A Framework for Cooperative Ontology Construction Based on Dependency Management of Modules

Kouji Kozaki, Eiichi Sunagawa, Yoshinobu Kitamura and Riichiro Mizoguchi

The Institute of Scientific and Industrial Research (ISIR), Osaka University
8-1 Mihogaoka, Ibaraki, Osaka, 567-0047 Japan
{kozaki, sunagawa, kita, miz}@ei.sanken.osaka-u.ac.jp

Abstract. To construct large scale ontologies, two major approaches are discussed by many researchers. One is a cooperative construction of ontologies, and the other is a modularization of ontologies. To combine these two approaches, this paper discusses a framework for supporting cooperative ontology construction based on dependency management among modularized ontologies. In such a situation, one of the key issues is the maintenance of consistency among inter-dependent ontologies because each ontology is revised asynchronously by different developers. In order to realize consistent development of ontologies, the framework provides two functions: to manage the dependencies between ontology modules and to keep and restore consistencies between them when they are influenced by changes of other modules. Furthermore, we outline an implementation of our framework in our environment for building/using ontology: Hozo.

Keywords: Cooperative ontology construction, Distributed development, Dependency management

1 Introduction

Ontological engineering has changed considerably for these years. Many systems have become to deal with multiple and dynamic ontologies rather than single and static ones. This trend is and will be accelerating because of the advancement of Semantic Web whose characteristic is decentralized. On the web, ontologies will be scattered from server to server and referred to by one another. For example, ontology creators and service providers would search and compile several ontologies on the web, and then, adapt them to their own needs. Especially, to construct large scale ontologies efficiently, many researchers discuss a modularization of ontologies [1]. Such modularized ontologies are treated meaningfully in every phase of the development process. At the beginning of ontology development, developers need to determine the scope of the ontology, and next, consider reuse of existing ontologies [2]. In these phases, dividing the target ontology into modules helps the developers to understand a total picture of the conceptual hierarchy particularly in a large scale ontology. And, it also helps to determine the scope of application of the reused ontology. In a phase of construction and maintenance, it forms the basis of

cooperative development. Furthermore, after publication of the ontologies, a developer of another ontology can reuse them as his/her own modules easily without carving out them from their source ontology if it is divided into modules in a reasonable manner.

In this research, we focus on the phase of construction and maintenance and discuss a framework for supporting cooperative ontology construction based on modularization of ontologies in a distributed environment. In such a situation, one of the key issues is the maintenance of consistency among inter-dependent ontologies because each ontology is revised asynchronously by different developers. In order to realize consistent development of ontologies, the framework has to support two functions: to manage the dependencies between ontologies and to keep and restore consistencies of them when they are changed[3]. This paper overviews the framework for distributed and cooperative ontology development based on dependency management of modularized ontologies and explains how the framework supports to keep and restore consistencies of the modules in the development processes. In this work, we have reconsidered the prototype system in previous work and improved its implementation. The remainder of this paper is organized as follows. Section 2 discusses the underlying distributed and cooperative ontology development we assume. In section 3, we summarize a flow of the distributed and cooperative ontology construction and discuss how to support each construction process in our framework. Section 4 introduces the implementation of our framework in our environment for building/using ontology: Hozo. In section 5, we discuss some related work followed by a summary of future work in section 6.

2 Distributed and Cooperative Construction of Ontology

We assume a situation where several modularized ontologies are constructed separately in a distributed environment and 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 (Fig.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 Fig.1, Ontology A is the depended ontology of Ontology B, and Ontology B is the dependent ontology of A. We call a development of ontologies in such a manner distributed ontology development.

In the distributed ontology development, developers construct multiple ontology modules in cooperation among the developers. They can reuse published modules of other ontologies if possible. It is a common way for ontology development to import 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 sometimes updated without considering how ontologies depend on them would be influenced by their changes because authorities for maintenance of the ontologies are separated and distributed to

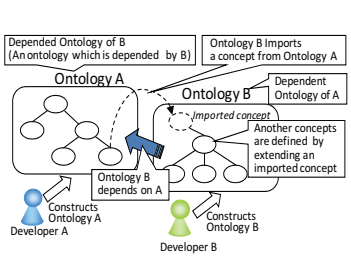

Fig.1. Distributed Ontology Development.

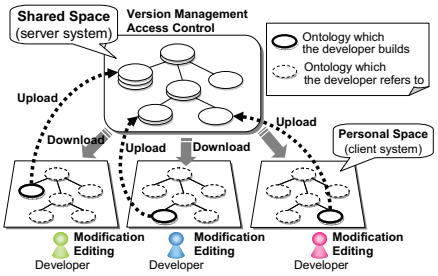

Fig.2. A Conceptual Framework for Distributed Ontology Development.

each developer. Therefore, when a developer changes his/her ontology, the change influences on its dependent ontologies. In many cases[1], such a change may cause inconsistencies among the ontologies. For consistent development of ontology modules, a system should manage dependencies among them and support their developers to harmonize them. Based on this observation, we have investigated how to manage the dependency among ontology modules and how a change of one ontology influences on others through its dependencies. And we have developed a framework for cooperative ontology construction in harmony among depended/dependent ontologies. Next section discusses the framework.

3. A Framework for Cooperative Ontology Construction based on Dependency Management of Modules

3.1. Flow of Distributed Ontology Development

Fig.2 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 open to other developers. The other is local (personal) spaces, 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, the target ontology can be regarded as a system of interrelated ontology modules 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

[1] We assume early stage of ontology development by trial and error.
[2] The same component ontology may be constructed by several developers asynchronously.

to. The distributed ontology development proceeds with the repetition of the following steps;

1. A developer gets latest information on ontologies which he builds or refers to from the ontology server. He downloads (updates) them from the shared space to the personal space (client) through an ontology manager. If it is needed, he locks ontologies to avoid simultaneous modification of the same ontology by others.
2. The developer analyzes changes in the updated ontologies and evaluates whether the changes are influencing on consistency of the ontology which he is constructing.
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. The framework helps such a 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 as a result. Then the dependency between his ontology and the referred ontology through the imported concepts is managed by the functions of dependency management.
5. After editing, the developer publishes his ontology by uploading (committing) it to the shared space. Then, he unlocks the ontology if he allows others to edit it.

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

We suppose another cooperative development process such as constructing a single ontology by many developers. Our distributed ontology development also can support such a 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 (updates) it to his personal space.
3. If the ontology has been updated by another developer, he analyses the changes by comparing the ontology with its old versions. The change analysis function of the framework supports him by showing the changes and its influence.
4. After the analysis, the developer edits the ontology. And then, he uploads (commits) the edited ontology to shared space and unlocks it.

To support the distributed and cooperative ontology construction discussed above, the framework provides four functions:(1)sharing ontologies on the shared space under version management and access control, (2)dependency management among modularized ontologies, (3)analysis of changes and their influences, and (4)suggestion of possible countermeasures for coping with each of the changes to keep and restore consistencies. We discuss the details of these functions in the following sections.

3.2. Version Management and Access Control of Ontologies

As a basic infrastructure for cooperative ontology construction, our framework provides the following two functions:

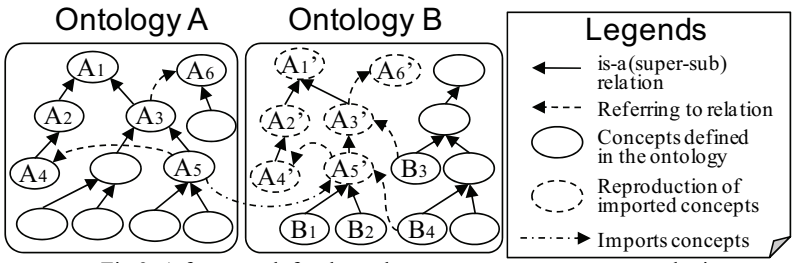

Fig.3. A framework for dependency management among ontologies.

Version Management: When the user uploads his ontology on the shared space (server), the old version of the ontology is moved to a backup space in the server. It is managed with its updated time and name of the developer, and it may be replaced by the latest version when the user requires.

Access Control: The server provides a mechanism for locking / unlocking ontologies to avoid that an ontology is updated by different developers at the same time. Because the units to be locked are modularized ontologies, its influence on the cooperative construction is kept to a minimum.

3.3. Dependency Management among Ontologies

When an ontology imports concepts[3] from other ontologies, the dependencies among ontologies are managed using reproduction of the concepts to be imported. As an example, we assume Ontology B imports concept A_5 defined in Ontology A (Fig.3). Then all the concepts depended by A_5 are reproduced with relations among them, and Ontology B imports these reproductions. It means the system reproduce all definition[4] related to the concept. In this example, "the super concept of A_5" (A_3 and A_1), "the concept referred by A_5" (A_4), "the super concepts and referred concepts of them (A_1, A_3 and A_4)" (A_1, A_2 and A_6) and relations among them are reproduced, and Ontology B imports these reproductions (they are shown by A_1' to A_6'). As the result, Ontology A becomes the depended ontology of Ontology B, and Ontology B becomes the dependent ontology of A. These reproductions have same definition with their original but belong to dependent ontology.

In Ontology B, another concept might be defined in the ontology by extending the imported concepts. The ways are divided into two types: defining sub concepts of them (B_1 and B_2 in Fig.3) and referring to them as constraints (B_3 and B_4 in Fig.3). These two types are represented by *is-a* (super-sub) relations and *referring-to* relations between reproductions of imported concepts and concepts defined in the ontology. These reproductions are used to manage dependencies among ontologies and to identify changes of depended ontologies. Because the dependencies are managed using relations between imported concept and concepts defined in dependent ontology [3], multiple dependencies (e.g. A depends on B, and B depends

[3] In OWL, the users cannot import a single concept, but they can import a whole ontology. But in our framework, the users may import concepts partially.

[4] The definition of concepts consists of id, name, super concept, comment, and slots.

on C) and circular dependencies (e.g. A depends on B, and B depends on A) can be managed by this framework.

3.4. Analysis of Changes and Their Influences

When a depended ontology is changed, the changes are analyzed by comparing its reproductions of imported concepts in the dependent ontology and their original concepts in the depended ontology. The types of changes are as follows:

1. If the original concept is not found[5] in the depended ontology, it means the concept was deleted.
2. If the definition of the original concept is different from that in the reproduction, it means the concept was modified.

The influences of the changes are analyzed by tracing the relations of reproductions whose original concept is changed. In Fig.3, we assume A_2 in Ontology A has been deleted. It means original concept of A_2' in Ontology B has been changed, and the change influences on A_4', A_5', B_1, B_2 and B_4 through their relations.

This analysis procedure is applicable to analyze the difference of an ontology and its old version. In the case, the comparison is done through all concepts and relations in the ontology. And the types of changes are as follows:

1. If a concept/relation is found only in the new ontology, it means the concept/relation was added in the new ontology.
2. If a concept/relation is found only in the old version, it means the concept/relation was deleted in the new ontology.
3. If the definition of a concept/relation in the new ontology is different from the same concept/relation in the old version, it means the concept/relation was modified.

The users can cancel part of the changes if it is needed.

3.5. Maintenance of Dependencies among Ontology Modules

For prevention and resolution of inconsistency in dependencies between ontologies, we can consider two approaches to maintain the consistencies. One is to restrict the change which influences on others seriously. Such a restriction helps developers to avoid inconsistency proactively. The other approach is to adapt the influences of the change and restore the consistencies by modifying influenced ontologies. We have taken the latter approach and have come up with five kinds of countermeasures for coping with each of the changes to keep and restore consistencies:

1) To accept the change

- **1-1) To modify the influenced ontology to be compliant with the change;** The developer makes agreement on the change of the ontology and modifies his/her ontology depending on it for adapting to the changed ontology.
- **1-2) To leave the depending ontology influenced by the change;** In some cases, the influenced ontology can be left unmodified, as the changed ontology does not contradict it.

[5] Because it is compared according to id of concepts, the change of id is regarded as a deletion and an addition of the concept.

2) To refuse the change

2-1) To modify the influenced ontology for compensation of the change; As far as preserving the consistency of the dependency, the developer modifies his/her ontology against the change to cancels the influence of the change.

2-2) To stay compliant with the previous version of the changed ontology; Under controlling versions of the ontologies, the dependency is kept without any modification. After that, when the influencing ontology would be changed so as to be acceptable, the dependent one would adapt to the change and the consistency would be recovered.

2-3) To break the dependency; In order to make the influenced ontology independent of the changed one, reproductions of imported concepts whose change influences on it are redefined as new concepts in the dependent ontology. It implies the dependency on the influencing ontology is broken.

1-1), 1-2) and 2-1) correspond to replacement reproductions of imported concepts with new reproductions based on changed concepts. In 1-1) and 1-2), the developer modifies his/her ontology after the replacement. 2-2) corresponds to do nothing, and 2-3) corresponds to redefinition as discussed above.

We investigated the patterns of the change and the possible way of modification to keep the consistency of the dependency for each pattern. The patterns of the change include the cases where a concept has been deleted, the label has been changed, a slot of a concept has been deleted and so on. For all the cases, we come up with 17 types of change of concepts according to the kind of dependency. And, as the countermeasures for the change, we devised 71 ways of modification. The influenced ontology is modified based on these countermeasures. The details are discussed in our previous work [3]. Though the target of our investigation is a frame language used in Hozo, it can be translated into OWL. Therefore, we suppose most of the patterns are applicable to OWL.

4. Implementation

We have implemented our framework in our environment for building/using ontology: Hozo. Here, we summarize how Hozo supports distributed and cooperative construction of ontologies.

4.1. Overview of Hozo

The features of Hozo include 1) Supporting role representation [4, 5], 2) Visualization of ontologies in a friendly GUI, 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 (Fig.4). The ontology editor provides a developer with a graphical interface through which they can browse and modify an ontology locally. The instance models can be developed using Model Editor which is a sub system of the Ontology Editor. The ontology server stores and manages ontologies under access control and version management. Developers can access and browse them through the

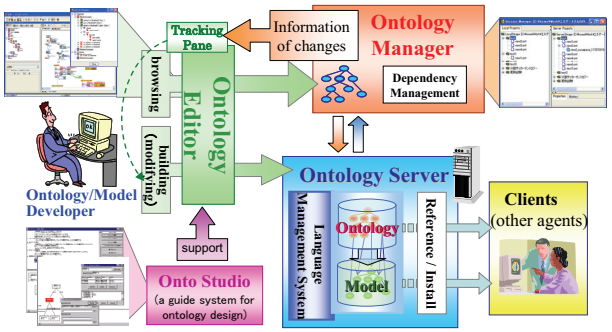

Fig.4. Architecture of Hozo

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 [6], and RDF(S). It also can import OWL partially[6]. The latest version of Hozo is published at the URL: http://www.hozo.jp.

4.2. Version Management and Access Control through Ontology Manager

Hozo can use a general file server as the ontology server. It uses a shared folder on the network or a WebDAV folder to store and share ontologies. The ontologies are managed by changing filenames and storing folders according to their dependencies and versions. The users can share ontologies through a local area network or the Internet. This simple mechanism makes it possible for the users to set up their own ontology server easily without complicated procedures. The user also can switch the ontology server to the other if necessary.

The ontology manager (Fig.5) 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 ontology modules such as "updated", "locked by another developer" and so on.
2. Access control to ontology modules (lock and unlock)
3. Version management of ontology modules
4. To search concepts defined in other ontology modules
5. Synchronize ontology modules in clients with those in the server

4.3. Dependency Management among Ontology modules

When the developer finds reusable concepts defined in other ontologies which are published in the server by other developers, he can import them to his ontology. The

[6] The OWL import mechanism is under improvement.

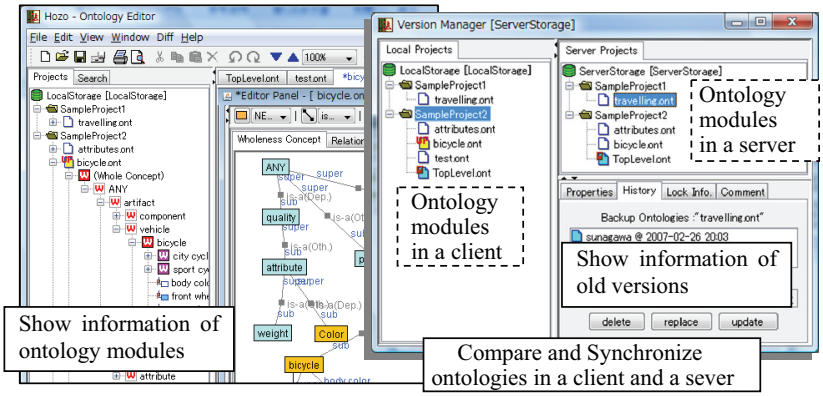

Fig.5. A snapshot of Ontology Manger.

ontology manager supports him to import the concepts through *Import Dialog* of the ontology manager. The dialog shows concepts in the selected ontology by tree structure based on *is-a* relation of them, and the developer selects concepts which he wants to import to his ontology. Then, the system finds all the concepts depended by the selected concepts, forms its dependency relations according to their relations, and finally reproductions of them are imported to his ontology through the procedure discussed in section 3.3. In the ontology editor, reproductions of imported concepts are represented with different color from other concepts, and the developer cannot modify[7] them to keep consistencies of ontologies.

4.4. Analysis of Changes of Depended Ontologies and Their Influences

Ontology Manager shows developers which ontology has been changed. To maintain the consistency of dependency, the developer should get more information on, for example, 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 (Fig.6). Those changes are classified in three types (deletion, modification and addition), and their types are represented by icons. The changes are shown by nodes with icons in a 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 (Fig.7), the ontology is visualized in network structures, and the changed concepts are represented by the same icons[8] as

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

[8] In 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.

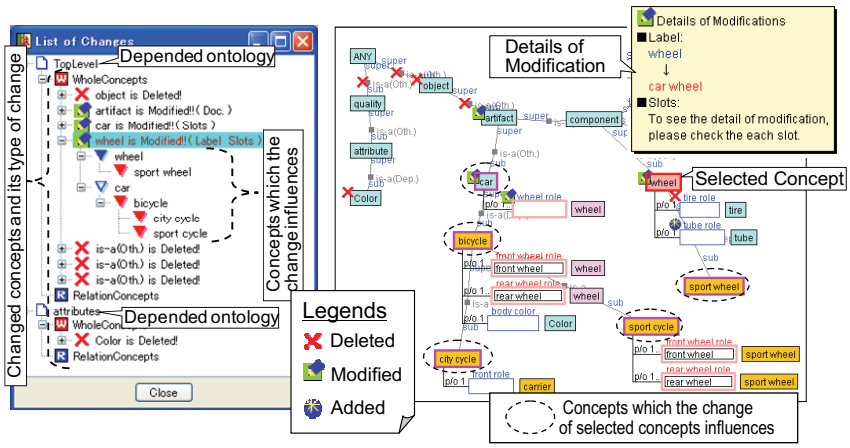

Fig.6. Tracking Pane. Fig.7. Representation of changes on Browsing Pane.

tracking pane shows. When the developer selects a changed concept, the concepts influenced by the change are highlighted in the browsing pane, and then, if the change type of the selected concept is modification, the details are shown.

4.5. 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 discussed in section 3.5. In the beginning, Hozo shows developers two major strategies: to accept the change and to reject it. The former corresponds to 1-1), 1-2) and 2-1)[9] discussed in section 3.5. The difference among them depends on the way of modifications after the acceptance of the change. The latter corresponds to 2-3) and implies to redefine the changed concept in his ontology. If the user chooses neither to accept nor to reject the change, it corresponds to 2-2).

For example, if the change type is modification of an imported concept, acceptance of the change corresponds to replacement of the imported concept with the modified one. If the change type is deletion of imported concepts, the acceptance corresponds to deletion of them. Developers can apply these countermeasures by selecting it through a popup menu in the browsing pane. After applying countermeasures, he edits his ontology for coping with the change if necessary. In such a case, it is helpful for him that the system shows the concepts influenced by the change. Furthermore, if he needs advanced strategies, the system shows him all countermeasures[10] with their details in a harmonizing pane.

[9] This strategy means that the user accepts the change and then he/she modifies against the change to cancel the influence of it.

[10] 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.

5. Related Work

Protégé has a semi-automatic tool for ontology merging and alignment named PROMPT [7]. It performs some tasks automatically and guides the user in performing other tasks. PROMPT also detects possible inconsistencies in the ontology, which result from the user's actions, and suggests ways to remedy them. For ontology evolution in collaborative environments [8], Protégé provides two functions: Change-management plugin which stores a list of class-wide changes with annotations and shows history of the change to the user, and Client-Server mode which support synchronous ontology editing by multiple users. SWOOP [9] also supports collaborative annotation for discussing and version control using change logs. But they does not support distributed construction of modularized ontologies discussed in section 2. Their methods for version control are also different form Hozo. They use change logs, but Hozo does not use them and analyzes the changes by comparing ontology with its old version. The approach of Hozo is applicable to ontologies on the Web without their change logs.

DILIGENT [10] and ONKI [11] supports distributed development of ontology through shared space for ontologies in the way as Hozo. But they do not have functions to suggest countermeasures for coping with each of the changes to the developer when depended ontologies are modified. KAON and Hozo focus on that changes in an ontology can cause inconsistencies in other dependent ontologies. And, in order to ensure their consistencies, they propose deriving evolution strategies [12, 13]. 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.

[14] proposed algorism for modularization of OWL ontology. We have not considered how to modularize ontology. It is one of our future works.

6. Conclusions and Future Work

In this paper, we discussed a framework for distributed and cooperative ontology development. The maintenance of consistencies among modularized ontologies is an essential issue especially in a distributed development. Our framework contributes to resolving the issue based on management of dependencies between ontology modules. The same framework also can support to construct a single ontology by many developers cooperatively. Furthermore, we have implemented the framework in our ontology development environment: Hozo. It supports distributed and cooperative ontology construction by different developers through LAN and Internet. Its functions for distributed ontology construction have been used by some researchers and got favorable comments by them. The latest version of Hozo is open to the public on the website (http://www.hozo.jp).

As future work, the authors plan to enhance our system according to the following future plan: (1) Functions to deal with OWL ontology. For example, we suppose to use OWL properties such as owl:imports and owl:priorVersion for management of

ontology on the Web. (2) Evaluation and reconsideration of strategies for keeping consistencies. (3) Consideration of appropriate modularization. (4) Maintenance of consistency among ontologies and its instance models based on our framework.

Acknowledgments

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

References

1. Seidenberg, J., Rector, A.: Web ontology segmentation: Analysis, classification and use. In: 15th International World Wide Web Conference, Edinburgh, Scotland (2006)
2. Noy, N.F., McGuinness D.L.: Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 (2001)
3. Sunagawa, E., Kozaki, K., Kitamura, Y., Mizoguchi, R.: An Environment for Distributed Ontology Development Based on Dependency Management, In: 2nd International Semantic Web Conference, pp. 453--468, Florida, USA (2003)
4. Kozaki K., Kitamura, Y., Mizoguchi, R.: 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.
5. Mizoguchi, R., Sunagawa, E., Kozaki, K., Kitamura, Y.: A Model of Roles in Ontology Development Tool: Hozo. J. Applied Ontology (to appear)
6. Kozaki K., Sunagawa, E., Kozaki, K., Kitamura, Y., Mizoguchi, R.: Role Representation Model Using OWL and SWRL, In: 2nd Workshop on Roles and Relationships in Object Oriented Programming, Multiagent Systems, and Ontologies, Berlin (2007)
7. Noy, N.F., Musen, M.A.: The PROMPT suite: Interactive tools for ontology merging and mapping. International Journal of Human-Computer Studies, 59(6), pp.983—1024 (2003)
8. Noy N., Chugh A., Liu W. and Musen M.: A Framework for Ontology Evolution in Collaborative Environments. In: 5th International Semantic Web Conference, Athens, GA, USA (2006)
9. Kalyanpur, A., Parsia, B., Sirin, B., Cuenca-Grau, B., Hendler, J.:Swoop: A 'Web' Ontology Editing Browser, Journal of Web Semantics Vol 4(2), pp. 144-153 (2005)
10. Tempich, C., Pinto, H.S., Sure, Y., Staab, S.: An Argumentation Ontology for DIstributed, Loosely-controlled and evolvInG Engineering processes of oNTologies (DILIGENT). In: The 2nd European Semantic Web Conference, Greece, pp. 241-256 (2005)
11. Valo, A., Hyvonen, E. Komurainen, V.: A Tool for Collaborative Ontology Development for the Semantic Web, in: Proc. of International Conference on Dublin Core and Metadata Applications 2005, Madrid, Spain (2005)
12. Stojanovic, L., Maedche, A., Motik, B. Stojanovic, N: User-driven Ontology Evolution Management, Proc. of EKAW 2002, Madrid, Spain, pp. 285-300 (2002)
13. Maedche, A., Motik, B., Stojanovic, L., Studer, R., Volz, R. : An Infrastructure for Searching, Reusing and Evolving Distributed Ontologies, The Twelfth International World Wide Web Conference, Budapest, Hungary (2003)
14. Aquin M., Sabou M., and Motta E.: Modularization: a Key for the Dynamic Selection of Relevant Knowledge Components, The First Workshop on Modular Ontologies (2006)