+OIL.

(6)  **Ontology/Model(instance) server**

Hozo has an ontology/model server which allows agents to access the ontologies and instance models through internet. OntoEdit and Protégé have an ontology server.

(7)  **Instance definition**

Hozo and WebODE can generate multiple instance models from an ontology.

(8)  **Inference**

All the four have inference

Table 1 Comparison of the four environments.

| | Development process | | | | | Use of ontology/instance | | | | Software level | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Methodological support | Step-wise guidance | Collaboration support | Dependency management for editing an | Ontology theory awareness | Compliance to the standard | Ontology/ model server | Interoperability | Inference service | Friendly GUI | Architecture | Extensibility |
| **OntoEdit** | Yes | No | Partly. It has a sophisticated lock mechanism | Yes. | Yes. It supports Ontoclean. | High | High | High | OntoBroker | Yes | Client/server | Plugin |
| **Hozo** | Yes | Yes | Yes. Influence of each change to others are managed and suggestions for modification are provided. | Yes. Dependencies between steps down to the original document is managed. | Yes. It can deal with the role concept. Multiple inheritance is not allowed | Middle | High | Middle | Own language for constraint checking | Yes | Client/server | API |
| **WebODE** | Yes | No | Partly. Access management for each user group. | No | Yes. It supports Ontoclean. | High | Middle | High | Prolog/Jess | Yes | Client/server | API/Plugin |
| **Protégé** | No | No | No. Local installation is assumed. | No | Implicit | High | Middle | High | PAL/Jess/Fa CT/F–Logic | Yes | Standalone | Plugin |

mechanisms.

## 1.9 Concluding remarks

Ontology engineering environments are still in its early phase of development. Although some are powerful as a software tool, but many are passive in the sense that few guidance or suggestion is made by the environment. Theory-awareness should be enriched further to make the environment more sophisticated. Especially, more precise guidelines for appropriate class and relationship identification are needed. Collaboration support becomes more and more important as ontology building requirements increases. Ontology alignment is also crucial for reusing the existing ontologies and for facilitating their interoperability. Combination of the strong functions of each environment of the four would realize a novel and better environment, which suggests that we are heading right directions to go.

## Literature

1.  Corcho, O., M. Fernandez-Lopez, A.Gomez-Perez and O. Vicente, WebODE: An Integrated Workbench for Ontology Representation, Reasoning and Exchange, Prof. of EKAW2002, Springer LNAI 2473 (2002) 138-153.
2.  Fernandez-Lopez, M., A.Gomez-Perez and J. Pazos Sierra, Building a Chemical Ontology Using Methontology and the Ontology Design Environment, IEEE Intelligent Systems, 14(1) (1999) 37-46.
3.  Fernandez-Lopez, M. and A.Gomez-Perez, The integration of OntoClearn in WebODE, Proceedings of the 1st Workshop on Evaluation of Ontology-based Tools (EON2002) , held at the 13th International Conference on Knowledge Engineering and Knowledge Management EKAW 2002 (2002) 38-52.
4.  Gruninger, R. and M. Fox, The design and evaluation of ontologies for enterprise engineering, Proc. of Comparison of implemented ontology, ECAI'94 Workshop, W13, pp.105-128, 1994
5.  Guarino, N. and C. Welty, Evaluating ontological decisions with OntoClean, Communications of the ACM, 2(45) (2002) 61-65.
6.  Kifer, M. G. Lausen and J.Wu, Logical foundations of object-oriented and framw-based languages, Journal of the ACM, 42 (1995) 741-843.
7.  Kouji Kozaki, Yoshinobu Kitamura, Mitsuru Ikeda, and Riichiro Mizoguchi, Development of an Environment for Building Ontologies Which Is Based on a Fundamental Consideration of "Relationship" and "Role", Proc. of the Sixth Pacific Knowledge Acquisition Workshop (PKAW2000) (2000) 205-221.
8.  Kouji Kozaki, Yoshinobu Kitamura, Mitsuru Ikeda, and Riichiro Mizoguchi, Hozo: An Environment for Building/Using Ontologies Based on a Fundamental Consideration of "Role" and "Relationship" Proc. of the 13th International Conference Knowledge Engineering and Knowledge Management (EKAW2002) (2002) 213-218.

9.  Mizoguchi, R., M. Ikeda, K. Seta and J. Vanwelkenhuysen, Ontology for Modeling the World from Problem Solving Perspectives, Proc. of IJCAI-95 Workshop on Basic OntologicalIssues in Knowledge Sharing (1995) 1-12.
10. Mizoguchi, R. K. Kozaki, T. Sano and Y. Kitamura, Construction and Deployment of a Plant Ontology, Proc. of the 12th International Conference Knowledge Engineering and Knowledge Management (EKAW2000) (2000) 113-128.
11. Musen, M. A., R. W. Fergerson, W. E. Grosso, M. Crubezy, H. Eriksson, N. F. Noy and S. W. Tu, The Evolution of Protégé: An Environment for Knowledge-Based Systems Development, *International Journal of Human-Computer Interaction*(in press)
12. Noy, N. F. and M. A. Musen, PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment, *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000.*
13. Schreiber, G. et al., Knowledge Engineering and Management: The Common KADS Methodology, MIT Press, Cambridge, Mass. (1999).
14. Staab, S. H. –P. Schunurr, R. Studer and Y. Sure, Knowledge processes and ontologies, IEEE Intelligent Systems, Special Issue on Knowledge Management, 16(1), (2001) 26-34.
15. Sure, Y., S. Staab, J. Angele. OntoEdit: Guiding Ontology Development by Methodology and Inferencing. In: R. Meersman, Z. Tari et al. (eds.). Proceedings of the Confederated International Conferences CoopIS, DOA and ODBASE (2002) Springer, LNCS 2519, 1205-1222.
16. Sure, Y., S. Staab, M. Erdmann, J. Angele, R. Studer and D. Wenke, OntoEdit: Collaborative ontology development for the semantic web, Proc. of ISWC2002, (2002) 221-235.
17. Handschuh, S. A. Maedche, l. Stojanovic and R. Volz: KAON—The KArlsruhe Ontology and Semantic Web Infrastructure, http://kaon.semanticweb.org/
18. Domingue, J.: Tadzebao and WebOnto: Discussing, browsing and editing ontologies on the web, Proc.of the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop,   Banff, Canada, 1998.
19. Sean Bechhofer, Ian Horrocks, Carole Goble, Robert Stevens. OilEd: a Reason-able Ontology Editor for the Semantic Web. Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna. Springer-Verlag LNAI Vol. 2174, pp 396--408. 2001.
20. Y. Sure and R. Studer. On-To-Knowledge Methodology - Final Version. Institute AIFB, University of Karlsruhe, On-To-Knowledge Deliverable 18, 2002.
21. Decker, S., Erdmann, M., Fensel, D., & Studer, R. (1999). Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information, pages 351-369. In: Meersman, R., Tari, Z., & Stevens, S. (Eds.). Database Semantics: Semantic Issues in Multimedia Systems. Kluwer Academic Publisher, 1999.
22. Gomez-Perez, A., Angele, J., Fernandez-Lopez, M., Christophides, V., Stutt, A., Sure, Y., et al. (2002). A survey on ontology tools. OntoWeb deliverable 1.3, Universidad Politecnia de Madrid.
23. Duineveld, A., Weiden, M, Kenepa, B., Benjamis, R. WonderTools? A comparative study of ontological engineering tools. Proceedings of KAW99. Banff. 1999.