# Ontology modeling in developing OSSs

Shingo Horiuchi, Masateru Inoue, Takashi Inoue and Tetsuya Yamamura
NTT Access Network Service Systems Laboratories
1-6 Nakase Mihama-ku, Chiba-shi, Chiba 261-0023 Japan
TEL: +81-43-211-3375 FAX: +81-43-211-2610
Email: {hor, inoue.masateru, inoue.takashi, yamamura}@ansl.ntt.co.jp

NTT Access Network Service Systems Laboratories

## Abstract

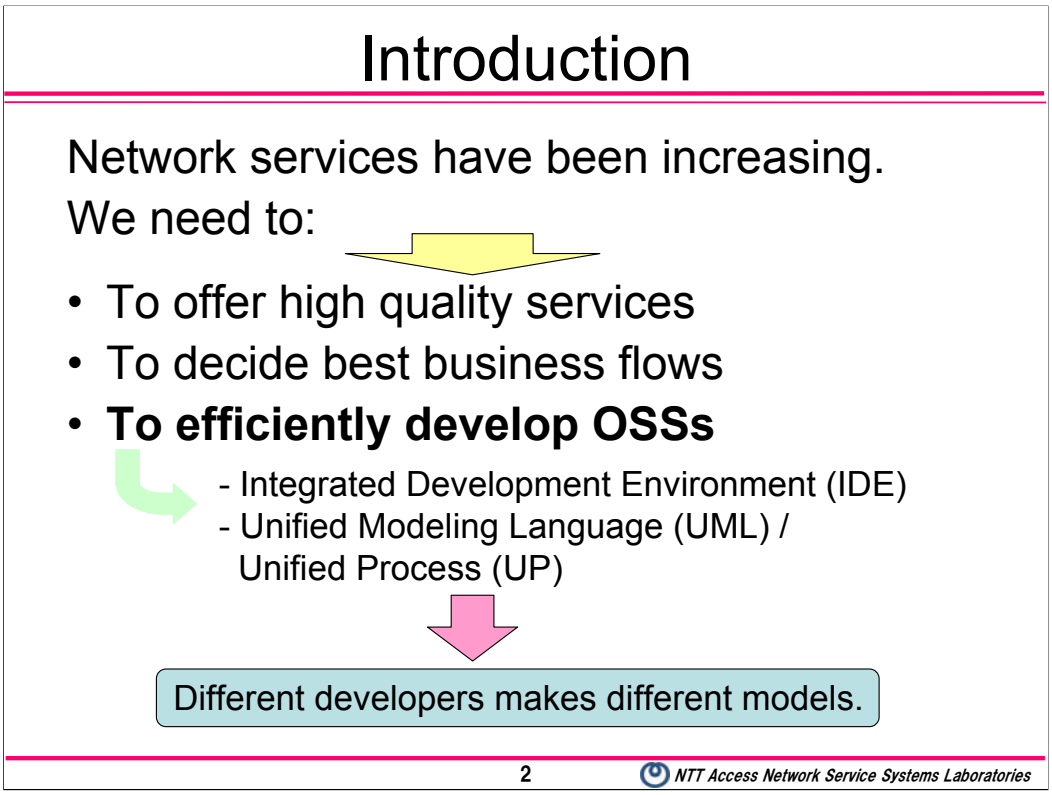
The increasing number of network services has highlighted the need to develop operation support systems (OSSs) for these services. It seems efficient to develop OSSs by reusing models of existing OSS developments and by sharing these models among developers.

Using ontologies to develop new OSSs would make it possible to find models similar to the ones to be developed.

The scheme described in our paper is as follows: using task/domain ontologies during the OSS development and reusing models from previous OSS developments. To show the effectiveness of task/domain ontologies, we have created a technical map concerning ontological technologies. We give an example of a task ontology for a Gigabit Ethernet - Passive Optical Network (GE-PON) OSS that demonstrates the reusability concept.

**Keywords:** OSS, Ontology, UML, Reuse of models

**Contact:** Shingo Horiuchi (hor@ansl.ntt.co.jp)



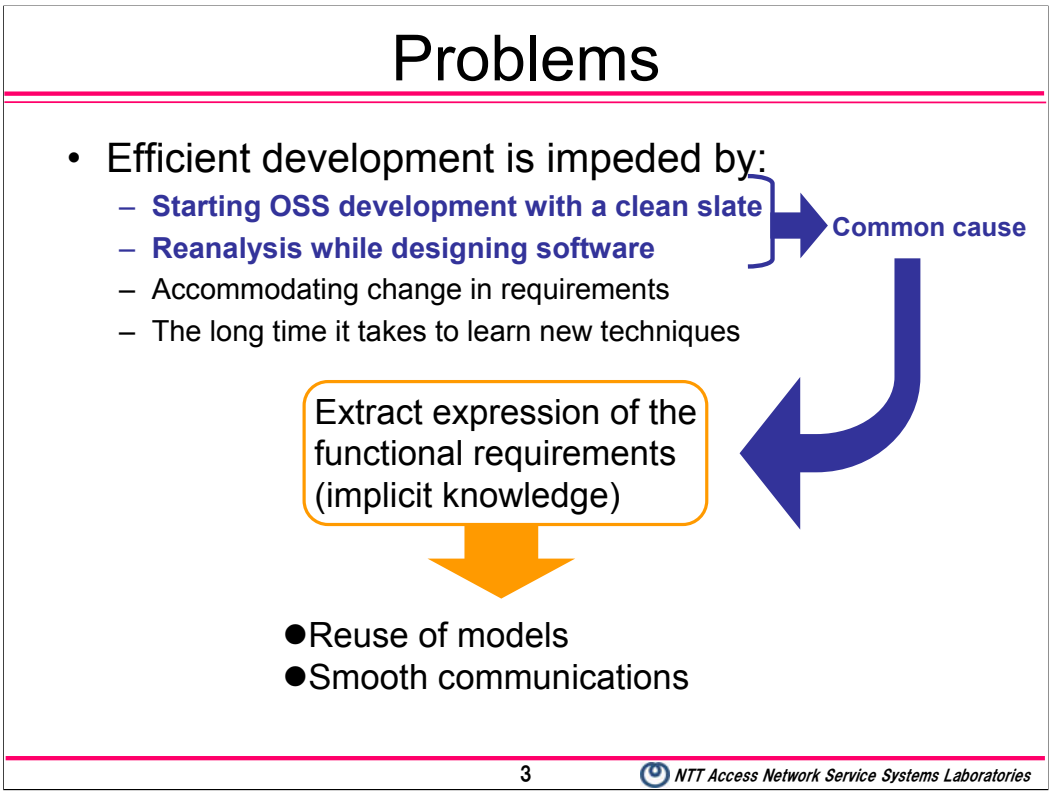

## Introduction

Network services have been increasing and becoming more diverse. For example, in Japan, it is possible to use fiber to the home (FTTH) services as well as asymmetric digital subscriber line (ADSL) services. Network service providers have reduced the price of these services, because competition is getting tough.

To compete effectively in the marketplace, it is necessary for a provider to offer high-quality services for its business users and to decide the best business flow to achieve a quick start of services. It is also necessary to efficiently develop operation support systems (OSSs) for these services [1].

To efficiently develop OSSs, developers use the Integrated Development Environment (IDE) and its development process, because it saves time when analyzing the software requirements. There are cases where many OSSs have been developed to manage various network elements. However, even though such OSSs have similar functions, it has been difficult to efficiently develop OSSs by exploiting the similarity of their functions.

For general software development, developers usually use the Unified Modeling Language/Unified Process (UML/UP) [2-3]. UML/UP provides a unified expression of the software models by which developers can graphically make models. However, there's no unified method to make the models. Therefore, different developers make different models of software development.




## Problems

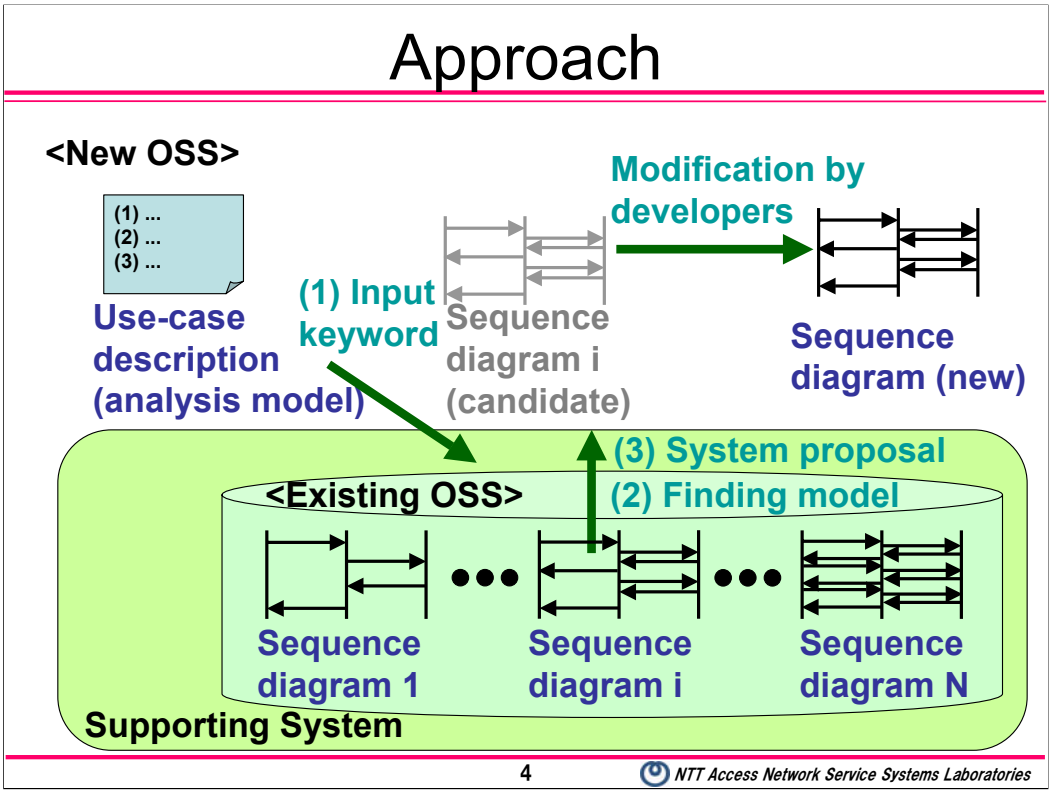
There are several problems that impair the efficiency of OSS development. The first one is that the development starts from a clean slate. This problem is caused by developers not making use of models similar to the ones used in existing OSS developments. For example, NE-OSS developers seldom use NE-OSS models developed for previous network elements, even though these NE-OSSs have the same functions. The second problem has to do with the reanalysis conducted during the design of the OSS, because the models aren't shared between the analysts and the designers. This is a very important problem because the designers and analysts are usually different people. The third problem is the difficulty in accommodating requirement changes in the analysis phase. The last one is the time it takes to learn new techniques, for example, Unified Process (UP) or Enterprise Java Beans (EJB).

The main subject in this paper is the first two problems, because they have a common cause of not having exactly described the models despite the use of UML. The models should be described exactly with integrated labels (terms) and should also be explicitly described with tasks of the implicit preconditions.

The aim in this work is to develop OSSs efficiently by creating smoother communications between clients and designers by introducing a regime of precise language in the functional requirements guiding the OSS development process and by reusing the models of the concepts included in the functional requirements.

For this purpose, it is important to formally express the implicit knowledge in order to identify and understand the concepts concerning the implicit knowledge.

To efficiently develop OSSs, we propose a support system that promotes reuse of similar models from existing OSS developments and sharing of models between the designers (developers in the design phase) and the analysts (developers in the analysis phase).




## Approach

From the requirements phase to the test phase, reuse of the models is promoted by using the dialogical system by which existing models that are similar to the new ones are used as support for making the new models.

During each development process phase, developers must clarify the models according to the levels of the phase. In this case, models that have similar functions help with the clarification.

The system supporting reuse has procedures for reusing models of OSS development as follows:

(1) Input keywords.

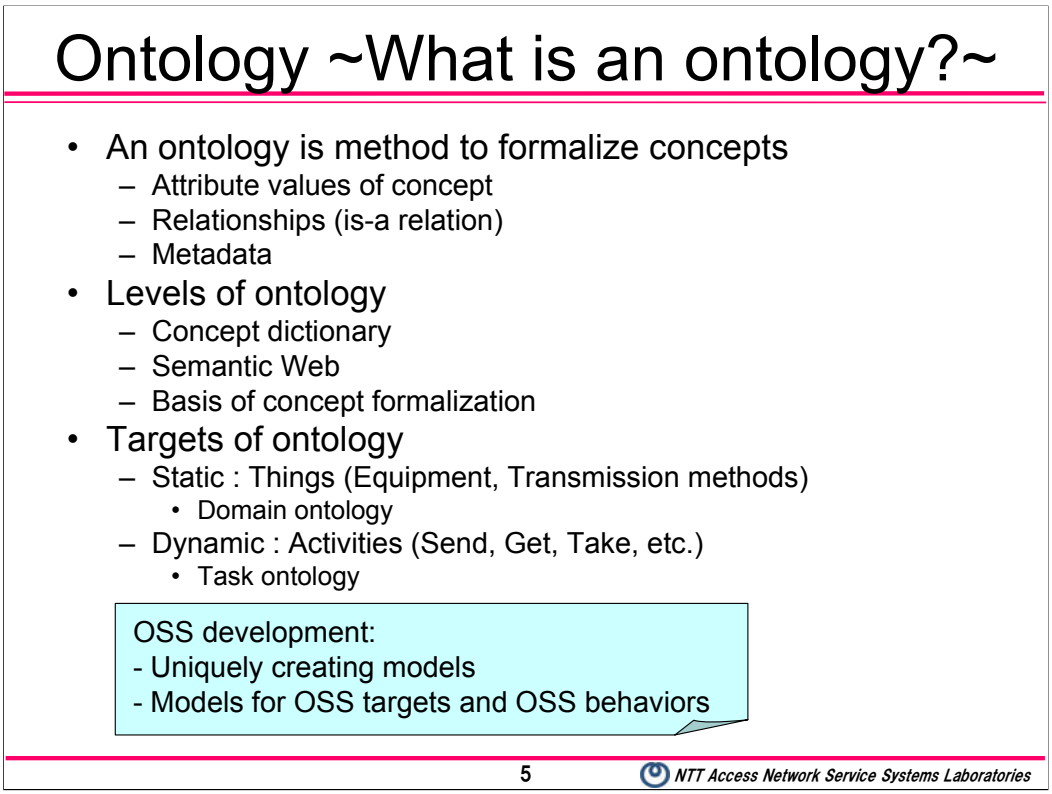
Developers input keywords that describe the incomplete requirements of the new OSS.

(2) Finding similar models.

The system searches for models of existing OSS developments that are similar to the one under development. To search for them, it is necessary to calculate the similarity between models. The proposed similarity is described on page 9.

(3) Proposal of models for reuse.

The system automatically proposes reusable models to the developers. After that, the developers describe the accurate requirements of the new OSS, based on the proposed models. For the system to be able to make an efficient proposal, it has to consider many things, for example, the ordering of the models it finds.




## Ontology ~What is an ontology?~

To identify the models, we should clarify the design rationale of why the labels (terms) are used in the models or what the implicit precondition for the dynamic models is. For this purpose, we should formalize the implicit knowledge, including the models with an ontology, which is useful for identification of the concepts. An ontology consists of concepts and the relationships among them. Examples of concepts are class, object, and attribute. Using an ontology makes it possible to find models similar to the ones in existing OSS developments and propose useful models to a designer.

Ontological technologies for formalizing concepts are quite diverse [4]. It is said that an ontology should provide guidelines for arranging huge quantities of information and should be useful for getting helpful information. In general, an ontology has three levels.

(1) Concept dictionary:

This is arranged as a dictionary, like a thesaurus, such as WordNet [5].

(2) Semantic Web:

This is described by Web Ontology Language (OWL) [6] and is used to efficiently access the information on the web. Many tools, such as DODDLE [7], have been developed and have a lot of applications.

(3) Basis of concept formalization:

This includes the upper, task, domain, and function ontologies.

During the development of an OSS, we must identify models that were made by other people. Then, we should make a unique ontology concerning these models. Therefore, we need an ontology for the basis of concept formalization, because not only should it be simply arranged according to the metadata but also should be uniquely defined by the different authors.

We desire to formalize the concepts included in the OSS's documents. Therefore, we have to deal with the equipment, the kinds of businesses targeted by the OSSs, and the behaviors of the OSS systems themselves. There is a domain ontology that formalizes static things, and a task ontology that formalizes dynamic activities.

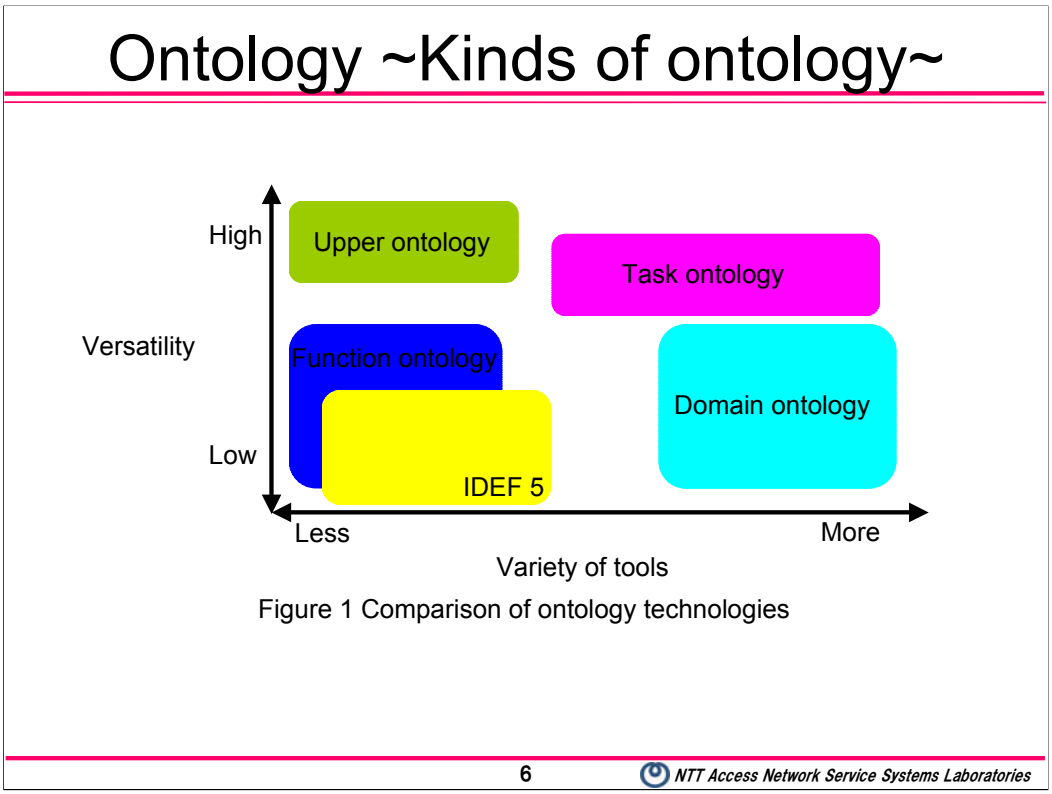
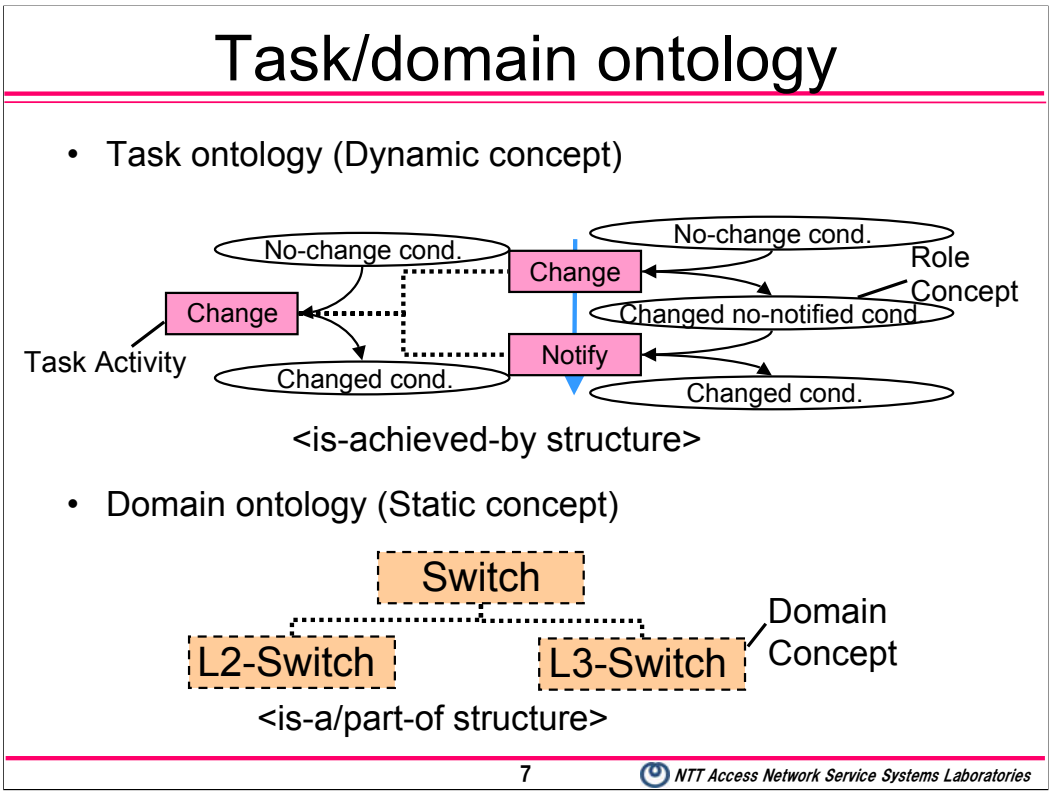



Figure 1 Comparison of ontology technologies

## Ontology ~Kinds of ontology~

We have studied many kinds of ontological techniques and assessed whether they meet our requirements (Fig. 1). To achieve the formulation of a system supporting developments, it's preferable to have ontological technologies that have a lot of tools. Although IDEF5 consists of a language and a method designed for creating, modifying, and maintaining ontologies, it has few applications.

In addition, we must be able to express the concepts concerning the development. The upper ontology is inadequate for arranging concepts embodied in actual equipment and businesses, because its purpose is only to make the classification categories when arranging the concepts. To express the concepts of the equipment the OSS targets, the domain ontology is best for expressing the relationship (is-a, part-of etc.) between concepts that are specific to the equipment or physical materials. In addition, the task ontology is adequate for the expressing the behavior of the OSS, for example, an optical line terminal (OLT) change function, because we can use the task ontology to formalize the relationship between the tasks that the behaviors consist of and the inputs/outputs of the tasks. We thus should select the domain and task ontologies and try to efficiently develop OSSs by using them [8].




## Task/domain ontology

The task ontology is a formalization of dynamic concepts, which are the problem-solving task flows concerning the business or system functions. It doesn't depend on the domain of the target problem, because it depends on the problem solving task flow.

The OSS processes start with the operator's inputs, the network alarms, and the service orders. The OSS behaviors are affected by the inputs and outputs. To formalize its behaviors, it is necessary to clarify how it should behave, according to the inputs and outputs and the behavioral structure of the business processes and/or the functions, in the development of the OSS.

The activity-first method (AFM) has been proposed as a means of clarify the behaviors of problem-solving tasks[9]. We think that a task ontology made by the AFM would be excellent in terms of the formalization of the task structures. A task ontology includes the role concepts that represent the relationship between the domain concept and the task. The role concepts are the roles that the object plays as the inputs or the outputs. Moreover, a task ontology is expressed with an 'is-achieved-by' structure. Thus, by using the AFM, we can describe the behaviors of the functions that OSSs have.

Domain ontology is the formalization of the relationship (ex. is-a, part-of) between the words that depend on the domain handled by the problem-solving task. For the development of OSSs, the domain is the network or business process that the OSS deals with, and the domain concepts are the names of the network services, the types of network equipment or the kinds of communication protocol. The difference between the domain ontology and the class diagrams of the object-oriented modeling or the standard or recommendation is whether or not there is an explicit description of the meanings. The domain ontology expresses these meanings.

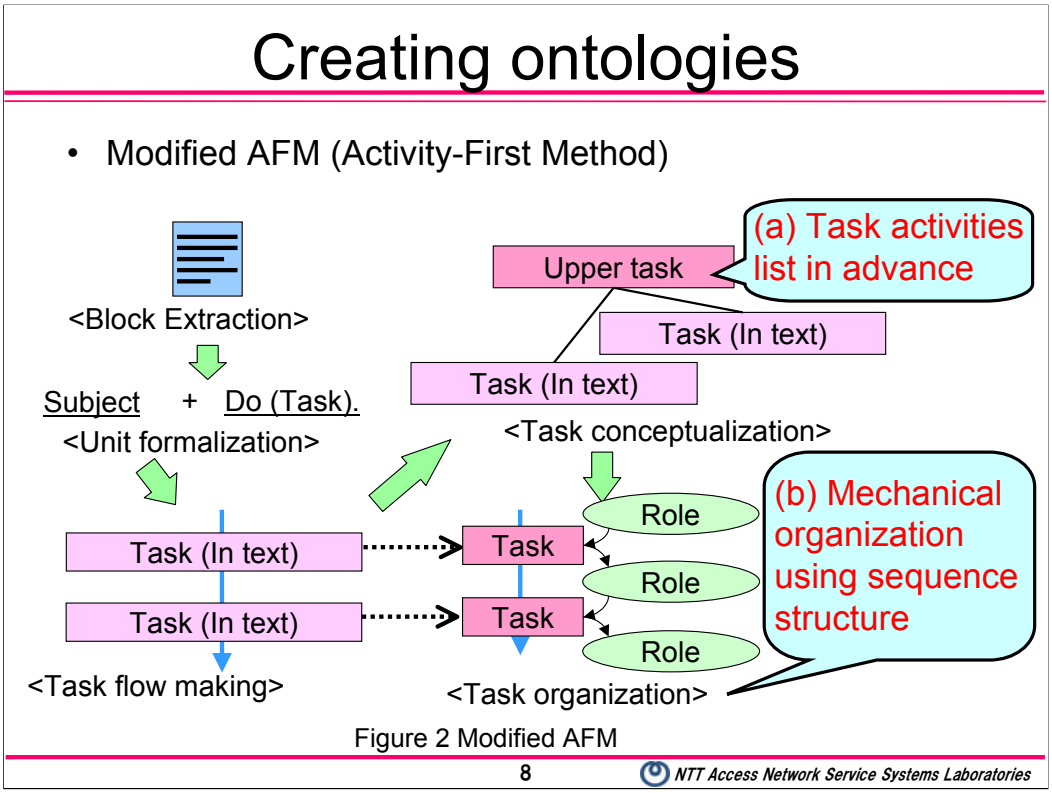


Figure 2 Modified AFM

## Creating ontologies

We decided to create a task ontology from OSS development documents by using the AFM. The AFM consists of several phases (Fig. 2) as follows:

(1) Unit formalization

(2) Task conceptualization

(3) Task organization

In the development of software like an OSS, we usually use the use-cases and the sequence diagrams that represent the models of the dynamic behaviors. In addition, the sequence diagrams are likely useful for describing the behaviors of the whole system. Therefore, we should create the task ontology based on the sequence diagrams. We also need to create the task ontology mechanically, so that the software developers can use it without having to know about the ontology itself. Accordingly, we propose modifying the AFM as follows [10-11].

(a) Task activities listed in advance

To make the task activities list for the models in advance, we use the verb that the task activities list includes for making the models in the developments, in order to mechanically integrate the task labels.

(b) Mechanical organization using sequence structure

We organize the flow of the tasks by using the sequence diagram's structure to create an ontology from the sequence diagrams. To determine who acts in the process, the sequence diagram decides the layer of the task structure.

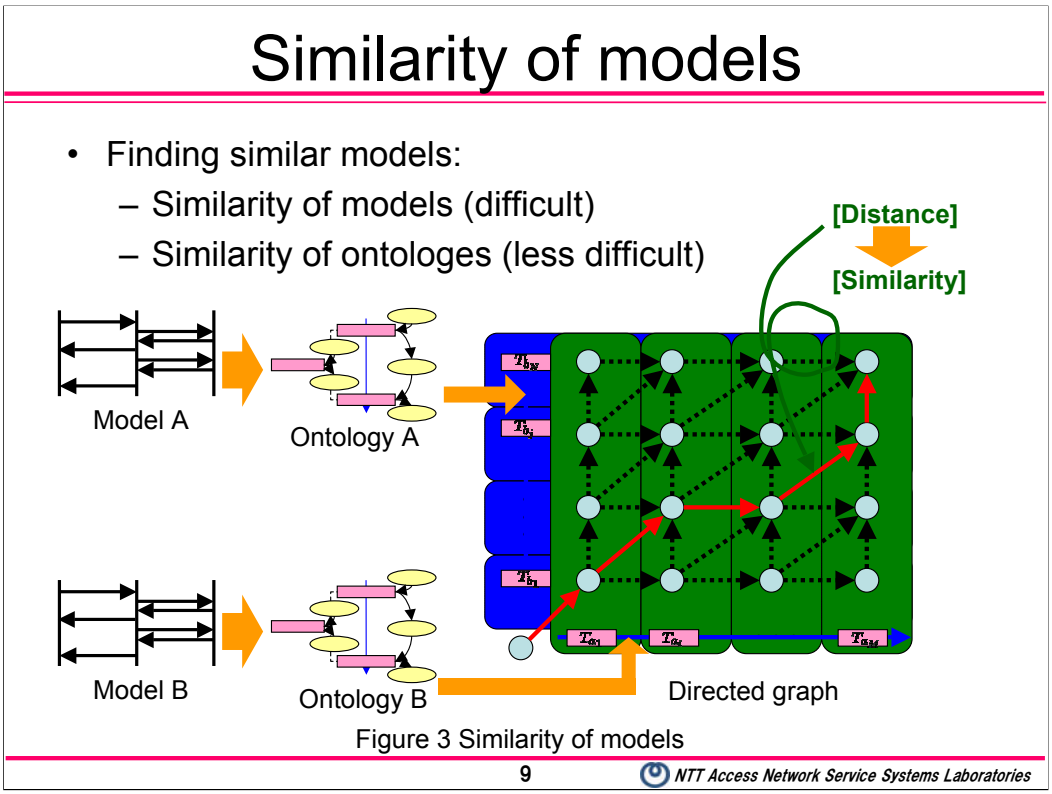



Figure 3 Similarity of models

## Similarity of models

To reuse models, we might consider calculating a similarity value that indicates the degree to refer to models when making new models. However, it is very difficult to calculate such a similarity because it depends on the tools used to make the models and the skills of the developers. In contrast, it is less difficult to calculate the similarity of ontologies that represent the essential meanings.

We have proposed criteria for the selection of such models, which is the similarity between the task ontologies calculated by dynamic programming matching technology and a shortest path searching algorithm like the Dijkstra method [11]. AS shown in Figure 3, to calculate the similarity, a directed graph is defined according to each task activity flow and by weights. The node of the graph is represented by the combination of the task activity of one ontology and the task activity of another ontology. The weight of the arc in the graph is also defined by the term (verb) similarities. We consider the degree of co-occurrence in the OSS development documents to be the term similarity. Note that there is another method to determine the term similarity[12], but its similarities can't reflect the term meanings of OSS developments.

After the similarities have all been calculated, the developers can use similar existing models as they find necessary. Consequently, they can save time instead of having to make the models from scratch.

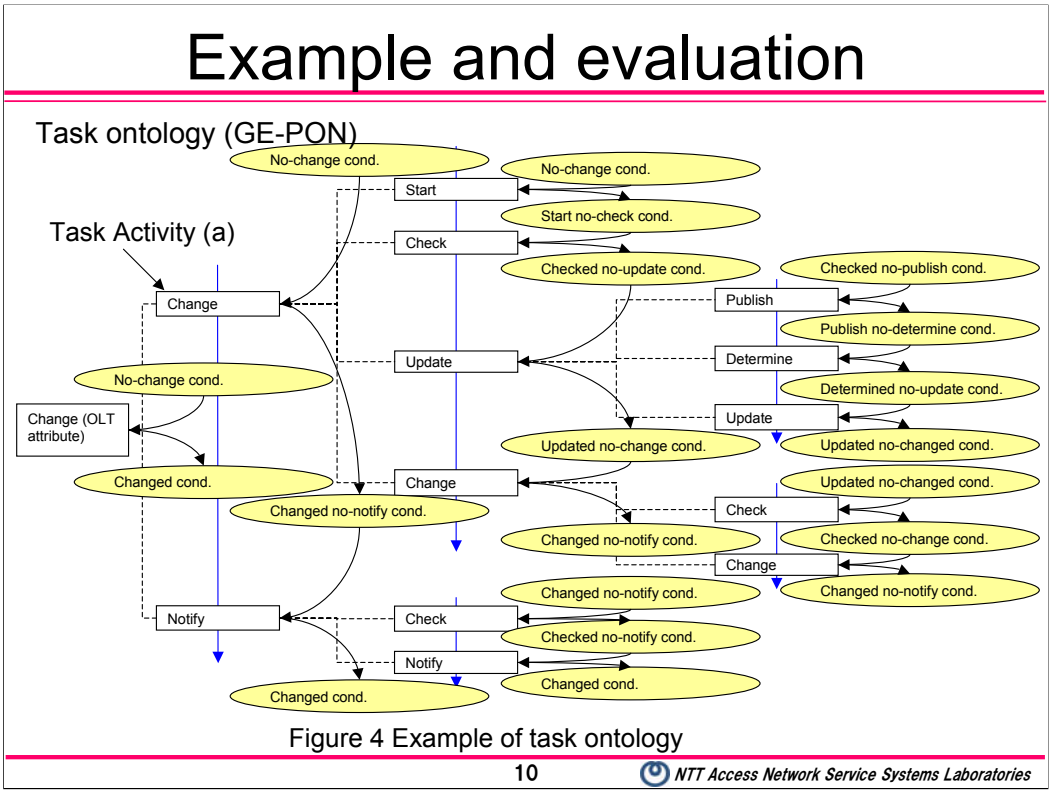



Figure 4 Example of task ontology

## Example and evaluation

We made a task ontology for about 70 functions of a gigabit ethernet - passive optical network (GE-PON) OSS. An example of the task ontology for a change function (change the Optical Line Terminal (OLT) attribute) is shown in Figure 4.

Each box represents the task activity, which is a verb, and each oval represents a role concept. The task activity on the left consists of a series of task activities along the arrow in sequential order. For example, the task activity (a) ‘Change’, in the figure, consists of subordinate task activities: start, check, update, and change.

We completely extracted the task activities that the function consists of, because the role in the middle of the two adjacent activities was defined. We also created layered task activities with an ‘is-a’ relationship when we created the task ontologies. The layered task activities can restrict the task activities in the task ontology with an ‘is-achieved-by’ structure; for example they restrict 218 task activities from the 481 task activities expressed in the documents of the GE-PON OSS.

This result means that the ambiguity of the task ontology is decreased. However, from this experiment, we realized that the structure of the task ontology depends on the grain size of the document in the OSS development. That is, when a development document is closer to one in the implementation phase, the task ontology has a higher tier structure. For example, in the requirement phase, the function Change (OLT attribute) consists of three task activities: Change (OLT attribute), Change and Notify on the left side of the Figure 4. In contrast, in the design phase, the function consists of the 14 task activities shown in Figure 4. Thus, it seems advisable to make models with the same grain size in the each phase of the development process.

# Further studies

- To reuse models
  - Mechanically make ontology
  - Cooperate with other IDE tools

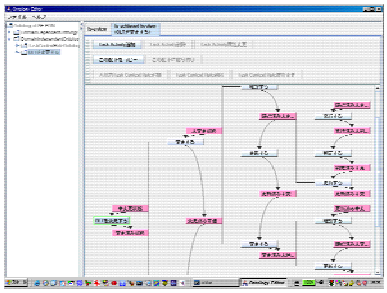


Figure 5 Ontology editor



### Further studies

For the sake of reusing models, there are some things to do along with the use of the previous example.

To reuse models of existing OSS developments using ontologies, we need to create a lot of ontologies. That means a tool for efficiently creating ontologies also needs to be developed, because there are no tools to support the task/domain ontologies that can represent the models of OSS developments. This tool should deal with OWL for cooperation with other ontology tools that have a validating function, etc. Figure 5 shows an example of a developing tool's display image. The example indicates the possibility of creating OWL files with a graphical user interface (GUI).

In addition, it is necessary to integrate this tool and the function to calculate the proposed similarity with IDE tools, and to support the whole development from the requirements phase to the implementation phase.

To show the effect of our approach, we should compare the models as determined from their calculated similarity with the original models.

As a final evaluation, we should check the effects in a practical setting. For that, we will have developers make models of a sample OSS either from scratch or by using a support tool that has knowledge of software models with ontologies, and then we will compare the times they take to make the models.

# Summary

[Model representation]

We can represent OSS models using a task/ domain ontology.

[How to create ontology]

We can mechanically create a task ontology from sequence diagrams by modified AFM.

[Reuse of models]

We can efficiently develop OSSs by reusing relevant models of existing OSS development.



**Summary**

In summary, we proposed the use of task and domain ontologies to represent the models used in the development of OSSs and proposed that a task ontology be created using a modified AFM. To ensure efficient development, we think that it's necessary to reuse relevant models in existing OSS developments based on an ontology. We are developing a tool-making ontology and will propose criteria for finding relevant models so that we can reuse the models.

**References**

[1] Masayoshi Ejiri, "Common Business Process and Process Components for IP/e Business Management," The journal of the institute of electronics, information and communication engineers, Vol. 87, No. 7, pp. 570-576, 2004.

[2] OMG, "OMG Unified Modeling Language Specification, Version 1.5," http://www.uml.org/.

[3] Ivar Jacobson, Grady Booch, and James Rumbaugh, The Unified Software Development Process, Addison Wesley Longman, Inc., 1999.

[4] Riichiro Mizoguchi, "Science of Intelligence Ontology Engineering," Ohmsha, Tokyo, 2005.

[5] Christiane Fellbaum, WordNet: An Electronic Lexical Database, Bradford Books, 1998.

[6] W3C, "OWL Web Ontology Language Reference," http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

[7] Takeshi Morita, Yoshihiro Shigeta, Naoki Sugiura, Naoki Fukuta, Noriaki Izumi, and Takahira Yamaguchi, "DODDLE-R: A Domain Ontology Development Environment for the Semantic Web," The 18th Annual Conference of the Japan Society for Artificial Intelligence, 2004.

[8] Shingo Horiuchi, Yasuhiro Suzuki, Takashi Inoue, Yasumi Matsuyuki, and Tetsuya Yamamura, "A study of application of ontology for OSS development," Proceedings of the 2004 IEICE Society Conference, 2004.

[9] Seiichi Ishikawa, Shigeki Kubo, Kouji Kozaki, Yoshinobu Kitamura, and Riichiro Mizoguchi, "Design and Development of a Guide System for Building an Ontology based on Task/Domain Role Concepts – A Case Study on an Oil Refinery Plant-," Transactions of the Japanese Society for Artificial Intelligence, Vol. 17, No. 5, pp. 585-597, 2002.

[10] Shingo Horiuchi, Takashi Inoue, Yasumi Matsuyuki, and Tetsuya Yamamura, "Approach of OSS design method using ontologies," Technical Report of IEICE TM2004-65, 2004.

[11] Shingo Horiuchi, Masateru Inoue, Takashi Inoue, Yasumi Matsuyuki, and Tetsuya Yamamura, "A study of how to reuse the deliverables of software development with ontology," Technical Report of JSAI, SIG-SWO-A404-04, 2005.

[12] Jin CUI, Eiji KOMATSU, Hiroshi YASUHARA, A CALCULATION OF SIMILARITY BETWEEN WORDS USING EDR ELECTRONIC DICTIONARY, IPSJ SIG Notes, SIGNL-93-1, 1993.