Distributed Construction of Ontologies Using Hozo

Kouji Kozaki　Eiichi Sunagawa　Yoshinobu Kitamura　Riichiro Mizoguchi

The Institute of Scientific and Industrial Research (ISIR), Osaka University
8-1 Mihogaoka, Ibaraki, Osaka, 567-0047 Japan
+81-6-6879-8416

{kozaki, sunagawa, kita, miz}@ei.sanken.osaka-u.ac.jp

ABSTRACT

This paper discusses Hozo's functionality for supporting distributed and collaborative construction of ontologies. In a distributed environment, each ontology is revised asynchronously by different developers. In such a situation, one of the key issues is the maintenance of consistency among inter-dependent ontologies. In order to realize consistent distributed development of ontologies, Hozo provides two functionalities: to manage the dependencies between ontologies and to keep and restore consistencies of ontologies when they are changed.

Keywords

Ontology, Distributed development, Dependency management

1. INTRODUCTION

Ontology is one of the key technologies to realize Semantic Web. In the Semantic Web, ontologies are shared and reused by different developers in a distributed environment. To construct large scale ontologies, it is necessary to collaborate with many developers. Therefore, distributed and collaborative construction of ontlogies is one of the most significant issues. In this paper, we discuss an ontology development system named Hozo to support a construction of ontologies. We focus on providing a framework for distributed development. Its main features include a dependency management of ontologies and a framework for supporting to keep consistency of ontologies. Section 2 discusses a distributed ontology development we assume. Section 3 summarizes Hozo and flow of distributed ontology development. The functionalities of Hozo to support such a development are described in section 4. In Section 5, discuss some related work. We conclude with a summary of some future works in Section 6, .

2. DISTRIBUTED CONSTRUCTION OF ONTOLOGY

We assume a situation where several ontologies are constructed separately in a distributed environment (and sometimes in parallel by different developers). In such a situation, some ontologies may import concepts (classes) defined in other ontologies, and another concept might be defined in the ontology by extending the imported concepts (see Figure 1). And then, it means the ontology B which imports concepts from Ontology A depends on ontology A. In this paper, we call ontologies which are depended by other ontology and those depend on others *depended ontologies*, and dependent ontologies, respectively. In Figure 1, Ontology A is the depended ontology of Ontology B, and Ontology B is the

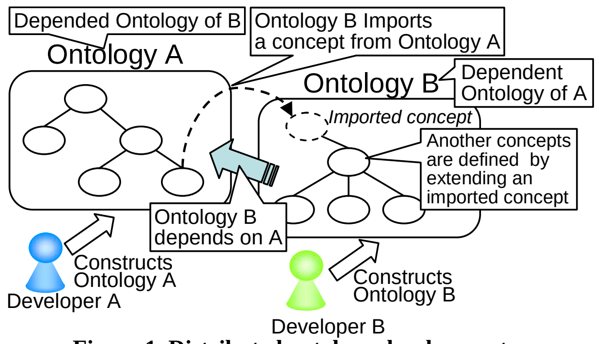


Figure 1. Distributed ontology development.

dependent ontology of A. The authors call a development of ontologies in such a manner *distributed ontology development*. The distributed ontology development applies to many situations: cooperative development, understanding the total picture of conceptual hierarchy of ontologies, reusing published ontologies and so on.

In the distributed ontology development, developers construct multiple ontologies in cooperation among the developers. They can reuse published components of other ontologies if possible. It is a common way for ontology development to extend an existing ontology into a target-specific ontology. However, when developers construct ontologies in parallel or reuse ontology which is under construction and thus unstable, consistency among the ontologies is easily broken because they are revised asynchronously without notice. Furthermore, they are possibly updated without concerning whether other ontologies depend on them or not and how those ontologies would be influenced by their changes because authorities for maintenance of the ontologies are separated and distributed to each developer of them. Therefore, when a developer changes his/her ontology, the change influences on dependent ontologies of it. In many cases, such a change may cause inconsistencies among the ontologies.

For consistent development of ontologies, a system should manage dependencies among the ontologies and support their developers to harmonize the ontologies. Practically, the system should have at least two following functionalities:

1) To manage ontologies with its dependencies on others.
2) To provide a framework to keep and restore consistencies of ontologies when they are changed.

Based on this observation, the authors have investigated how a change of one ontology influences on others through its dependencies. And they have devised strategies for the change in order to keep and restore the consistency of them [1]. They have implemented a framework for ontology development in harmony among depended/dependent ontologies as an extension of Hozo (our ontology development tool) [2].

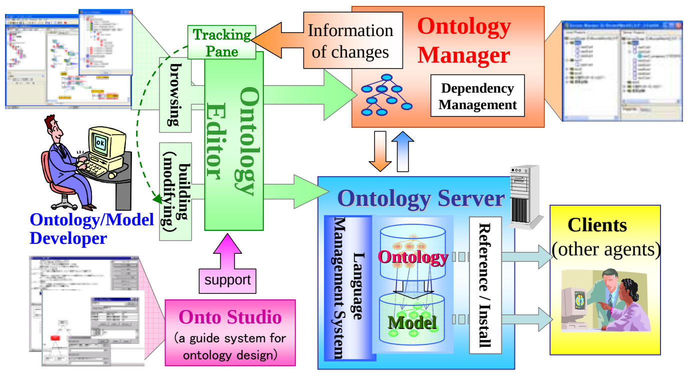


Figure 2. Architecture of Hozo

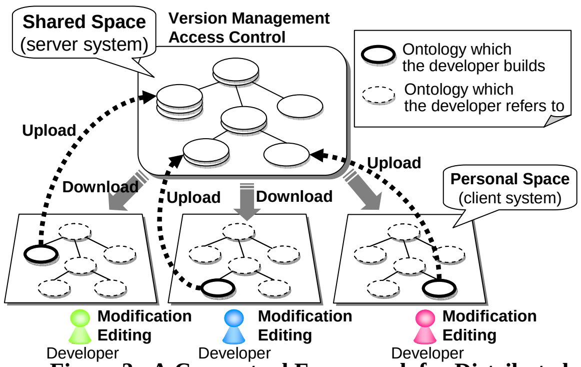


Figure 3. A Conceptual Framework for Distributed Ontology Development.

3. HOZO

3.1 Overview of Hozo

We have developed an environment for building/using ontologies, named Hozo, based on both of a fundamental consideration of an ontological theory and a methodology of building an ontology. The features of Hozo include 1) Supporting role representation [3, 4], 2) Visualization of ontologies in a well considered format, and 3) Distributed development based on management of dependencies between ontologies. Hozo is composed of Ontology Editor, Onto-Studio (a guide system for ontology design), Ontology Server and Ontology Manager (see Figure 2). The ontology editor provides a developer with a graphical interface, through which they can browse and modify an ontology locally. The ontology server stores and manages ontologies under access control and version management. The developer can access and browse them through the ontology manager. Furthermore, the ontology editor of Hozo provides a user support module to maintain consistencies of the dependencies among ontologies, called Tracking Pane. Hozo's native language is XML-based frame language and ontologies can be exported in OWL, and RDF(S). It does not support native OWL, but it can import OWL partially[1]. The latest version of Hozo is published at the URL: http://www.hozo.jp.

3.2 Flow of Distributed Ontology Development

Figure 3 shows a skeleton of our conceptual framework for distributed ontology development. It consists of two parts in a server-client architecture. One is a shared space, where developers store ontologies to be opened to other developers. The other is a local space, where each developer builds and modifies each ontology which he is responsible for. The developers cannot edit the ontologies stored in the shared space directly. Under access control and version management, they edit the personal copies of ontologies locally and upload them to the shared space when necessary.

In the distributed ontology development, a target ontology can be regarded as a system of interrelated ontologies stored in the shared space. They are constructed in cooperation among the developers. Each developer constructs some of them under his responsibility[2]. Then, he may refer to other ontologies and import concepts defined in them. It implies that each developer has two kinds of ontologies: ontologies which the developer builds and ontologies which he/she refers to.

Distributed ontology development proceeds in the repetition of the following steps;

1) A developer gets latest information on ontologies which he builds or refers to from the ontology server. And he downloads them form shared space to personal space (client) through an ontology manager. If it is needed, he locks ontologies to avoid that someone updates those ontologies while he is editing them.

2) The developer analyzes changes in the updated ontologies and evaluate whether the changes are influencing on consistency of the ontology which he is constructing with the help of ontology manager.

3) If the changes cause inconsistency in his ontology, the developer modifies his ontology in order to keep and restore its consistency with the updated ontologies. Hozo helps this modification process by suggesting possible countermeasures for coping with each of the changes.

4) After the modification, the developer starts editing his ontology as he needs. While editing the ontology, he can imports and use concepts from other ontologies which he refers to. Then the dependency between his ontology and the referred ontology through the imported concepts is managed by the ontology manager.

5) After editing, the developer publishes his ontology by uploading it to the shared space. And then, he unlocks the ontology if he allows other developers to edit it.

Every developer goes over this process individually and in parallel, and then the whole target ontology evolves. As a result of their developing processes the whole target ontology is constructed in the shared space.

We suppose another collaborative development process such as constructing a single ontology by many developers. Our distributed ontology development also can support such process in the repetition of the following steps:

1) The developers share a target single ontology in the shared space. The ontology server manages versions of the ontology and accesses to it.

2) When a developer edits the target ontology, he locks the ontology and downloads it to his personal space.

[1] The OWL import mechanism is under improvement.

[2] A same component ontology may be constructed by several developers.

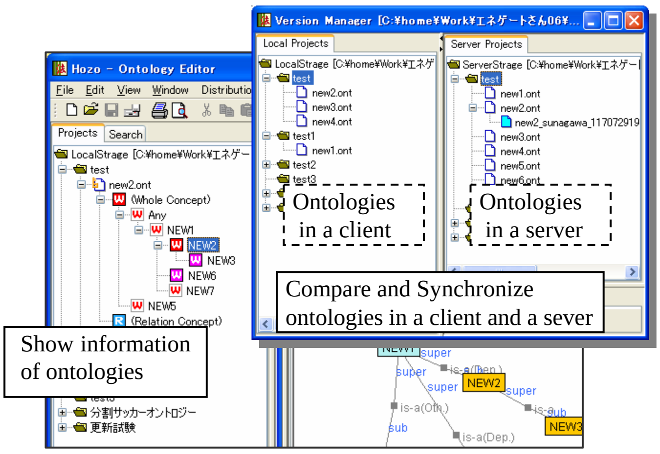

Version Manager [C:¥home¥Work¥エネゲートさん06¥...
Local Projects
Server Projects
Hozo – Ontology Editor
File Edit View Window Distributio
Projects Search
Ontologies in a client
Ontologies in a server
Compare and Synchronize ontologies in a client and a sever
Show information of ontologies
NEW1
NEW2
NEW3
super
sub
is-a

Figure 4. A snapshot of Ontology Manger.

3) If the ontology has been updated by another developer, he analysis the changes by comparing the ontology with old versions of it. The ontology manager supports him by showing the changes and its influence.

4) After the analysis, the developer edits the ontology. And then, he uploads the edited ontology to shared space and unlocks it.

4. DISTRIBUTED CONSTRUCTION USING HOZO

Here, the authors summarize how Hozo supports distributed and collaborative construction of ontologies.

4.1 Dependency Management of Ontologies

4.1.1 Ontology Server and Ontology Manager

The ontology manager (see Figure 4) acts as a bridge between the personal space (in a client) and the shared space which the ontology server provides. It carries out the following functions:

1) To show the latest information on the ontologies such as "updated", "locked by another developer" and so on.

2) Access control to ontologies (lock and unlock)

3) Version management of ontologies

4) To search concepts defined in other ontologies

5) Synchronize ontologies in client with ontologies in server

4.1.2 Import concepts form other ontologies

When the developer find reusable concepts defined in other ontologies which are published in the server by other developers, he can import them to his ontology. The ontology manager supports him to import the concepts through *Import Dialog* of the ontology manager (see Figure 5). The dialog shows concepts in the selected ontology by tree structure based on *is-a* relation of them, and the developer selects a concept which he wants to import to his ontology. And then the system finds all the concepts depended by the selected concept form its dependencies[3], and it is imported with them.

[3] For example, super classes of the selected concept, and concepts which the selected concept refers to.

In the ontology editor, imported concepts are represented with different color from other concepts, and the developer cannot modify[4] them to keep consistencies of ontologies.

4.2 Harmonizing Interrelated Ontologies in their Conceptual Dependencies

4.2.1 Checking changes of depended ontologies

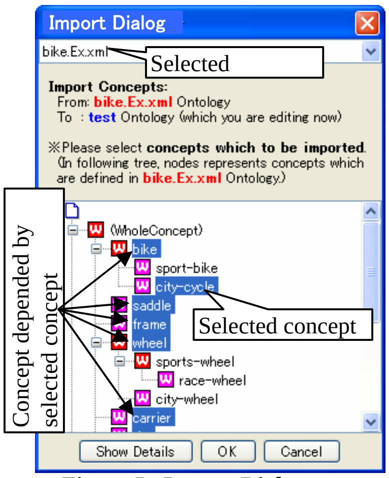
Ontology Manager shows developers which ontology has been changed. To maintain the consistency of dependency, the developer should get more information on, for example, that what concepts/slots in the depended ontology have been changed and which concepts in his ontology are influenced by the changes. Hozo shows such information on the tracking pane and the browsing pane of its ontology editor.

The tracking pane lists the changes in depended ontologies which influence on his ontology (see Figure 6). Those changes are classified in three types (deleted, modified and added), and their types are represented by icons. The changes are shown by nodes with icons in tree structure, and the developer can know which concepts are influenced by the change through child nodes of the nodes. By clicking a node representing a concept, the selected concept in the ontology is pointed in the browsing pane of ontology editor.

In the browsing pane (see Figure 7), the ontology is visualized in network structures, and the changed concepts are represented by same icons[5] as tracking pane shows. When the developer selects a changed concept, the concepts influenced by the change are highlighted on the browsing pane. And then, if the change type of the selected concept is modification, the details are shown.

4.2.2 Modifying the ontology to keep the consistency

To keep the consistency of the ontology, Hozo suggests possible countermeasures for coping with each of the changes to the developer. These countermeasures are devised through our investigation on conceptual dependencies of ontologies and the change type of imported concepts [1].


Import Dialog

Import Concepts:
From: bike.Ex.xml Ontology
To : test Ontology (which you are editing now)

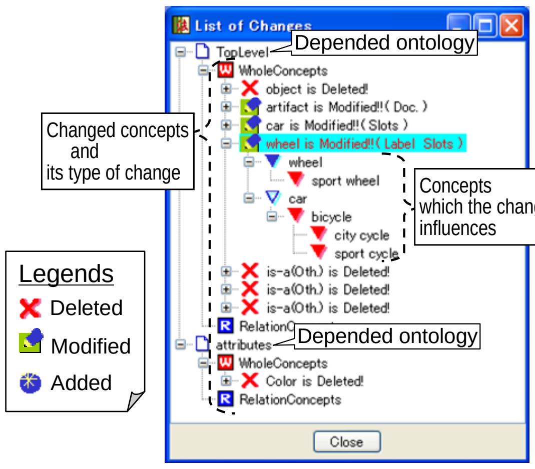
※Please select concepts which to be imported.
(In following tree, nodes represents concepts which are defined in bike.Ex.xml Ontology.)

Figure 5. Import Dialog.

In the beginning, Hozo shows the developer two major strategies: to accept the change and to reject it. The latter implies to redefine the changed concept in his ontology. For example, if the change type is modification of an imported concept, acceptance of the change corresponds to replace the imported concept with the

[4] The developer can use imported concepts to define another concept. For example, he can define sub classes of them.

[5] On the browsing pane, sky blue nodes represent imported concepts from depended ontologies. Therefore, only sky blue nodes can have the icons because the changes appear only on the imported concepts in the distributed ontology development.


List of Changes
Depended ontology
Changed concepts and its type of change
Concepts which the change influences
Legends
Deleted
Modified
Added
Close

```
TopLevel
  WholeConcepts
    object is Deleted!
    artifact is Modified!!( Doc. )
    car is Modified!!( Slots )
    wheel is Modified!!( Label Slots )
      wheel
        sport wheel
      car
        bicycle
          city cycle
          sport cycle
    is-a(Oth.) is Deleted!
    is-a(Oth.) is Deleted!
    is-a(Oth.) is Deleted!
  RelationC
attributes
  WholeConcepts
    Color is Deleted!
  RelationConcepts
```

Figure 6. Tracking Pane.

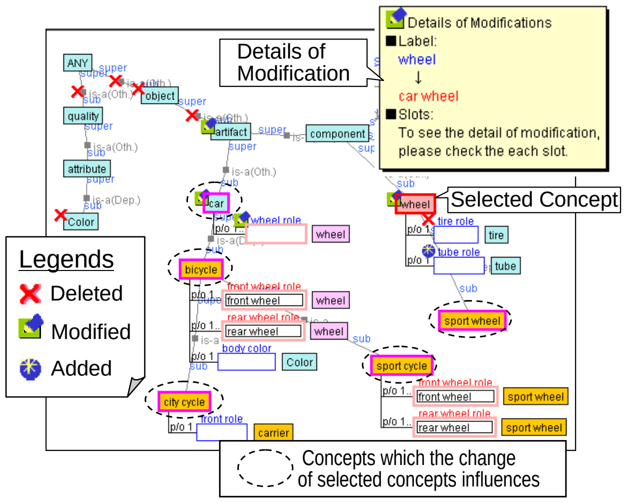


Figure 7. Representation of changes on Browsing Pane.

modified one. And, if the change type is deletion of imported concept, the acceptance corresponds to deletion of the concept. The developer can apply these countermeasures by selecting it through a pop up menu on the browsing pane. After applying countermeasures, he edits his ontology for coping with the change if necessary. In that case, it is helpful to him that the system shows the concepts influenced by the change. Furthermore, if he needs advanced strategies, the system shows him all countermeasures[6] with their details on a harmonizing pane.

5. Related Works

PromptDiff includes a version-comparison algorithm and enables users to view the differences between the versions [5]. It takes same approach with us, but it does not support distributed development discussed section 2. DILIGENT [6] and ONKI [7] supports distributed development of ontology through shared space for ontologies in the same way with Hozo. But they do not have functionalities to suggest countermeasures for coping with each of the changes to the developer when depended ontologies are modified. KAON and ours focus on that changes in one ontology can cause inconsistencies in other dependent ontologies. And, in order to ensure their consistencies, they propose deriving evolution strategies [8]. But it does not provides strategies which reduce the influences against the changes although Hozo suggests them (e.g. deletion of a concept can be canceled by redefining it in another ontology). The difference is caused by different treatment of relationship between depended ontologies and dependent ontologies.

6. CONCLUSION AND FUTURE WORK

In this paper, the authors discussed some functionalities of Hozo to support distributed ontology construction. Harmonization of ontology is an essential issue especially in a distributed development. Our system contributes to resolving the issue based on management of dependencies between ontologies. The functionalities can support to construct a single ontology by many developers collaboratively.

As future work, the authors plan to enhance our system according to the following future plan: (1) Maintenance of consistency among ontologies and its instance models based on our framework. (2) Supporting native OWL without transformation of file format. (3) Functionality to deal with OWL imports which refer to OWL files that are kept elsewhere. (4) Functionality for checking the ontology with a reasoner.

[6] We have not implemented some of advanced countermeasures yet. But, we suppose the two major strategies are enough for coping with the change in a lot of cases.

7. ACKNOWLEDGMENTS

We are grateful to Mr. Mamoru Ohta for his support to implement our system.

8. REFERENCES

[1] E. Sunagawa, et al.: An Environment for Distributed Ontology Development Based on Dependency Management, Proc. of ISWC2003, pp. 453-468, 2003.

[2] Kozaki K., et al.: Hozo: An Environment for Building/Using Ontologies Based on a Fundamental Consideration of "Role" and "Relationship", Proc. of EKAW2002, pp.213-218, Siguenza, Spain, 2002

[3] Sunagawa, E., et al. :Organizing Role-concepts in Ontology Development Environment: Hozo, Proc. of 2005 AAAI Fall Symposium on Roles, an interdisciplinary perspective, 2005

[4] Kozaki K., et al.: Fundamental Consideration of Role Concepts for Ontology Evaluation, Proc. of EON2006 Edinburgh, United Kingdom, May 22, 2006

[5] Noy, N., et al.: Tracking Changes during Ontology Evolution, Proc. of ISWC2004, Hiroshima, Japan, pp.259-273, 2004

[6] Tempich, C., et al.: An Argumentation Ontology for DIstributed, Loosely-controlled and evolvInG Engineering processes of oNTologies (DILIGENT), Proc. of ESWC2005, Greece, pp. 241-256, 2005

[7] Valo, A., Hyvonen, E. and Komurainen, V.: A Tool for Collaborative Ontology Development for the Semantic Web, in: Proc. of DC 2005, Madrid, Spain, 2005.

[8] Stojanovic, L., Maedche, A., Motik, B. and Stojanovic, N., User-driven Ontology Evolution Management, Proc. of EKAW 2002, Madrid, Spain, pp. 285-300, 2002