NEW
GENERATION
COMPUTING

Tutorial on Ontological Engineering

Part 2: Ontology Development, Tools and Languages

Riichiro MIZOGUCHI

The Institute of Scientific and Industrial Research, Osaka University

8-1 Mihogaoka, Ibaraki, Osaka 567-0047, Japan

miz@ei.sanken.osaka-u.ac.jp

Abstract   Practical aspects of ontological engineering are discussed in this part. First topic is the methodology of ontology development. Next, ontology representation languages and support tools are discussed as well as ontology alignment and merging which are becoming practically important to cope with distributed development of ontologies. We next discuss several ontologies developed thus far including large-scale knowledge bases such as Cyc, practical domain ontologies such as Enterprise ontology and gene ontology and generic ontologies such as PSL: Process Specification Language and SUO: Standard Upper Ontology. The first topic of ontology applications is the semantic web in which semantic interoperability, metadata and web service ontology are described. e-Learning is also a good application area of ontology in which LOM: Learning Object Metadata and ontology-aware authoring systems are discussed followed by conclusion.

1. Ontology Development Methodology

    1.1 Overview of the Methodologies

    [1] The methodology by Ushold and King
    [2] TOVE Methodology
    [3] METHONTOLOGY
    [4] On-To-Knowledge Methodology
    [5] AFM: Activity-First Method in Hozo
    [6] Summary

    1.2 Three-layer Model of Guidelines

    [1] Guidelines at Middle-layer
    [2] Guidelines at Bottom-layer

2. Ontology Representation Languages and Tools

    2.1 Languages

    [1] Ontolingua
    [2] RDF(S)

§1 Ontology Development Methodology [32]

Ontology development is not an easy task. It requires skills and is still an art rather than technology. People need a sophisticated methodology to help them develop an ontology. Although ontology building methodologies are not matured enough, there are some methodologies available. After a brief overview of some typical methodologies followed by a summarizing comparison of them, a set of finer-grained guidelines are presented in this section.

1.1 Overview of the Methodologies

[1] The methodology by Ushold and King [8]

The skeleton of Ushold and Kings' methodology is as follows:

1. Purpose identification
2. Building the ontology
3. Evaluation
4. Documentation

Although their methodology has been developed based on their experience in building the Enterprise Ontology,[8] it is general and applicable to building other domain ontologies. The heart of the methodology is the procedure of informal ontology development shown below.

1. Scope definition

   a. Concepts collection by brain storming
   b. Clustering of the concepts collected
   c. Refinement of the concept set by investigating what concepts are basic, what proportion is appropriate between numbers of generic and specific concepts, etc.

2. Determination of word name
   For each concept, select a natural word which has only one meaning. If there is no appropriate word for representing a concept, then create a new one.
3. Definition
   Meaning definition in an ontology is prescriptive in the sense that it should represent the meaning of a concept intended by the developers.

The informal ontology developed is translated into a formal language, Ontolingua in the case of Enterprise Ontology. The unique feature of this methodology is that it strongly recommends the utility of an informal ontology which is easily understood by many people and works as a useful specification of a formalized ontology.

[2] TOVE Methodology [42]

It was developed intended to help enterprise process modeling at Toronto University. It is composed of the following core steps:

1. Make motivating scenarios informally in order to formalize the requirement specification of the ontology.
2. Using the scenarios, formulate *competency questions* to be answered by a model built based on the ontology. The ontology must be able to provide vocabulary for expressing these questions. Axioms in the ontology should be able to characterize answers to the competency questions. The questions play the role of constraints and are used to evaluate the resulting ontology. Some examples of competency questions include:

   a. What activities have to be done to achieve the goal?
   b. Given activities at multiple time points in the future, what are the characteristics of those activities and resources at other time points?

    c. What if a task is shifted backward(forward)?

3. Extract a set of terms from the informal competency questions. And then, the terms are formalized in a formal language to put into the ontology.
4. Formalize the competency questions by defining the terms and writing axioms for interpretation of the terms. The ontology is thus augmented.
5. Establish conditions for characterizing the completeness of the ontology.

The strategy of competency question is well-accepted and used in On-To-Knowledge methodology and the guidelines by Noy.[30]

[3] METHONTOLOGY[22]
    METHONTOLOGY has been developed at Polytechnic University of Madrid and is based on IEEE standards for Developing Software Life Cycle Processes, 1074-1995. It has WebODE discussed in 2.2[2] as a support tool. Some guidelines provided are as follows:

1. Project management process
   Guidelines for planning, project control, quality control, etc.
2. Ontology development process
   Guidelines for envisioned use of the ontology, explication of the envisioned users, conceptualization of the target domain, formalization of ontology, implementation, etc.
3. Support activities
   Guidelines for knowledge acquisition, evaluation, ontology integration, documentation, version management, etc.

[4] On-To-Knowledge Methodology[37]
    It was developed at Karlsruhe University based on a two-loop architecture: Knowledge process and knowledge meta process for introducing and maintaining ontology-based knowledge management. Knowledge process is a normal knowledge use and evolution process. The knowledge meta process is a methodology of ontology development and is composed of five major steps(with 13 sub-steps) shown below. OntoEdit discussed in 2.2[1] is a support tool for this methodology.

1. Feasibility study
2. Kickoff
3. Refinement
4. Evaluation
5. Application & evaluation

[5] AFM: Activity-First Method in Hozo[23]
    AFM(Activity-First Method) is a method of building task and domain ontologies 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. It consists of 4 phases and 12 steps as follows:

1. *Extraction of task-units*:

    (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*:

    (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*:

    (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*:

    (11) Discriminate between the roles dependent on the domain concepts and the basic concepts.
    (12) Organize the domain concepts in an *is-a* hierarchy. What is built is semi-automatically translated into the Ontology Editor formulation.

In practice, these steps are done not in a waterfall manner. Users can go back and forth during the process. AFM is supported by Onto-Studio subsystem in Hozo described in 2.2 [4].

[6] Summary
AFM is a bit special in that it is mainly for task ontology development and starts the process after determining the source document from which an ontology is extracted, which implies the scope definition and purpose identification are assumed to have been already done. The other four roughly share the skeletal structure of the whole process management. When developing a large-scale ontology, the development process management becomes critical. In such a case, METHONTOLOGY and On-To-Knowledge are very helpful. To obtain an informal ontology at the early phase of development, Ushold and King's methodology is useful. TOVE methodology is the most formal among the exist-

ing ones in that it first enumerates the questions to be answered by the resulting ontology and formalize them in a formal language to use them for verification of the ontology. Its competency question strategy is popular and usable in any methodology. On-To-Knowledge methodology is a natural extension of KADS methodology[34] for knowledge bases development. It works well especially for knowledge management applications. Users could adopt all the good features of the above methodologies successfully in their ontology building processes.

1.2   Three-layer Model of Guidelines

An ontology building methodology can be composed of the following three-layer guidelines:

1. Top-layer: The coarsest level which specifies the whole building process compliant with the conventional software development process, since an implemented ontology is a kind of a computer program.
2. Middle layer: Generic constraints and guidelines which specify major steps as well as their ordering.
3. Bottom layer: The most fine-grain guidelines such as those for class identification

Unfortunately, most of the existing methodologies are concerned mainly with the top-layer, though some partially discuss topics at the middle-layer. What are the more important for novices to develop a good ontology, however, would be guidelines at the middle-layer and bottom-layer, since they directly influence the quality of the ontology developed. AFM is mainly concerned with the middle layer. Literature[30] is a good introduction to how to design an ontology with a few guidelines at the bottom-layer. The following is the author's speculation of guidelines at the middle-layer and bottom-layer.

Top-layer: the whole building process compliant with the conventional *software development process*

Middle-layer: *Generic constraints and guidelines* which specify major steps

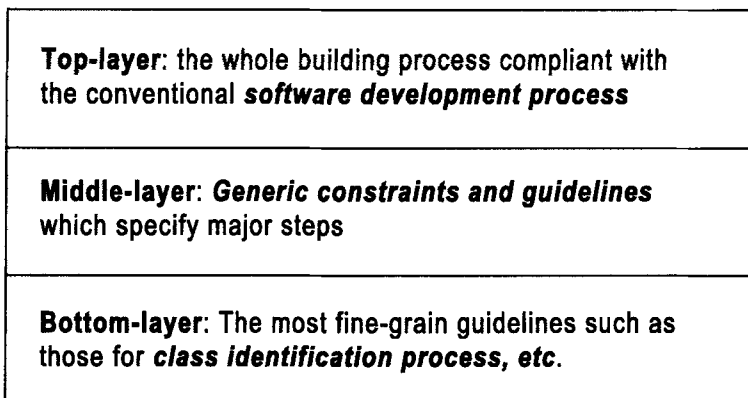**Bottom-layer**: The most fine-grain guidelines such as those for *class identification process, etc*.

Fig. 1 Three-layer Model of Ontology Building Methodology

[1] Guidelines at Middle-layer

a) Concepts rather than terms
One cannot stress the importance of this distinction too much, since people will be easily trapped by the endless terminological discussion departing from the underlying conceptual structure of the target domain. Ontology is totally independent of terminological problems.

b) Mixed and flexible strategies of Top-down, Bottom-up and Middle-out is strongly suggested. Never stick to only one of the strategies.

c) Top-level category should be identified in the early phase of the development process to govern the rest of the steps.

d) When you deal with a concept, identify its main components; that is, *"part-of"* relation as well as its main attributes. You can thus find and extend candidates of concepts to be included in the ontology.

e) Axiom writing should be done after finishing *is-a* hierarchy building and informal term definition.

f) Note that you cannot define any concept completely in theory. Therefore, do not stick to the definition of each term too much. At the best, you only can give necessary conditions of them. Term definition in the early phase can be rough. Detailed definition of a term should be done after you grasp the whole structure of the ontology, that is, after building *is-a* hierarchy.

g) Never try to seriously define a term one by one. Definition of a concept needs sufficient contextual information which is usually not available in the early phase. Terms are related to each other and could have several meanings which should be clarified by the context given.

h) Arrange and resolve the terminological issues(how to name a concept) at the last step.

i) When you find the necessity to define more than one meaning for one term, then you are facing the terminological problem. Each term should correspond to exactly one concept in onotology, since you are not building a dictionary, but a well-organized conceptual structure. Each term is only a label of the concept. You of course can build a dictionary after building ontology.

j) Put a higher priority on *is-a* hierarchy construction than term definition. Carefully designed *is-a* hierarchy gives you a correct context to define a term.

k) When you get stuck with a term definition, follow either one of the following:

   i) Multiple meanings? Then concentrate on meaning one by one.
   ii) Multiple Viewpoints? Make the viewpoint explicit and then try it again
   iii) Check if you are discussing terminology.
   iv) Use *is-a* hierarchy to give enough context.

[2] Guidelines at Bottom-layer

(1) Determine an essential property for each concept and instance. It could

be informal. At least, when you come into a trouble, having an essential property of each concept helps you a lot.

(2) Each subclass of a super class is distinguished by the values of exactly one attribute of the super class.

(3) Proper use of *is-a* relation should inherit the "Essential" property of its super classes.

(4) Clear and consistent differentiation between basic concepts (*man, rice, oil, etc.*) and role concepts (*teacher, food, fuel, etc.*).

(5) Avoid the use of multiple inheritance relation as much as possible. When you want to use multiple inheritance relation, then it is usually the time to make misconceptualization of the world. Suspect if you are dealing with a *Role concept*, or either one of them is a *part of* the concept(see part 3 article for details).

(6) Do not worry about the vagueness of the boundary between similar concepts. Most boundaries between concepts are vague. A good example is color. While there cannot be a clear boundary between ANY colors, we all share very clear understanding about color, red, blue, etc.

(7) Pay a lot of attention on the context issue, I mean, try to build a context-independent ontology by making the context explicit One of the typical contexts is the *task* you are involved in.

(8) Class partition is not a *part-of* relation. Ex. Male is not *part-of* human.

(9) Deal with "Representation" issue carefully(see part 3 article for details). A copy of the book of "Hamlet" is not an instance of Hamlet(what Shakespeare wrote). It is an instance of book. There can be infinite number of copies of a book with the same content. Needless to say, what Shakespeare wrote is only one and is independent of if it is written on sheet of papers or on anything else.

(10) If multiple meanings are identified in a term, then create a new term, say, Term1 and Term2 temporarily to make each term has only one meaning, since we are dealing with a concept rather than a term.

(11) When you notice you do not have an appropriate term to represent a concept you identify, do not hesitate to coin a new term. The new term could be temporary which will be fixed at the last stage.

(12) Consult a reliable upper ontology when you find the necessity of a general and high level distinction of categories.

§2   Ontology Representation Languages and Tools

This section discusses ontology representation languages and ontology development tools. Usually, an ontology development methodology has its own support tool which has a function to generate the ontology and instances in a few ontology representation languages.

2.1 Languages

[1] Ontolingua

Ontolingua[31] is originally an interlingua for ontology representation and sharing developed by KSL(Knowledge Systems Lab) at Stanford University. It is designed by adding frame-like representation and translation functionalities to KIF(Knowledge Interchange Format)[18] which is a logic-based interlingua for knowledge representation. It can translate from and to some description logics languages such as Loom, Epikit, etc. Ontolingua itself does not have an inference functionality. It has currently developed into a development environment which provides a set of ontology development functions (browse, create, edit, modify and use ontologies) and a library of modular and reusable ontologies. Although it had been a key language for ontology representation for years since its development, it is not active recently because of the advent of XML family languages described below. The following is an example of Ontolingua code.

```
(define-class Tutoring-objective (?t-obj)
''Attributes are also represented as slots."

  :def (and (individual ?t-obj)
          (value-type ?t-obj Tutoring.policy Policy))
  :axiom-def
    (subclass-partition
       Tutoring-objective
       (setof Transfer-of-knowledge Remedy)))
```

This is an implementation of a class *"Tutoring objective"* in a tutoring task ontology for ITS(Intelligent Tutoring System) whose instance is named *?t-obj*. The quotation in the second line is a comment. *:def* and *:axiom-def* allows users to write a necessary condition and definition(necessary and sufficient condition), respectively. The reserved predicates/functions *individual, value-type, subclass-partition* and *setof* mean an instance is an individual rather than a set, value type of a slot as a semantic constraint, partitioning of a class into subclasses and construction of a set, respectively.

In this example, *:def* part reads "any instance *?t-obj* of class *Tutoring-objective* is an individual and its value of *Tutoring.policy* slot has to be an instance of *Policy* class" and *:axiom-def* part reads "the class *Tutoring-objective* has two subclasses as its class partition *Transfer-of-knowledge* and *Remedy*". *Tutoring.policy* is a slot name of the class *Tutoring-objective* and is considered being defined in this code. It is used as an access function afterward.

[2] RDF(S)

RDF(Resource Description Framework) is a framework for metadata description developed by W3C(WWW Consortium). It employs the triplet model *<object, attribute, value>*, well-known in AI community, in which *object* is called resource representing a web page. A triplet itself can be an *object* and a *value*. *Value* can take a string or resource. *Object* and *value* are considered as a node and *attribute* as a link between nodes. Thus, an RDF model forms a semantic network. RDF has an XML-based syntax(called serialization) which makes it

resembles a common XML-based mark up language. But, RDF is different from such a language in that it is a data representation model rather than a language and that the XML's data model is the nesting structure of information and the frame-like model with slots.

Metadata is data about data. In the context of internet information processing, data is whatever is accessed by URL. Any internet resource contains information which is considered as an instance of a certain class. Using XML, basically you can mark up any piece of the original information in-line in which case an XML tag corresponds to a class whose instance is the thing marked-up by the tag. However, what RDF does is different. It creates a new representation in which it contains meta information which usually does not appear in the original resource, that is, metadata about the original information(data). For example, let us take an article as an original data. At the top, it usually contains a character string, say, "Riichiro Mizoguchi". If I mark it up as

```
<author> Riichiro Mizoguchi </author>
```

in the text, then it becomes explicit that the string denotes the author of this article. On the other hand, in the RDF representation of the metadata of the article might include the date when it was published which might not be described in the article. Assuming the article is put at `http://www.ei.sanken.osaka-u.ac.jp/pub/WI2001-Miz.pdf`, the RDF description would be:

```
<rdf:Description rdf:about=''http://www.ei.sanken.osaka-u.ac.jp/pub/WI2001-Miz.pdf">
        <author>Riichiro Mizoguchi</author>
        <pub-date>2001-10-23</pub-date>
</rdf:Description>
```

Although RDF has been designed for metadata representation model, it can be used as a general-purpose knowledge representation, which might be apparent from the fact that it is a kind of semantic network model.

RDF schema is a language to define tags(vocabulary) RDF uses. The most typical and common metadata such as the *creator*(*author*) of a resource and the *date* of its creation are defined in DC(Dublin Core)[5] in which 15 metadata elements are defined. RDF schema does not have to define them for use in RDF but can borrow those 15 metadata elements with the name space: dc: Name space is the functionality given by XML and is used for designating a local world in which a set of tags are valid to avoid conflict between other tag sets. Although, at a first glance, the correspondence between RDF and RDF schema and that between XML and XML schema look equal or at least similar, it is not true. The major role of XML schema is to constrain the instance to which a tag is attached. On the other hand, the major roles of RDF schema include giving tags with definition and their taxonomy to RDF, though it also specifies constraints of the possible values of the triplet. While XML is usable without XML schema, RDF is useless without RDF schema.

RDF schema has its built-in classes and meta-classes by which users can define any class and relation. *Rdfs:Resource* and its two subclasses: *rdf:sClass* and *rdfs:Property* are the key meta-classes. Every ordinary class defined in RDF
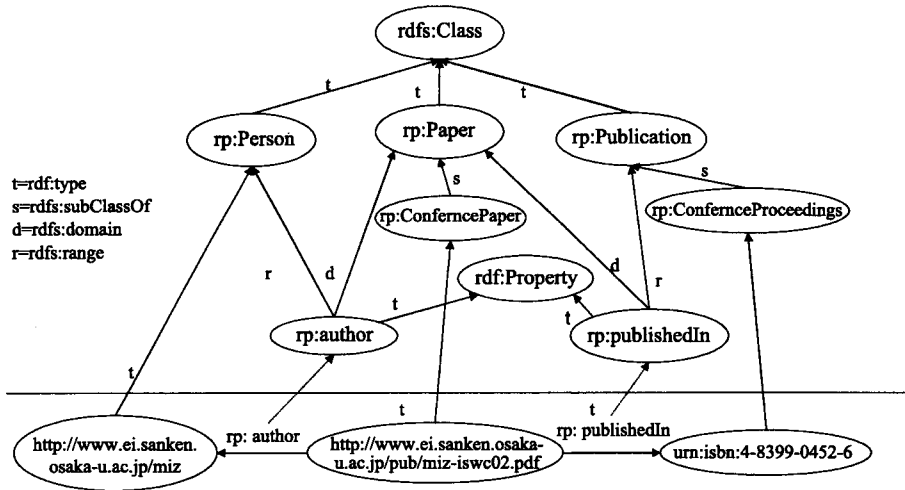
Fig. 2   A Simple Class Hierarchy of RDF Schema

Schema is an instance of *rdfs:Class*. In the same way, every property and relation defined in RDF Schema is an instance of *rdfs:Property*. For example, *rdf:subclass-of* is an instance of *rdfs:Property* and is a built-in relation. This shows that attributes and relations are not distinguished in RDF Schema. Relations and attributes are defined globally, that is, independently of any class unlike frame-based languages in which an attribute is defined as a slot of each class. This comes from Description logics conventions.

Fig. 2 shows a simple class hierarchy of RDF schema including some application-oriented classes related to the example shown in the above RDF code. The following is RDF Schema code of the hierarchy:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdfs:Class rdf:ID="Paper">
   <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdfs:Class>
<rdfs:Class rdf:ID="ConferencePaper">
   <rdfs:subClassOf rdf:resource="#Paper"/>
 </rdfs:Class>
<rdfs:Class rdf:ID="Person"/>
<rdfs:Class rdf:ID="Publication"/>
<rdfs:Class rdf:ID="ConferenceProceedings">
    <rdfs:subClassOf rdf:resource="#Publication"/>
 </rdfs:Class>
<rdf:Property rdf:ID="title">
    <rdfs:domain rdf:resource="#Paper"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
 </rdf:Property>
<rdf:Property rdf:ID="name">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
 </rdf:Property>
<rdf:Property rdf:ID="author">
    <rdfs:domain rdf:resource="#Paper"/>
    <rdfs:range rdf:resource="#Person"/>
```

```
  </rdf:Property>
<rdf:Property rdf:ID="publishedIn">
    <rdfs:domain rdf:resource="#Paper"/>
    <rdfs:range rdf:resource="#Publication"/>
  </rdf:Property>
</rdf:RDF>
```

Assuming the above RDF Schema is put at `http://www.ei.sanken.osaka-u.ac.jp/eg/rp`, an RDF expression using this schema definition in which "rp" is used as a name space is as follows:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:rp="http://www.ei.sanken.osaka-u.ac.jp/eg/rp#">
<rdf:Description rdf:about="urn:issn: 0302-9743">
        <rdf:type rdf:resource=
        "http://www.ei.sanken.osaka-u.ac.jp/eg/rp#ConferenceProceedings"/>
</rdf:Description>
<rp:ConferenceProceedings rdf:about="urn: issn: 0302-9743"/>
<rp:Person rdf:about="http://www.ei.sanken.osaka-u.ac.jp/~miz">
        <rp:name>Riichiro Mizoguchi</rp:name>
</rp:Person>
<rp:ConferencePaper rdf:about=
                   "http://www.ei.sanken.osaka-u.ac.jp/pub/WI2001-Miz.pdf">
        <rp:title> Ontological Engineering: Foundation of the next generation
                   knowledge processing </rp:title>
        <rp:author rdf:resource="http://www.ei.sanken.osaka-u.ac.jp/~miz"/>
        <rp:publishedIn rdf:resource="urn: issn: 0302-9743"/>
</rp:ConferencePaper>
</rdf:RDF>
```

[3] OWL(DAML+OIL)

Web Ontology Language(OWL) is also a language developed by W3C. OWL is designed to make it a common language for ontology representation and is based on DAML+OIL.[3] OWL is an extension of RDF Schema and also employs the triple model. Its design principle includes developing a standard language for ontology representation to enable semantic web, and hence extensibility, modifiability and interoperability are given the highest priority. At the same time, it tries to achieve a good trade-off between scalability and expressive power.

Functionality related to the constraints for instances of a class include: *unionOf* for a Boolean operation of instance sets, *disjointWith* for mutual exclusiveness of classes, *oneOf* for enumeration of all instances, etc. Other functionality for constraints for property value include *rdfs:domain* and *rdfs:range* for restricting domain and range of a relations/property, *minCardinality* for constraining the number of values, *transitiveProperty*, *inverseOf* and so on. In most of the places where a class name is written, a class expression in terms of the Boolean operations of classes can be written to augment class specialization capability. Functionality for interoperability in the distributed environment of semantic web include *sameClassAs*, *differentFrom*, etc. to make it easier to export/import classes. In spite of its rich functionality, OWL is less powerful than the first order predicate logic in logical expression used in axiom writing, since

such functionality is to be covered by the rule layer which is the next higher layer than the ontology layer in the layered cake.[1]

In the same example above, for example, if one wants to add some constraints, he/she has to use OWL. The following OWL code shows a constraint stating ConferencePaper and JournalPapers are mutually exclusive.

```
<owl:Class rdf:ID="JournalPaper">
   <rdfs:subClassOf rdf:resource="#Paper"/>
   <owl:disjointWith rdf:resource="#ConferencePaper"/>
 </owl:Class>
```

A property defined globally can be specialized for a specific class shown in the following where a relation "*author*" is restricted to have more than or equal to two authors when it is applied to CoAuthoredPaper class.

```
<owl:Class rdf:ID="#CoAuthoredPaper">
   <rdfs:subClassOf rdf:resource="#Paper"/>
   <rdfs:subClassOf>
       <owl:Restriction>
         <owl:onProperty rdf:resource="#author"/>
         <owl:minCardinarity>2</owl:minCardinarity>
       </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

[4] Summary

Summarizing the above languages from knowledge representation(KR) point of view, they are within the paradigm which KR community has developed thus far. The new aspect is that they employ XML syntax to cope with web information processing. RDF(S) is a kind of semantic network. OWL is the same as RDF(S) in its data model and in the top-level ontology. The class *rdfs:Property* is a symbol level concept rather than an ontological concept. Therefore, RDF(S) does not distinguish between relations, attributes and features in spite of that all the three are essentially different. OWL does not provide users with adequate modeling facility for representing an ontology, though it is very appropriate for ontology interchange and sharing. In fact, an ontology is something scaffolding conventional knowledge representation onto the real world, that is, the fundamental structure of the world of interest, which require a sophisticated ontology theory. Ontology representation languages are expected to reflect the results of such ontology theories.

2.2 Tools

Incorporating the methodologies and languages, there have been developed many environments for ontology development. Among them, this section takes up OntoEdit,[39] WebODE,[2] Protégé[28] and Hozo[21,36] which cover a wide range of ontology development process rather than being a single-purpose tool which should be covered elsewhere.

[1] OntoEdit

OntoEdit,[39] professional version, is an ontology engineering environment

to support the development and maintenance of an ontology. Ontology development process in OntoEdit is based on their own methodology, On-To-Knowledge discussed in 1.4[37] which is originally based on Common KADS.[34] Two tools, OntoKick and Mind2Onto, are prepared for supporting the 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 discussed in 1.2 which the resulting ontology and ontology-based applications have to answer. 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.

The refinement phase is for developers to use an editor to refine the ontological structure and the definition of concepts and relations. Like most of other tools, OntoEdit employs the client/server architecture where ontologies are managed in a server and multiple clients access and modify one. A sophisticated transaction control is introduced to enable concurrent development of an ontology in a collaborative manner. It employs Ontoclean method[13] mentioned in 2.2[2] and discussed in Part 3 to refine the is-a hierarchy.

The key process in the evaluation 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. 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.

OntoEdit employs F-Logic[19] 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.

[2] WebODE

WebODE[2] is a scalable and integrated workbench for ontology engineering based on the ontology development methodology METHONTOLOGY described in 1.3. 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. 3 and covers most of the processes appearing in the ontology lifecycle. While Protégé-2000 and OntoEdit are based on plug-in architecture, WebODE is based on a client-server architecture which
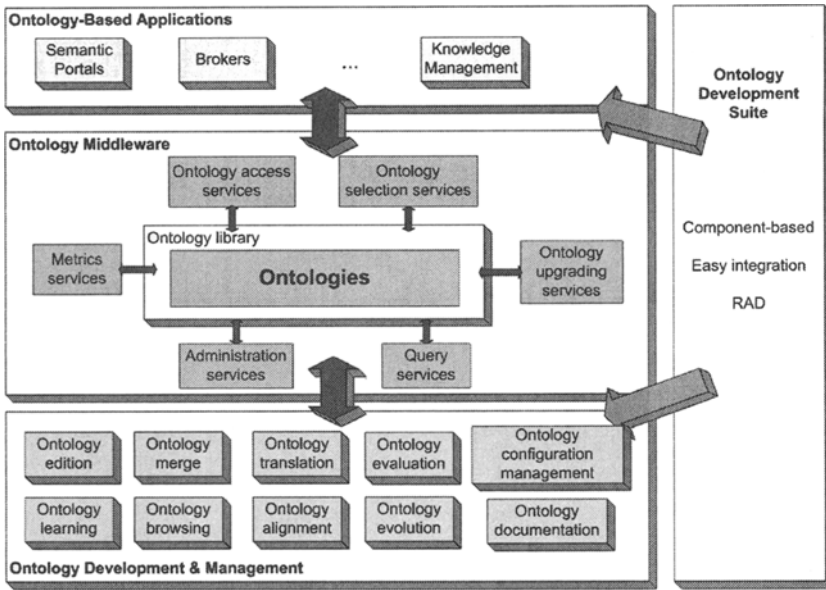
Fig. 3 Architecture of WebODE[2]

provides high extensibility and usability by allowing the addition of new services and the use of existing services. Ontologies are 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. Although WebODE is an integrated tool sets covering most of the activities in ontology lifecycle, it has no explicit stepwise guidance function unlike Hozo.

In the ontology development phase, WebODE has ontology editing service, WAB: WebODE Axiom builder service, inference engine service, interoperability service and ontology documentation service. 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[http://www.ai.sri.com/~okbc/] 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.

Like OntoEdit, WebODE has Ontoclean methodology[13] to build a theoretically correct *is-a* hierarchy. The tool is called ODEClean. Ontology for Ontoclean is composed of the top level universal ontology developed by Guar-

ino, 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 ODE-Clean 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. To support the use process of ontology, WebODE has several functions. Like Hozo, WebODE 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.

[3] Protégé-2000

Protégé-2000[28] 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. So-called "meta-tuning" can be easily done 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[29] discussed in 2.3[2]. It performs some tasks automatically and guides the user in performing other tasks.

[4] OE: Ontology editor in Hozo

"Hozo[*1]" is an integrated ontology engineering environment for building/using task ontology and domain ontology based on fundamental ontological theories.[21,36] "Hozo" is composed of "Ontology Editor," "Onto-Studio" and "Ontology Server." 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, food, fuel, etc. with other basic concepts such as human, vegetable, 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)

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. In each step Onto-Studio, which supports AFM method described in 1.5, 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.

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 treats "role concept" and "relation" on the basis of fundamental consideration discussed in Reference.[21] This interface consists of the following four parts(Fig. 4):

1. *Is-a* **hierarchy browser** displays the ontology in a hierarchical structure according to only *is-a* relation between concepts.

[*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.
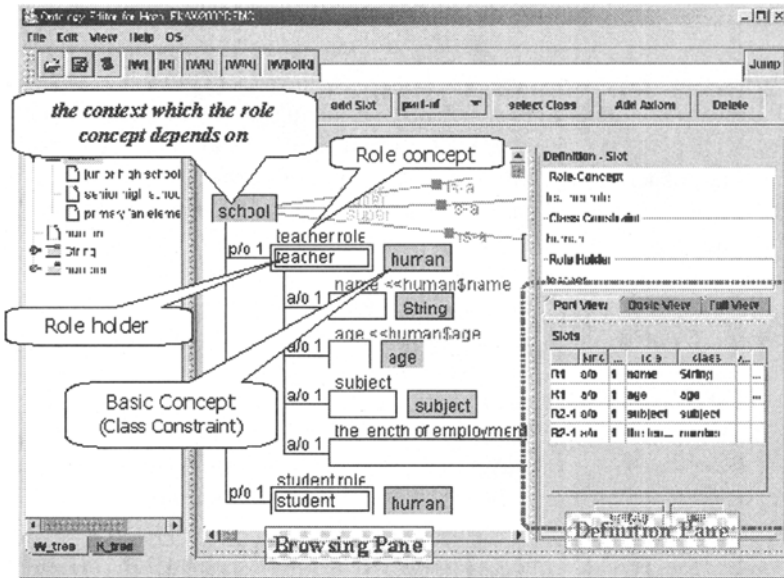
Fig. 4 GUI of Ontology Editor in Hozo

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 and WebODE, Hozo does not allow multiple users to edit the same concept at the same time. Instead, Ontology Editor allows users to divide an ontology into several component ontologies and manages the dependency between them to enable the concurrent development of an ontology. The dependency between the component ontologies is 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 ontology 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.

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 the help of domain experts.[24] 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 built by Hozo 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.

2.3   Ontology Alignment and Merging

An ontology is reusable and sharable in its nature. When building an ontology, if necessary, portion of another ontology should be incorporated in it and in the semantic web context, it is necessary to make the metadata interoperable, which requires merging or alignment of several ontologies. However, the job is not easy to do because of the freedom of naming scheme which prohibit automatic processing of ontology merging/alignment. Here presented are two systems for supporting ontology merge.

[1]   ONIONS[11]

ONIONS(ONtological Integration Of Naïve Sources) is a methodology for merging ontologies and is composed of two major steps: (a) Re-engineering of ontology building data, and (b) merging the ontologies. The first step is further divided into steps such as *extraction, formatting, analysis and formalization* of relevant data. The merging process is based on a *core ontology* which has to be built if it is not available. The ontologies are mapped onto the core ontology to be eventually merged into one ontology taking care of synonymy, polysemy, and taxonomy.

[2]   PROMPT[29]

Although PROMPT is a plug-in tool of Protégé-2000, its procedure is generic enough to be used across various platforms. Fig. 5 shows a flow of the alignment process. Suggestions are made according to the following procedures:

1. Searching for similar names of concepts and their slots.
2. Watching class hierarchy(if a user merge two classes whose super classes are similar, then suggestion of merge of the super classes is made)
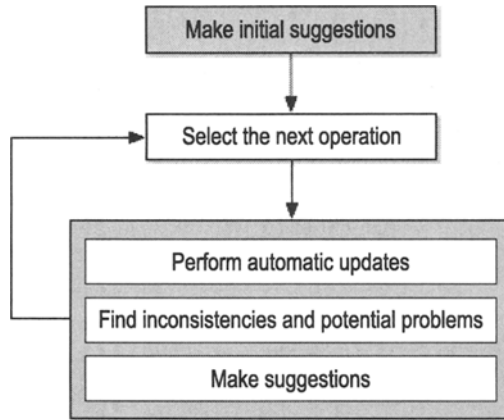
Fig. 5 Control flow of PROMPT[29]

3. Watching slots and their values.

PROMPT can also detect conflicts by observing name conflicts(same name assigned to multiple frames), dangling references(a frame refers to another frame that does not exist), slot-value restrictions violating class inheritance, etc.

§3   Ontologies Developed

Considerable amount of ontologies are already built and used. Here presented are some of them.

3.1   CYC http://www.cyc.com/

Cyc project began in 1984 and it now has more than 100 K atomic concepts axiomatized by a set of more than 1M handcrafted assertions described in nth-order predicate calculus using more than 10K predicates. The knowledge base is the largest in the world and partially covers commonsense knowledge. OpenCyc[33] is the open source version of the Cyc. Cycorp®, the builders of Cyc, has set up an independent organization to disseminate and administer OpenCyc which has about 6,000 concepts with 60,000 assertions and is considered as an upper ontology. Fig. 6 shows a diagram of top-level ontology OpenCyc.

Some characteristics of OpenCyc includes(Cyc terms are headed by '#$'):

1. #$is-a relation in OpenCyc is meant *instance-of* and #$genls relation is used for normal *is-a*.
2. A class is treated as a collection of its instances, so one must be careful not to confuse a collection(a class) and a mathematical set.
3. #$element-of relation, which is a more general relation than #$is-a, is introduced to distinguish *member-of*(#$element-of) and *instance-of*(#$is-a) relations.
4. Multiple inheritance is extensively used. So, users have to pay a closest attention to instance management, especially in the case of instance gen-
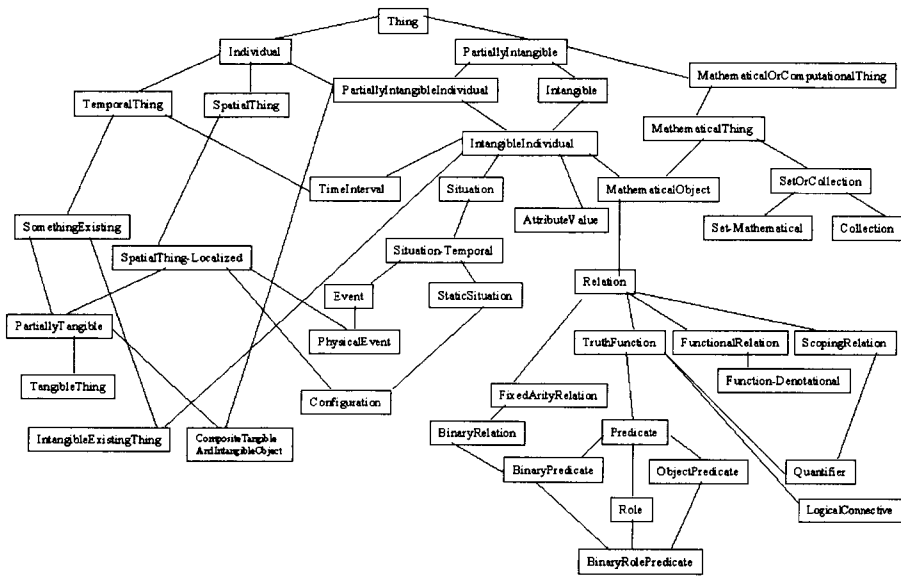
Fig. 6   Upper Ontology of OpenCyc

[http://www.cyc.com/cycdoc/vocab/upperont-diagram.html]

eration and extinction, since identity of the instance cannot be managed properly.

5. #$is-a (*instance-of*) relation is formed besides between an individual and a class.

   For example, my today's lunch event is an instance of #$Event which is a specialization of #$TemporalThing which is an instance of #$Temporal-StuffType which is an instance of #$SecondOrderCollection. A good hypertext documentation of Opencyc is found at: http://www.cyc.com/cycdoc/vocab/vocab-toc.html

6. Due to the problem of inconsistency within a huge knowledge base, microtheories are introduced, so that each microtheory containing a bunch of assertions is consistent by sharing common assumptions about the world.

7. Cyc(OpenCyc) is a useful knowledge base for natural language understanding, since it is built under the goal to capture commonsense people possess.

3.2   Wordnet

While WordNet®, developed by the Cognitive Science Lab. at Princeton University, is an online lexical reference system, its upper level structure is considered as a top-level ontology. Although it is useful as a lexical resource, it has room to improve from ontological point of view. It might be apparent because it has been developed to reflect natural language phenomena, that is, laymen's understanding way of the world. As Guarino points out, there are quite a few in-

appropriate organization of concepts. Typical examples include (a) confusion of concepts and individuals, (b) confusion between object-level and meta-level, (c) incorrect use of *is-a* relation. See Reference[14] for details. Guarino and his group are doing reorganization of the top-level ontology of WordNet. WordNet version 2 is available at `http://www.cogsci.princeton.edu/~wn/wn2.0.shtml`.

3.3 Enterprise Ontology `http://www.aiai.ed.ac.uk/project/enterprise/enterprise/ontology.html`

EO(Enterprise Ontology) has been developed by Mike Uschold and his group at AIAI, Edinburgh University in 1996.[8] It is a pioneering achievement and has a sophisticated methodology for developing a real scale ontology described in 1.1. The purpose of EO includes:

1. To guarantee smooth communication between participants for facilitating sharing the unified understanding about the enterprise model by providing necessary and sufficient vocabulary
2. To provide an infrastructure that is stable but at the same time adaptable to the change of understanding about and requirements to the enterprise model
3. To augment interoperability of various application programs of an enterprise model by using EO as an interlingua for information exchange

Table 1 shows some typical concepts contained in EO.

EO has been evaluated by the project members as well as by those outside the project. In the internal evaluation, they tried to incorporate EO into tools set for enterprise modeling already developed by the whole project. In the attempt, two new ontologies called competency ontology and knowledge space ontology were developed to make up for the abstract characteristic of EO. EO was used as set of vocabulary. In the external evaluation, EO was applied to the three tasks such as bid analysis, market analysis and continuous process improvement. Unfortunately, however, the result was no good. It is mainly because: (1) People's skill in the use of ontology was poor, (2) No computer support such as an ontology browser was available so that it is not easy to understand EO as a whole. (3) Many domain-specific terms are missing. Although the evaluation was not very satisfactory, it was confirmed EO worked as a common vocabulary to lead participants to a common understanding about enterprise model. Good features of EO as an ontology include (a) it introduces *activity* and *activity specification* to model activities in the real world and the planning world. And (b) it introduces the concept *role* explicitly. However, it caused another difficulty to understand EO, since Ontlingua is not so powerful enough to deal with the concept of role that users cannot follow the definition easily.

Table 1 Some Concepts in EO

[http://www.aiai.ed.ac.uk/project/enterprise/enterprise/
ontology.html]

activity	organization	strategy	marketing	time
Activity	Person	Purpose	Sale	Time line
Activity specification	Machine	Hold purpose	Potential sale	Time interval
Execute	Corporation	Intended purpose	For sale	Time point
Executed activity specification	Partnership	Purpose-holder	Sale offer	
T-Begin	Partner	Strategic purpose	Vendor	
T-End	Legal entitiy	Objective	Actual customer	
Pre-Condition	Organizational unit	Vision	Potential customer	
Effect	Management	Mission	Customer	
Doer	Delegate	Goal	Reseller	
Sub-Activity	Management link	Help achieve	Product	
Authority	Legal ownership	Strategy	Asking price	
Activity owner	Non-Legal Ownership	Strategic planning	Sale price	
Event	Ownership	Strategic action	Market	
......	......	......	......	
Resource substitution			Promotion	
			Competitor	

3.4 Gene Ontology http://www.geneontology.org/doc/index.expanded.shtml

Biology is one of the most active research communities in developing and using ontology. In each of the various topics such as genomic, cellular, structure, phenotype and so on, tremendous amount of data are being produced everyday. The problem is there are syntactic and semantic differences in expressing such information, which prevents researchers from retrieving and utilizing the relevant information to facilitate their daily research activity. That is, they are suffering from so-called interoperability of the vast amount of information. What they need is an ontology which provides a common vocabulary. In genomics, the need of such common vocabulary is critical to further acceleration of the understanding of gene functions. This is why Gene ontology consortium has been established under the goal: *"to produce a controlled vocabulary that can be applied to all organisms even as knowledge of gene and protein roles in cells is accumulating and changing" (Exerpt from Reference[10])*.

The three major components of GO are *molecular function, biological process* and *cellular component*. GO contains as of July, 2003, about 1300 component, 5400 function and 7300 process terms. Terms in GO are mainly used for annotation of the existing databases to make them interoperable. GO is available online http://www.geneontology.org/doc/GO.doc.html. Fig. 7 shows how Go looks like. The links are not equal to *is-a* link. This is why every
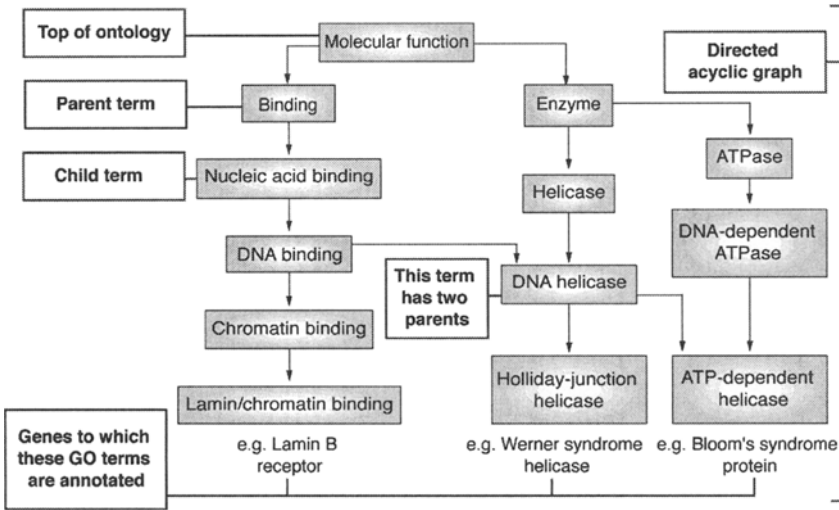
Fig. 7 Directed Graph Representation of GO
[http://www.geneontology.org/doc/GO.doc.html#ontologies]
Image courtesy of the Europian Bioinformatics Institute (www.ebi.ac.uk) and
the Gene Ontology Consortium (www.geneontology.org)

GO term must follow the rule; if a child term describes a gene product, then all its parent terms must also apply to the gene product. It is very important to note that GO is not a dictated standard. GO needs to be adaptive to the rapidly changing findings about gene by a democratic way.

Rigorously speaking, however, GO is a well-defined dictionary rather than an ontology. It needs some improvement from the ontological theory point of view.[35] Cell signaling network ontology is developed at Tokyo University[40][http://www.ontology.jp/].

3.5 Process Ontology: PSL http://ats.nist.gov/psl/psl2.html

Process engineering community is not an exception. It has also serious terminological problems which prevent information exchange and interoperating with application systems such as scheduling systems, production simulation systems. For example, while the term "resource" is used to mean "information source necessary for decision making" in workflow systems, it is used to mean "personnel or machine" in production planning systems. PSL(Process Specification Language) is developed by NIST(National Institute of Standards and Technology, USA) to resolve such difficulties.

The semantics of all terminology within PSL is formally specified in axioms in KIF: Knowledge Interchange Format.[18] PSL ontology does not have an *is-a* hierarchy among concepts. It is more like a common vocabulary with rigorous definition in logic. PSL ontology is generic enough to cover various scheduling, planning and other process-oriented activities and is composed of three major components: PSL core, foundational theories and PSL extensions.

PSL core is composed of four concepts such as *activity*, *activity occurrence*, *time point* and *object* and three relations such as *participates-in*, *before*, and *occurrence-of*. Its unique features are (a) Completely declarative definition of all the concepts in KIF and (b) Differentiation between the two concepts: *activity* and *activity occurrence*. The former is for giving adequate specification of the semantics of the process terminology to avoid inconsistent interpretations and uses of information among application programs. The latter is for clear discrimination between the concept of activity itself and its occurrence. So *activity* is free from time which is for *activity occurrence*. Representation of subactivity relations is done associated with *activity* rather than *activity occurrence*. Taking lunch is an *activity* and Taking lunch at 12:30 pm on Aug. 11 in 2003 is an *activity occurrence*. While this conceptualization might look somewhat odd, it is consistent with what I explained in the concluding remarks of Part 1. An action has two ways of conceptualization: One is that focusing on an *event* which has happened in the real-world. The other is that focusing on the intrinsic property.

Some examples of definition of a concept are shown below.

Definition 1.   Timepoint q is betweenEq timepoints p and r if and only if p is before or equal to q, and q is before or equal to r.

```
(defrelation betweenEq (?p ?q ?r) :=
    (and (beforeEq ?p ?q)
         (beforeEq ?q ?r)))
```

Definition 2.   An activity occurrence is-occurring-at a timepoint p if and only if p is betweenEq the activity occurrence's begin and end points.

```
(defrelation is-occurring-at (?occ ?p) :=
    (and (activity-occurrence ?occ)
         (betweenEq (beginof ?occ) ?p (endof ?occ))))
```

Axiom 1.   The occurrence-of relation only holds between activities and activity-occurrences.

```
(forall (?a ?occ)
    (=> (occurrence-of ?occ ?a)
        (and (activity ?a)
             (activity-occurrence ?occ))))
```

Axiom 2.   An activity-occurrence is the occurrence-of a single activity.

```
(forall (?occ ?a1 ?a2)
    (=> (and (occurrence-of ?occ ?a1)
             (occurrence-of ?occ ?a2))
        (= ?a1 ?a2)))
```

Axioms and definitions are organized as shown in Fig. 8. Formalization in PSL is thorough. It even formalizes what an integer is and has rich axioms about time. On the other hand, concepts processed by activities are left untouched and domain-specific activities such as "painting" are out of focus. Thus, PSL is very much interested in domain-independent inference. It contrasts very well with EO which is mainly concerned with concept extraction from the enterprise domain.

PSL-Core
Outer Core
  Subactivity Theory
  Theory of Occurrence Trees
  Theory of Discrete States
  Theory of Atomic Activities
  Theory of Complex Activities
  Activity Occurrence
Duration and Ordering Theories
  Duration Theory
  Subactivity Occurrence Ordering
  Iterated Activities
  Occurrence Tree Automorphisms
  Envelopes and Umbrae
Resource Theories
  Resource Requirements Theory
  Resource Sets
Actor and Agent Theories
  Activity Performance
Series: Definitional Extensions of PSL
Activity Extensions
  Deterministic Activities: Permuting Branch Structure
  Nondeterministic Activities: Folding Branch Structure
  Nondeterministic Activities: Branch Structure and Ordering
  Nondeterministic Activities: Repetitive Branch Structure
  Spectrum of Activities: Permuting Activity Trees
  .....

Fig. 8 Axioms and Definitions in PSL

3.6 Standard Upper Ontology(SUO) `http://suo.ieee.org/`

IEEE Standard Upper Ontology has begun in May, 2001 after one year preliminary discussion to design a large, general-purpose formal ontology. As mentioned in Part 1, hot topics there include 3D(3D space with time) modeling vs. 4D(including time as the 4th dimension) modeling, multiple ontologies vs. monolithic ontology, etc. The former is concerned with **endurantism**(which claims clear boundary between object(continuant) and process(occurrent)) vs. **perdurantism**(which claims no boundary between the two) argument which seems never-ending. The latter is also very hot, since while, in theory, ontology seems to be universal, it is practically impossible. The current agreement of the SUO is to have a few number of candidate ontologies together with a meta-ontology based on category theory [16] providing a mechanism for managing, integrating and interoperating with multiple ontologies. As of September, 2003, in addition to the two approved candidates: SUMO(Suggested Upper Merged Ontology) proposed by Teknolwedge and OpenCyc,[33] DOLCE[6] ontology designed by N. Guarino and his group and ISO/FDIS 15926-2 (`http://www.tc184-sc4.org/wg3ndocs/wg3n1328/lifecycle_integration_schema.html`) as a 4D ontology (See 4.2[3]) are being proposed as other candidates. There would be a long way to go before we come to an agreement.

As of June 2002, SUMO contains 965 terms and 3742 assertions. The ontology can be browsed online (http://ontology.teknowledge.com), and source files for all of the versions of the ontology can be freely downloaded (http://ontology.teknowledge.com/cgi-bin/cvsweb.cgi/SUO/).

3.7   Other Activities

[1]   WonderWeb http://wonderweb.semanticweb.org/

WonderWeb is a comprehensive project on ontological engineering in Europe. Its main purpose is to establish "Ontology infrastructure for the semantic web". Twenty nine deliverables are prepared under the following 6 work packages. WP.1 (Ontology) Language Architecture; WP.2 (Ontology building) Tools and Services; WP.3 Foundational Ontologies; WP.4 Ontology Engineering; WP.5 Assessment, Dissemination and Evaluation (of the project outcome); WP.6 Project Management. DOLCE is the result of WP3.

[2]   DAML+OIL ontology library http://www.daml.org/ontologies/

DAML project has an ontology base where, as of August in 2003, 251 ontologies are stored in DAML+OIL or OWL. Although some of them look toy, some look very serious. The biggest in number of classes of ontology there is Cancer ontology mentioned below. It also contains DAML version of OpenCyc.

[3]   Cancer ontology[12)]

NCI: National Cancer Institute, USA, has developed a huge ontology intended for NCI offices and divisions to use the Thesaurus as a source of codes associated with concepts to annotate data and other information sources and facilitate information reuse. As of February in 2003 the NCI Thesaurus contains about 26,000 concepts and about 71,000 terms divided into 24 taxonomies which cover administrative, applied and basic science and clinical terminology. Its home page is at http://www.mindswap.org/2003/CancerOntology/.

§4   Applications

4.1   Typology of Ontology Applications

Considering the roles and characteristics of an ontology discussed in Part 1, we can classify the applications of ontology as Jasper and Ushold have done.[17)] In this section, after classifying the ontology applications, we describe some of the typical applications.

Type 1: Ontology as a common vocabulary   This is the most straightforward application type of ontology. As discussed in this article several times, a few ontologies are currently developed in domains for this purpose. Gene and Cancer ontologies are typical examples of this type. Although having a common vocabulary is a first step towards knowledge systematization of the domain, there are a lot to do before realizing it.

Type 2: Ontology as the help of information access   WWW is a huge informa-
tion source which would be able to give us enormous value. This is why people
are so enthusiastic about making information access more and more intelligent.
Metadata, Ontology-based information search, Knowledge management(KM)
are those efforts. Ontology gives a foundation of those research activities in
two ways: One is to provide metadata elements and vocabulary to put anno-
tations on the WWW resources and the other is to use class hierarchy and
relations among classes for interpreting the metadata attached to each resource.
KM utilizes both.

Type 3: Ontology as the medium for mutual understanding   Mutual understand-
ing is always necessary between (a) humans and humans, (b) humans and soft-
ware agents, (c) software agents and software agents. Even the communication
between humans, ontology can be useful especially for knowledge-intensive en-
gineering such as concurrent engineering, business process reengineering, etc.
where interdisciplinary collaboration is required. Understanding between hu-
mans and software agents is seen in the case of WWW resource search. The
requirements specified by the users have to be properly understood by search
engines through the shared ontology or ontology translation. Semantic web[1] is
the biggest application of this type. The same happens in the case of com-
munication between software agents. FIPA takes it up and produces FIPA
ontology service specification(http://www.fipa.org/specs/fipa00086/
XC00086D.html#_ftn15). In order to enable software agents to communicate
each other, they need a common protocol and vocabulary. An ontology plays
a role of a common vocabulary in a more advanced way than that in type 1
because such vocabularies are managed by ontology agents in a formal way to
process queries about ontologies by other agents.

Type 4: Ontology as specification   An instance is a model of the real-world thing
and an ontology is a model on instances, so an ontolology is a meta-model
which specifies what instance model is possible. Applications which utilize the
model specification functionality of an ontology include an authoring support
system which has to know what it is going to author. I call such an authoring
tool which has declarative specification about what it is going to produce as
an ontology "an ontology-aware" authoring tool. An ontology-aware authoring
tool can exploit the utility of an ontology. Protégé-2000 which is known as
an ontology development tool has been a knowledge acquisition meta-tool in
that it uses class definitions as a specification of instances which are target of
acquisition guided by the specification. What users do is to design a set of
domain-specific class definitions. Then Protégé-2000 automatically becomes a
knowledge(instance model) acquisition system of the domain.

Type 5: Ontology as foundation of knowledge systematization   An ontology pro-
vides us with a kernel conceptual structure. This functionality is especially
significant for the upper ontology. Knowledge systematization requires:

1. Formalization of the basic terms
2. Positioning them in appropriate places in the conceptual structure by identifying relationship to others
3. Further identification of necessary relationships between concepts
4. Deeper understanding of concepts considering their use situations and compile them in a use-ready form

These are what an ontology can provide. In the task of data exchange in engineering domains, for example, there have been a long effort of standardization of the date exchange format.

EPISTLE,[7] which is a successor of the STEP: the Standard for the Exchange of Product Model Data activity at ISO, is a good example of ontology use for knowledge systematization. The problem of STEP was multiple conceptualizations of product data for respective domains, which prevented engineers from having a unified model of product data. To realize it, they did need a common conceptual structure which is the basic motivation of the EPISTLE project.

In the production/manufacturing domain and computer-assisted design domain, sharing engineers' expertise among engineers has been a hot topic for facilitating their knowledge-intensive activities. Functional knowledge systematization based on functional ontology[20,26] is a long-term activity aiming at the goal. It has been successfully deployed in the industries and will be discussed in detail in Part 3. Similar activities are found in Nanotechnology[27] and Instructional design communities.[25]

4.2   Some Applications

[1]   Semantic web `http://www.w3.org/2001/sw/`
The semantic web is an effort to make the current WWW computer-understandable. Computers currently process WWW resources only as a sequence of bits or characters, which causes low performance in finding relevant WWW pages and hence causes information overflow. Semantic web is different. A search engine would understand the content of each piece of WWW information and hence it could find much more relevant information with much less irrelevant information. Furthermore, one would be able to produce many kinds of *intelligent* applications on top of semantic web in the areas of e-Commerce, e-Learning, etc. to utilize the WWW maximally.

In the future, semantic web technology would bring us a kind of revolution to a knowledge base building. Conventionally, a knowledge base has been something to design and build upon request. However, WWW and semantic web technologies facilitate automatic building of knowledge resources so that a huge knowledge base virtually exists out there, and hence the problem to solve would become not to build a knowledge base from scratch but to collect appropriate web pages out of already existing WWW knowledge resources, to reorganize and to merge them. To make this happen, we need to solve a lot of problems by providing key technology.

(a) Meta-data and semantic interoperability(type 3)   The key technology to make the semantic web happen is the XML-based markup language family with ontology-based semantic processing. So-called "Layered cake[1]" is an architecture for it in which URI->XML->RDF(S)->OWL->Logic->Proof->Trust form layered structure of technology. URI provides a unique variable, XML a markup syntax and name space, RDF(S) a metadata model, OWL ontology, Logic logical foundation, Proof record of proof and Trust establishes the trust relationships. Technologically, the key issues are semantic interoperability among the metadata and trust and practically, a key issue is how to persuade users to produce metadata for their resources. The WWW has grown in a bottom up manner with the help of HTML technology. Some say the semantic web will grow as the same way as the WWW, but others say the semantic web needs different strategies to grow. It is true that writing metadata is harder than writing an HTML page and the merit of the former is less appealing than the latter.

Technologically, ontology plays the key role in making metadata interoperable. The meaning of the vocabulary in a metadata is defined in the ontology. It is OK in a single metadata case. The problem is how to make inference in a case of multiple metadata defined in multiple ontologies. Ontology mapping, alignment or merging discussed in 2.3 is critical. The topic is challenging. Although OWL has functions to support and some research has been done on this topic, there remains a lot to do. We have a dilemma on this issue. If we had the universal ontology everyone shares, semantic interoperability were no more a problem. If every metadata producer has his/her own ontology, it is a mess and we never be able to achieve semantic interoperability among them. So, the solution has to be in between the two extreme. However, we do not know where it should be. I at least can say we need a few reliable upper ontologies each of which has clear and explicit ontological choice it is based on and a reasonable number of domain ontologies beautifully designed in a principled way.

(b) Web service ontology[4] http://www.daml.org/services/ **(type 2)**   Web services (http://www.w3.org/2002/ws/) are Web-accessible computer programs which are platform-independent. Their characteristics are described in terms of WSDL: Web Services Description Language[http://www.w3.org/TR/wsdl] and found by UDDI: Universal Description, Discovery and Integration of web services [http://www.uddi.org/] and communicate with each other through SOAP: Simple Object Access Protocol[http://www.w3.org/TR/SOAP/] exchanging information for processing in XML. Web services activity shares the same problem with the WWW, that is, how to find the relevant services. It is straightforward to apply the semantic web technology to solve this problem. DAML-S is an ontology for semantic markup of the web services. The purpose of metadata writing for web services has the following four:

1. Automatic Web service discovery.
2. Automatic web services invocation
3. Automatic web services composition and interoperation
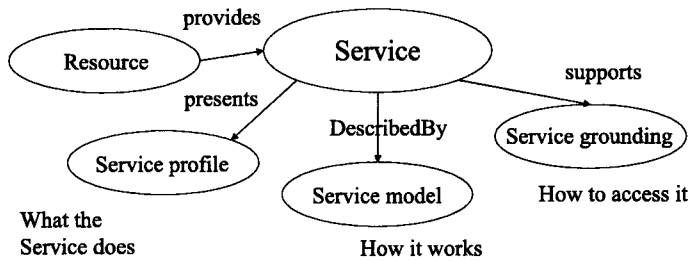4. Automatic web services execution monitoring

Fig. 9   Top-level Ontology of DAML-S[4]

These goals are really ambitious. Especially, 3. is close to automatic programming which is an old and hard topic. Fig. 9 shows top-level ontology of DAML-S. *Service profile* contains concepts and attributes used for describing services to use for finding services. For representing what the service does, it has *input, output, precondition* and *effect. Service model* is responsible for representing how the service works. DAML-S employs the process ontology which requires precondition, input and out parameters, participants in a process and effects. Input, output and effects are connected to those introduced in *Service profile.* Process has three major subclasses such as *AtomicProcess, SimpleProcess* and *CompositeProcess* as shown below.

```
<daml:Class rdf:ID="Process">
  <rdfs:comment> The most general class of processes </rdfs:comment>
  <daml:disjointUnionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#AtomicProcess"/>
    <daml:Class rdf:about="#SimpleProcess"/>
    <daml:Class rdf:about="#CompositeProcess"/>
  </daml:unionOf>
</daml:Class>
```

AtomicProcess is a process which is directly invocable, that is, it has no subprocess, and hence it has to be grounded on WSDL description. *SimpleProcess* is used for representation of *atomic* or *CompositeProcess* in the case of planning or reasoning. It is not grounded on WSDL but an abstract single-step process. *CompositeProcess* is decomposable into subprocesses. DAML-S ontology also contains concepts for control such as *sequence, split, unordered if-then-else* and so on.

DAML-S has the following three assumptions:

1. An *AtomicProcess* corresponds to an operation of WSDL.
2. Input and output of each *AtomicProcess* correspond to WSDL message part.
3. The type of each WSDL message part can be specified as the range of a DAML-S parameter.

These are not serious limitations, in fact most of them are got rid of at the latest version(V0.9) which will be called OWL-S. Although omitted here, there are more important classes such as, *Time, Resource and Capacity*, in DAML-S.

[2] e-Learning `http://ltsc.ieee.org/`

e-Learning has been growing into a big industry. Not only school learning but also training of personnel in a company and life-long learning require sophisticated and easy-to-use learning support systems. One of the demanding issues in e-learning is reusability and interoperability of learning materials. To cope with this issue, standardization of LMS: Learning Management Systems, LOM: Learning Object Metadata, etc. have been done in ADL: Advanced Distributed Learning (`http://www.adlnet.org/index.cfm?fuseaction=scormabt`), ARIADNE(`http://www.ariadne-eu.org/`), IEEE LTSC: Learning Technology Standards Committee (`http://ltsc.ieee.org/`) and ISO SC36.

(a) LOM: Learning Object Metadata `http://ltsc.ieee.org/wg12/index.html` **(type 2)**   By LO: learning objects, we mean any contents used in learning independently of its grain size. LOM is metadata of LO. LOM is partially based on Dublin core because both are metadata for information resources and contains learning-specific elements. A metadata is a pair of attribute and its value. The former is called metadata element and the latter vocabulary. LOM metadata elements consist of nine categories: the general category, the lifecycle category, the meta-metadata category, the technical category, the educational category, the rights category, the relation category, the annotation category and the classification category. LOM ontology is mainly for specifying vocabulary. LOM is carefully designed to attain maximal learning-subject-independence just like CD-ROM standard is independent of its contents. To deploy LOM, however, users need to fill in the value part of LOM with domain-specific vocabulary which might decrease its interoperability. This is why well-designed ontology is necessary for each category. Especially, ontologies for important concepts such as educational goal, competency, learning object types, etc. are being developed.[15]

(b) Ontology-aware authoring system (type 4)   Authoring of learning/teaching courseware is a kind of task which can be characterized by task ontology described in Part 1. Among various performance systems, an authoring system is special in that it is a meta-system because it generates a learning material which partially specifies the behavior of a learning support system together with structured learning topics. Of particular importance here is that there are two kinds of task ontologies: One is authoring task ontology and the other is tutoring task ontology in the case of courseware for an intelligent tutoring system. The latter ontology plays the key role. An authoring system which knows tutoring task ontology can behave intelligently because it knows what it is generating for what purpose.[25]

[3]   Knowledge systematization (type 5)

(a) EPISTLE [7]   In order to come up with a unified view of concepts in the oil and plant engineering domain, EPISTLE has a sophisticated model called ECM: Epistle Core Model. It is based on the 4D viewpoint of the world(see

3.6), that is, it tries to model an entity as a trajectory in the 4D space. It is necessary for them to model artifacts with its lifecycle from its design phase until its installation and use phase. Other unique features of ECM include: (a) introduction of the intended and actual worlds to model things in the design or planning phase and thins in the actual world with a special link *realization-of* (instead of *instance-of*) between the two kinds of things. (b) Two classes called *"Individual"* and *"Class"* introduced under the *"Thing"* class. The former is for representing all kinds of individual existing not only in the actual world but in planning or imaginary worlds. It is a very unique idea. An implementation of *Individual* class is shown bellow.

```
ENTITY individual
    SUPERTYPE OF
    (ONEOF(plural_individual, single_individual) ANDOR ONEOF(state,
    temporal_boundary_of_state) ANDOR actual_individual ANDOR
    whole_individual ANDOR ONEOF(point_in_space, vector_in_space))
  SUBTYPE OF (thing);
END_ENTITY;
```

ECM thus provides a convincing data model supported by a unique ontology and serves as a core model through which existing domain-specific data models are mapped. What EPISTLE has done is not just a simple data mapping, but an effort to systematizing the structure of the engineering product model world.

§5   Concluding Remarks

While the importance of ontology has been already well-received in many domains as well as in computer science, there still exist at least two major interpretations of what an otology is. In the semantic web context, ontology plays the key role in interoperability of metadata. Such an ontology would be something like "a logical theory accounting for the intended meaning of a formal vocabulary" defined by Zuniga.[41] On the other hand, people working on upper ontology believe that an ontology should have philosophical justification to some extent. These two sometimes cause conflicts which will need a long time to resolve.

Many tools are available now and help people develop an ontology. However, it is still true that building a good ontology is not easy. There are two kinds of difficulties: One is how to identify and organize classes and the other is how to define classes in terms of axioms. Although the former should be taken care of by an ontology building methodology, we have few convincing guidelines. Part 3 will discuss an effort to this direction. While the latter is taken care of by the existing tools, rigorous definition of a concept is not an easy task. Users need a lot of knowledge and skills about formal stuff. One of the most critical issues is how much to put in axioms. An ontology is expected to be reusable and sharable, and hence less task-dependent. If an ontology has too much knowledge, it becomes yet another knowledge base. We need a guideline for drawing the boundary between application-dependent knowledge bases and a shared ontology. The idea of task ontology[23] would be partially helpful to do

it because it helps separate task-dependency from the domain ontology.

A few ontologies have been developed and used in non-IT domains where an ontology is mainly used as a common vocabulary. In the Semantic web, well-formalized ontologies are expected to contribute to realizing intelligent behaviors in finding appropriate WWW pages and web services and types 2 and 3 applications are produced. Applications of types 4 and 5 are still not active yet. Knowledge systematization, type 5, is a promising application of ontological engineering which will be discussed in detail in Part 3.

Acknowledgements

References

1) Berners-Lee, T., Hendler, J. and Lassila, O., "The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation," *The Semantic Web, Scientific American*, May 2001.

2) Corcho, O., Fernandez-Lopez, M., Gomez-Perez, A. and Vicente, O., "WebODE: An Integrated Workbench for Ontology Representation, Reasoning and Exchange," in *Proc. of EKAW2002, Springer LNAI 2473*, pp. 138–153, 2002.

3) http://www.daml.org/

4) http://www.daml.org/services/

5) http://dublincore.org/

6) http://www.loa-cnr.it/DOLCE.html

7) http://www.btinternet.com/~chris.angus/epistle/index.html

8) http://www.aiai.ed.ac.uk/project/enterprise/enterprise/ontology.html

9) http://www.fipa.org/specs/fipa00086/XC00086D.html#_ftn15

10) http://www.geneontology.org/doc/index.expanded.shtml

11) Gangemi, A., Pisanelli, D. M. and Steve, G., "An Overview of the ONIONS Project: Applying Ontologies to the Integration of Medical Terminologies," *Data Knowledge Engineering, 31, 2*, pp. 183–220, 1999.

12) Golbeck, J., Fragoso, G., Hartel, F., Hendler, J., Parsia, B. and Oberthaler, J., "The National Cancer Institute's Thesaurus and Ontology," *Journal of Web Semantics, 1, 1*, Oct. 2003.

13) Guarino, N. and Welty, C., "Evaluating ontological decisions with OntoClean," *Communications of the ACM, 2, 45*, pp. 61–65, 2002.

14) Oltramari, A., Gangemi, A., Guarino, N. and Masolo, C., "Restructuring Word-Net's Top-Level: The OntoClean Approach," in *Proc. of LREC2002 (OntoLex workshop)*, Las Palmas, Spain.

15) Hirata, K., Takaoka, Y., Ohta, M. and Ikeda, M., "The Meaning of LOM and LOM Authoring Tool on HRD," in *Proc. of International Conference on Dublin Core*

and Metadata Applications 2001, pp. 259–262, National Institute of Informatics, Tokyo, Japan, Oct. 22–26, 2001.

16) http://suo.ieee.org/IFF/

17) Jasper, R. and Uschold, M., "A Framework for Understanding and Classifying Ontology Applications," in *Proc. of KAW99*, Banff, Canada, Oct. 1999.

18) http://logic.stanford.edu/kif/kif.html

19) Kifer, M., Lausen, G. and Wu, J., "Logical Foundations of Object-oriented and Frame-based Languages," *Journal of the ACM, 42*, pp. 741–843, 1995.

20) Kitamura, Y. and Mizoguchi, R., "Ontology-based Description of Functional Design Knowledge and its Use in a Functional Way Server," *Expert Systems with Application, 24, 2*, pp. 153–166, Feb. 2003.

21) Kozaki, K., Kitamura, Y., Ikeda, M. and Mizoguchi, R., "Hozo: An Environment for Building/Using Ontologies Based on a Fundamental Consideration of "Role" and "Relationship," in *Proc. of the 13th International Conference Knowledge Engineering and Knowledge Management (EKAW2002)*, pp. 213–218, 2002.

22) Fernandez-Lopez, M., Gomez-Perez, A. and Pazos Sierra, J., "Building a Chemical Ontology Using Methontology and the Ontology Design Environment," *IEEE Intelligent Systems, 14, 1*, pp. 37–46, 1999.

23) Mizoguchi, R., Ikeda, M., Seta, K. and Vanwelkenhuysen, J., "Ontology for Modeling the World from Problem Solving Perspectives," in *Proc. of IJCAI-95 Workshop on Basic OntologicalIssues in Knowledge Sharing*, pp. 1–12, 1995.

24) Mizoguchi, R., Kozaki, K., Sano, T. and Kitamura, Y., "Construction and Deployment of a Plant Ontology," in *Proc. of the 12th International Conference Knowledge Engineering and Knowledge Management (EKAW2000)*, pp. 113–128, 2000.

25) Mizoguchi, R. and Bourdeau, J., "Using Ontological Engineering to Overcome AI-ED Problems," *International Journal of Artificial Intelligence in Education, 11, 2*, pp. 107–121, 2000.

26) Mizoguchi, R., "Ontology-based Systematization of Functional Knowledge," in *Proc. of TMCE2002: Tools and methods of competitive engineering*, Wuhan, P.R.China, pp. 45–64, 2002.

27) "Knowledge Systematization Through Ontological Engineering," in *Proc. of the 6th Sanken International Symposium — Data mining, Semantic web and Computational sciences —*, Osaka University, Mar. 2003.

28) Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubezy, M., Eriksson, H., Noy, N. F. and Tu, S. W., "The Evolution of Protégé: An Environment for Knowledge-Based Systems Development," *International Journal of Human-Computer Interaction, 58, 1*, pp. 89–123, 2003.
     http://smi.stanford.edu/pubs/SMI_Abstracts/SMI-2002-0943.html

29) Noy, N. F. and Musen, M. A., "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment," *Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, Austin, TX, 2000.

30) Noy, N. F. and McGuinness, D. L., "Ontology Development 101: A Guide to Creating Your First Ontology," *SMI technical report SMI-2001-0880*, 2001.

31) http://www.ksl.stanford.edu/software/ontolingua/

32) http://ontoweb.aifb.uni-karlsruhe.de/About/Deliverables/D1.4-v1.0.pdf

33) http://www.opencyc.org/

34) Schreiber, G. et al., *Knowledge Engineering and Management: The Common KADS Methodology*, MIT Press, Cambridge, Mass., 1999.

35) http://www.bioinfo.de/isb/2002/02/0017/

36) Sunagawa, E., Kozaki, K., Kitamura, Y. and Mizoguchi, R., "An Environment for Distributed Ontology Development Based on Dependency Management," in *Proc. of ISWC-2003*, Sanibel Island, Florida, USA, 2003.

37) Staab, S. H., Schunurr, P., Studer, R. and Sure, Y., "Knowledge processes and ontologies," *IEEE Intelligent Systems, Special Issue on Knowledge Management, 16, 1*, pp. 26–34, 2001.

38) http://www.steptools.com/library/standard/

39) Sure, Y., Staab, S., Erdmann, M., Angele, J., Studer, R. and Wenke, D., "OntoEdit: Collaborative Ontology Development for the Semantic Web," in *Proc. of ISWC2002*, pp. 221–235, 2002.

40) http://www.ontology.jp/

41) Zuniga, Z., "Ontology: Its transformation from philosophy to information systems," in *Proc. of Formal Ontology in Information Systems* (Welty and Smith eds.), ACM Press, pp. 187–197, 2001.

42) Gruninger, R. and Fox, M., "The design and evaluation of ontologies for enterprise engineering," in *Proc. of Comparison of Implemented Ontologies, ECAI'94 Workshop, W13*, pp. 105–128, 1994.

Riichiro Mizoguchi, Ph.D.: He is Professor of the Department of Knowledge Systems, the Institute of Scientific and Industrial Research, Osaka University. He received his B.S., M.S., and Ph.D. degrees from Osaka University in 1972, 1974 and 1977 respectively. From 1978 to 1986 he was research associate in the Institute of Scientific and Industrial Research, Osaka University. From 1986 to 1989 he was Associate Professor there. His research interests include Non-parametric data analyses, Knowledge-based systems, Ontological engineering and Intelligent learning support systems. He is a member of the Japanese Society for Artificial Intelligence, the Institute of Electronics, Information and Communica-tion Engineers, the Information Processing Society of Japan, the Japanese Society for Information and Systems in Education, Intl. AI in Education (IAIED) Soc., AAAI, IEEE and APC of AACE. Currently, he is President of IAIED Soc. and APC of AACE. He received honorable mention for the Pattern Recognition Society Award, the Institute of Electronics, Information and Communication Engineers Award, 10th Anniversary Paper Award from the Japanese Society for Artificial Intelligence and Best paper Award of ICCE99 in 1985, 1988, 1996 and 1999, respectively. He can be reached at miz@ei.sanken.osaka-u.ac.jp