Composing and Supervising Language Services on the Internet

Masahiro Tanaka
Abstract

Services computing, which is a new paradigm for constructing software based on services, is now one of the hottest topics both in research community and in business. Services computing allows us to reduce the initial cost of software development and increase the reusability of the components.

Services computing can also realize the collective intelligence on the Web. We can coordinate language resources such as dictionaries, parallel texts and machine translators by wrapping them as Web services. We refer to the Web services which wrap language resources as language services. Coordinating the language services based on services computing technologies makes it easier to develop support tools for various fields of intercultural collaboration.

However, we still have a problem with coordinating the language services.

- There are a lot of dictionaries and parallel texts available on the Web. But we need to extract machine-readable contents from them to wrap them as language services because they are usually designed for humans. The extraction costs a lot because it requires us to manually edit the contents. This prevents people in the community of each field of intercultural collaboration from publishing their language resource as a language services.

- Language services which different organizations provide in open environment are not designed to be used together. This is be-
cause it is impossible for the service provider to expect how their services are used. The designer of a composite service which combines the language services cannot know the runtime environment because the behaviors of services frequently change in open environment. These features of services may cause failure of execution of composite service.

In this research, we aim at establishing the cycle of construction of language services and coordination of them by solving the above problems. Therefore we have tackled the following issues.

(1) Ontology extraction from tables
We have to recognize which part in a language resource is required for language service in order to extract machine-readable contents from the language resource. As for a language resource of parallel texts in English and Japanese, we need to know which part is an English expression and which part is the corresponding Japanese expression. On the basis of the semantics of the contents, the wrapper program returns the response message which the user requested.

In this work, we focused on extraction from language resources which have table structures because many language resources developed in the field of intercultural collaboration are represented as tables. The method we proposed tries to extract ontologies from the tables based on the semantics of the contents which table structures imply. Our method allows users to extract the contents in machine-readable format only by giving a small example of data in the table. This is easy even for the user which have no expertise on machine-readable format nor language services.

(2) Predicting connectability of Web services
The specifications of language services depend on the language resources wrapped. As we described before, an output of a Web service in a composite Web service may not be valid as the input of another Web service. The execution failure of any of Web service in a composite Web service leads to execution failure of the whole composite Web service.
If Web services which cost a lot are used in the composite Web service and are executed before the execution of the composite Web service stops, the cost for the execution of the Web services goes to waste.

Therefore we proposed a method which predicts whether all atomic Web services in a composite Web service can be executed for the given input based on the specifications of atomic Web services and structure of the composite Web service. We adopted a formal specification language and checks if the input to any atomic Web service is valid. Our method allows the user to describe the specification as far as known. Moreover, if the specification is not enough and execution fails, it learns specification from the result of the execution. We can reduce the cost of wasteful execution of atomic Web services by cancelling the execution of a composite Web service when our prediction that any of atomic Web service fails.

(3) Meta-level Control of Composite Web Services

The behavior of a Web service can unexpectedly change in open environment due to some features of Web services. The change of the behavior may cause execution failure of a composite Web service. However, generally speaking, the problems that can be triggered by the runtime environment are extremely diverse. This prevents the designer of a composite Web service from adding functions that can avoid the problems.

Our solution is the concept of Service Supervision. Running instances of a composite Web service are monitored and controlled from outside the composite Web service. Service Supervision makes the following points possible. First users can control behaviors of composite Web services by changing the execution state even if users are not authorized to modify the composite Web services in terms of protection of the intellectual rights. Next a control pattern for coordinating certain Web services can be applied to various composite Web services in order to reduce the load of designing control processes for coordination. To implement the Service Supervision, We introduced meta-level control of a composite Web service, which monitors and changes the execu-
tion state of a composite Web service. Then we used choreography described in WS-CDL in order to define protocols of interactions for coordination.
Acknowledgments

First of all, I would like to express my gratitude to my supervisor, Professor Toru Ishida at Kyoto University. It is almost impossible to overstate what I learned from his professional attitude. His supervision has without doubt changed my outlook on life.

I am also deeply grateful to members of my thesis committee, Professor Masatoshi Yoshikawa and Professor Katsumi Tanaka. They spent their time on the discussion with me despite their busy schedule and I got a lot of inspiration for my research.

Furthermore I am indebted to Associate Professor Shigeo Matsubara and Assistant Professor Hiromitsu Hattori, who always gave me kind advice and invaluable suggestions.

Discussions with my colleagues at our laboratory also enriched my life in these three years. In particular, I wish to thank Cho Heeryon, Yuu Nakajima and Lin Donghui, colleagues in the PhD programs. I believe in that their encouragement made this work possible.

Ms. Yoko Kubota and Ms. Terumi Kosugi always helped me on administrative affairs. Their support gave me the freedom from a lot of annoying paperwork.

I have greatly benefited from the experience of working with the people at NICT (National Institute of Information and Communication Technology). They impressed me by their hard-working and profound expertise.

Finally, I want to express my gratitude to my parents for their continuous support. It was very happy for me to have the place where
I can feel at peace.

This research was partially supported by Global COE Program “Informatics Education and Research Center for Knowledge-Circulating Society” and Japan Society for the Promotion of Science (JSPS), Grant-in-Aid for JSPS Fellows.
# Contents

1 Introduction ........................................ 1
  1.1 Objectives ........................................ 1
  1.2 Research Issues ................................... 5
  1.3 Thesis Outline ..................................... 9

2 Background .......................................... 11
  2.1 Services Computing ................................. 11
     2.1.1 Web service composition ....................... 12
     2.1.2 Controlling composite Web services .......... 14
  2.2 Collective Intelligence for Supporting Intercultural Collaboration .................................. 15
     2.2.1 Language resources on the Web ............... 16
     2.2.2 Language service ............................... 18

3 Ontology Extraction from Tables ....................... 21
  3.1 Introduction ....................................... 21
  3.2 Constructing Language Services from Language Resources on the Internet ...................... 24
  3.3 Generalizing Table Structures .................... 27
     3.3.1 Formalization of table structure ............ 29
     3.3.2 Procedure for extracting ontologies .......... 33
  3.4 Evaluation ......................................... 35
  3.5 Related Works ..................................... 41
  3.6 Summary ........................................... 45
## Predicting Connectability of Web Services

4.1 Introduction ................................................. 47  
4.2 Formal Specification to Model Web Services .......... 49  
4.3 Acquiring Input Specifications ............................. 59  
4.4 Experiment .................................................. 65  
4.5 Related Works .............................................. 70  
4.6 Summary ...................................................... 71  

## Meta-level Control of Composite Web Services

5.1 Introduction ................................................. 73  
5.2 Meta-level Control for Coordinating Web Services .... 76  
5.2.1 Coordinating Web services in open environment ... 77  
5.2.2 Meta-level control of composite Web services ... 80  
5.3 Coordination Control Based on Choreography ........... 83  
5.3.1 Choreography for Meta-level Control .............. 83  
5.3.2 Architecture ............................................. 86  
5.4 Discussion ................................................... 88  
5.5 Related Works .............................................. 92  
5.6 Summary ...................................................... 94  

## Conclusion

6.1 Contributions .............................................. 95  
6.2 Future Directions .......................................... 98  

Bibliography .................................................. 101  

Publications .................................................. 111
List of Figures

1.1 Relationship among research issues. 8

3.1 Multilingual living information provided by Council of Local Authorities for International Relations (CLAIR). 25
3.2 Language resource annotated by using RDF/OWL. 27
3.3 Model of adjacency of cells. 30
3.4 RDF graph representing relationship between words in a table. 31
3.5 Structure representing relationship between words in a table. 31
3.6 Symbols for modeling a pattern of a table structure. 31
3.7 Procedure for generating symbols. 32
3.8 Generalized structure for the table in Fig. 3.1. 33
3.9 Procedure for extracting relationships from a table. 36
3.10 Classes of tables. 39

4.1 Definition of request message. 51
4.2 Specification of a machine translator service. 51
4.3 Sequence control construct and dataflows. 54
4.4 Specification of the block in Fig. 4.3. 55
4.5 Formal specification of If-Then-Else. 56
4.6 Formal specification of Split+Join. 57
4.7 Formal specification of Choice. 58
4.8 Formal specification of While. 59
4.9 Version space and partial order for hypotheses. 61
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.10</td>
<td>An updated axiom.</td>
<td>63</td>
</tr>
<tr>
<td>4.11</td>
<td>The process of predicting connectability and learning input</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>specifications.</td>
<td></td>
</tr>
<tr>
<td>4.12</td>
<td>The architecture for predicting and learning connectability.</td>
<td>65</td>
</tr>
<tr>
<td>4.13</td>
<td>Composite service for translation in a special domain.</td>
<td>66</td>
</tr>
<tr>
<td>4.14</td>
<td>Number of wasteful execution of atomic/composite services</td>
<td>69</td>
</tr>
<tr>
<td>5.1</td>
<td>Web services in open environment</td>
<td>76</td>
</tr>
<tr>
<td>5.2</td>
<td>Composite Web service for translating documents.</td>
<td>78</td>
</tr>
<tr>
<td>5.3</td>
<td>Coordination by modifying a composite Web service.</td>
<td>80</td>
</tr>
<tr>
<td>5.4</td>
<td>Supervision composite Web service.</td>
<td>82</td>
</tr>
<tr>
<td>5.5</td>
<td>Choreography for controlling interactions.</td>
<td>85</td>
</tr>
<tr>
<td>5.6</td>
<td>The architecture for realizing Service Supervision.</td>
<td>86</td>
</tr>
<tr>
<td>5.7</td>
<td>Comparison of execution time.</td>
<td>90</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Applicability to language resources. ............................. 37
3.2 Precision and recall .................................................. 37
3.3 Number of extracted RDF statements. .................. 40
4.1 Process of acquiring input constraints. ..................... 63
5.1 Complexity of composite Web services. ................. 91
Chapter 1

Introduction

1.1 Objectives

Services computing, which is a new paradigm for constructing software based on services, is now one of the hottest topics both in research community and in business. Services computing allows us to reduce the initial cost of software development and increase the reusability of the components. In services computing, application software consists of Web services which are deployed on remote servers and can be invoked via internet instead of legacy software components.

Services computing based on Web services have been mainly used within one company or among companies that have common goals in order to make their business processes more effective. However, programs and contents of various organizations have recently become available as Web services in open environment. Some of the Web services provide functionalities including knowledge processing or knowledge contents. Therefore we can realize the collective intelligence on the Web by coordinating the Web services.

In the Language Grid project, they have collected various language resources such as dictionaries, parallel texts and machine translators from various organizations and wrapped them as Web services. We refer to the services which wraps such language resources as language
services. The goal of the Language Grid is to support activities in the field of intercultural collaboration by realizing collective intelligence based on services computing technologies.

Generally speaking, however, language resources developed by different organizations do not have the common interfaces. This is the reason it is difficult to coordinate the language resources. Therefore they introduced the standard protocol and definition of interface in services computing such as SOAP and WSDL. This allows us to access the language services on remote servers by exchanging XML messages in the standardized way. They also standardized the interface of the language services based on the types of them. This makes it easy to use the language services.

In the field of intercultural collaboration, support tools customized for each field are required. Service computing technologies and language services based on them can help us to achieve the requirements for the tools.

For example, we can produce a machine translator Web service which is specialized in a special domain by coordinating a general machine translator Web service and a technical term dictionary for the domain. Machine translation is frequently used in the field of intercultural collaboration. We can provide a customized machine translation embedded in a tool only by selecting a technical term dictionary which is appropriate for each field.

The goal of our research shown in this thesis is to give energy to activities in the field of intercultural collaboration based on services computing technologies by establishing a cycle of new language service construction and use of them.

From the viewpoint of linguistic support, the feature of the field of intercultural collaboration is the uniqueness of the terms and expressions used in the field. This often leads to decline of the translation quality by machine translators. This is the reason we need to coordinate a general machine translator Web service and a Web service which wraps the technical term dictionary or parallel texts for the domain.
The technical term dictionary or parallel texts are often developed by the community which works on each field of intercultural collaboration. But we still have a problem with the cost for constructing language services from the language resources.

In the Language Grid project, they have already published many language services from dictionaries and parallel texts. As for the language resources available as Web pages in HTML or PDF, they had to extract the required contents and transformed them into the format for wrapper programs of language services. The components in the wrapper program which are responsible for message exchange via internet are common in any language services. This is the reason why what we have to do for constructing dictionary or parallel texts service is only to extract the contents for the wrapper program. But the extraction often costs a lot. Extraction using regular expression was useful for the work, but it still costs a lot due to the irregular descriptions and complicated format. We have to let each community in the field of intercultural communication publish their language resources as Web service by themselves in order to realize collaborative intelligence. However, a community in the field of intercultural communication does not usually have staffs enough to do the extraction.

In this research, we first proposed a method for extracting contents in machine-readable format from dictionaries or parallel texts on the Web. This method allows users to construct their own language services at a low cost.

Once domain specific language services such as technical term dictionaries are provided from fields of intercultural collaboration, the language services are used with other Web services. Some Web services, such as a machine translator Web service, wrap the contents or programs which are developed by commercial companies.

Such programs or contents sometimes limit the traffic or cost some money based on the contracts between users and providers because the development of the resource costs a lot. EuroWordNet, which is a WordNet designed for European languages, was wrapped and the
language service for the resource is available on the Language Grid. But this language service limits the traffic during a certain period of time.

This may cause the problem with the execution of a composite Web service, which consists of these language services. Language services which wrap the language resources collected from various organizations have different specification. Moreover, the developer of the language services does not know what service is used with his service. Therefore a Web service may fail when the output of other Web service is given as an input because the input is not valid for the Web service. This leads to the failure of the execution of the composite Web service. If Web services which costs a lot are used in the composite Web service and are executed before the execution of the composite Web service stops, the cost for the execution of the Web services goes to waste.

Coordination between language services which are constructed from language resources of small communities in the field of intercutural collaboration and language services provided by companies increase the values of both services. Therefore we proposed a method for predicting connectability between Web services to prevent the wasteful execution in a composite Web service.

We need to consider another characteristic to appropriately execute a composite Web service instead of legacy software components.

Web services are usually deployed on the server which the providers are responsible for. This is the reason the provider can stop or update his Web service anytime he wants. This may cause unexpected changes of the behavior of Web services. Moreover, a Web service is shared by many users. Therefore the throughput of the Web service can decline due to many requests to the Web service. New Web services can be also added to the environment after a composite Web service is designed.

We have to adapt a composite Web service to such changes of the environment. For example, we can improve the QoS of a composite Web service by changing Web services used in the composite Web service when a new better Web service is released.
Many previous works have proposed methods for calculating QoS or checking the SLA (service level agreement) at runtime. But it is difficult for the existing frameworks for executing a composite Web to implement such methods. This requires us to modify a composite Web service itself to implement the methods. However, the processes for the method are not related to business logic which the composite Web service should realize and the reusability of the composite Web service decline by implementing the methods. Moreover in open environment, the user of the composite Web service is not allowed to modify the composite Web service because the user belongs to a different organization with that of the designer of the composite Web service.

In order to solve these problems, we aim at developing a framework which controls execution of a composite Web service at meta-level.

1.2 Research Issues

In this thesis, we tackled the following research issues in order to achieve the goal described in the previous section.

Ontology extraction from tables As we described in the previous section, we need machine-readable contents to input into the database of the wrapper program in order to construct language services which can be invoked via the internet. Automatic approaches on information extraction can be a help for reducing the cost of the extraction of machine-readable contents from language resources. The extracted data has to clearly show the semantics of the contents in the language resources. As for a language resource of parallel texts in English and Japanese, we need to know which part is an English expression and which part is the corresponding Japanese expression. On the basis of the semantics of the contents, the wrapper program returns the response message which the user requested.

In this work, we focused on extraction from language resources
which have table structures because many language resources developed in the field of intercultural collaboration are represented as tables. The method we proposed tries to extract ontologies from the tables based on the semantics of the contents which table structures imply. Our method allows users to extract the contents in machine-readable format only by giving a small example of data described in the table. This is easy even for the user which has no expertise on machine-readable format nor language services.

Predicting connectability of Web services The specifications of language services depend on the language resources wrapped. As we described before, an output of a Web service in a composite Web service may not be valid as the input of another Web service. The execution failure of any of Web service in a composite Web service leads to execution failure of the whole composite Web service. In open environment, however, the provider of atomic Web service, the designer of a composite Web service and users often belong to different organizations. This is the reason why the user cannot modify the atomic Web services or the composite Web services even when it fails for a specific input. Moreover, the methods for validation of composite Web service which have been proposed in previous works are not useful because the specification required for the validation are often unavailable in open environment.

Therefore we proposed a method which predicts whether all atomic Web services in a composite Web service can be executed for the given input based on the specifications of atomic Web services and structure of the composite Web service. We adopted a formal specification language and checks if the input to any atomic Web service is valid. Our method allows the user to describe the specification as far as known. Moreover, if the specification is not enough and execution fails, it learns specifi-
cation from the result of the execution. We can reduce the cost of wasteful execution of atomic Web services by cancelling the execution of a composite Web service if we predict that any of atomic Web service fails.

**Meta-level Control of composite Web service** The behavior of a Web service can unexpectedly change in open environment due to some characteristics of Web services. The change of the behavior may cause execution failure of a composite Web service. However, generally speaking, the problems that can be triggered by the runtime environment are extremely diverse. This prevents the designer of a composite Web service from adding functions that can avoid the problems. On the other hand, the users of a composite Web service may know details of the runtime environment. But they also have difficulty in coordinating Web services to suit the runtime environment. Our solution is the concept of Service Supervision. Running instances of a composite Web service are monitored and controlled from outside the composite Web service. This does not demand modification of the original composite Web service. Service Supervision makes the following points possible. First users can control behaviors of composite Web services by changing the execution state even if users are not authorized to modify the composite Web services in terms of protection of the intellectual rights. Next a control pattern for coordinating certain Web services can be applied to various composite Web services in order to reduce the load of designing control processes for coordination. Moreover users can simulate execution of a composite Web service to inspect its behaviors in the runtime environment without paying the cost for invoking Web services. In this thesis, we mainly focused on the first two points and combined the following approaches in order to realize Service Supervision. We introduced meta-level control of a composite Web service, which monitors and changes the execution
state of a composite Web service. Then we used choreography described in WS-CDL in order to define protocols of interactions for coordination.

The relationship among these three research issues are illustrated in Fig. 1.1.

![Figure 1.1: Relationship among research issues.](image)

In each field of intercultural collaboration, they have their own language resource for a specific domain. The users in the field construct a language service from their language resource by the method for extracting ontology from tables. They can obtain the contents of the language resource in machine-readable format using the method. The constructed new language service is registered to the service repository.

Composite Web services are deployed on the composite Web service execution engine to combine atomic Web services. The execution engine interprets and executes a composite Web service when it is given a request from a client. The composite Web service invokes Web services registered to the service repository.

The users in the field of intercultural collaboration send a request to the execution engine via some tools. The tool can select atomic Web
services bound to the invoked composite Web service according to the requirements of each field.

However, the execution of the composite Web service may fail due to the conditions of the runtime environment. Therefore we introduce the execution control engine for composite Web services which monitors/controls execution state of the running instance of the composite Web service. The execution control engine realizes runtime adaptation such as switching services if a Web service is temporary unavailable.

As one of the runtime control components based on the execution control mechanism, we also introduce the prediction of connectability of Web services. This component collects information of a running instance of a composite Web service and cancels the execution if it obviously fails via execution control engine.

1.3 Thesis Outline

The rest of this thesis is organized as follows. In Chapter 2, we will explain services computing technologies, and overview language resources on the Web and the way of wrapping and coordinating the language resources as the background of our research. Next we propose the method for extracting ontology from tables on the Web in Chapter 3. The method allows users to easily extract machine-readable contents for language service construction. Then we describe the work which aims at prediction of connectability of Web services in Chapter 4. In Chapter 5, we show the concept of Service Supervision and the implementation which realizes execution control of a composite Web service. Finally we conclude this thesis and mention future works in Chapter 6.
Chapter 2

Background

In this chapter, we first overview services computing, which is a new paradigm for software development, and the related technologies as the background of our research. Then we describe the support for intercultural collaboration based on collective intelligence using services computing technologies as the motivation of our research.

2.1 Services Computing

Services computing is a new paradigm for constructing software based on services and now one of the hottest topic both in research community and in business. Services computing allows us to reduce the initial cost of software development and increase the reusability of the components.

Although the area of services computing just started, it is expected to be the breakthrough of information technologies. Therefore many research groups and enterprises are working in this area and new technologies and businesses are emerging day by day. In Japan, unfortunately, there are few research groups working in this area and researches in our country have fallen behind the international trend. The works shown in this thesis are based on the Language Grid, an infrastructure of Web service, which is the result from one of the major research
groups in services computing.

We refer to services which are deployed on remote servers and invoked via the internet as Web service. The meaning of “Web service” depends on the context, but we refer to ones which provide remote procedure call based on exchange of XML messages on the internet. Such a “Web service” is sometimes called an “XML Web service”.

In this section, we describe technologies related to our research from the viewpoint of services computing.

2.1.1 Web service composition

Web services provide the way of remote procedure call (RPC) based on XML message exchange. There are some standards in order to keep the interoperability of RPC, such as SOAP (Simple Object Access Protocol) [soa 00] and WSDL (Web Service Definition Language) [wsd 01]. SOAP is a protocol for message exchange among Web services and their clients. It defines the XML format which describes RPC and the structures of its parameters. On the other hand, WSDL is a language for definition of interfaces of Web services. It describes operations which the Web service provides, data types, and encodings for exchanging messages on the internet.

Web services mainly focus on invocation from programs, not directly from human. Thus WSDL is machine-readable but Web services usually do not provide the interface for humans. We can effectively configure required functions of applications by coordinating Web services which have various functions.

There are two prominent ways of creating a new Web service by combining multiple Web services (composite Web service). The first way is to invoke Web services as a function call in programming languages. In many major programming languages, the libraries which deal with SOAP and WSDL are available. Another way is to define a business process which has Web service invocations as tasks. The granularity of process of Web services is rather coarse. This is because
tasks in business process which are different departments are expected to be replaced with Web services. In this thesis, we refer to the latter as “composite Web service”.

A composite Web service defined as a business process is described in a language for Web service processes, such as WS-BPEL[1].

Many previous works have tried to automatically generate a composite Web service based on the given task and constraints of available Web services. They have adopted AI planning techniques, logic, process algebra, model checking and so on. However, we do not still have a practical system which can generate a composite Web service. Thus a human designer who has the expertise on services computing technologies usually produces composite Web services.

On the other hand, reuse of a composite Web service is expected to be more practical. This is because we often have Web services which are functionally equivalent but have non-functional properties such as quality of results or response time. Once a composite Web service is given, we can select the functionally equivalent Web services which satisfy the requirements of our application by setting the endpoint for the Web service.

The designer of a composite Web service provides his/her composite Web service in WS-BPEL[1] through the combination of abstract atomic Web services, for which only the interfaces, and not the endpoints, are defined. We refer to such a composite Web service as an abstract composite Web service. Standardizing the interfaces of abstract atomic Web services makes it easy for an application developer to realize the functions of the abstract composite Web service desired. The application developer simply sets endpoints for the abstract atomic Web services forming the abstract composite service; this identifies the concrete atomic Web services that will be actually invoked. We refer to such an implemented composite Web service as a concrete composite Web service.

Hassine et al. referred to this style of Web service composition as horizontal Web service composition[13]. The authors labeled
another style of Web service composition as vertical Web service composition. Vertical composition finds the best combination of the abstract Web services. Most of works on Web service composition have focused on vertical composition [McIlraith 02, Sirin 04] and they automatically generate Web service workflows using AI planning. On the other hand, horizontal composition aims at finding the best combination of concrete Web services which have the same functional properties and different non-functional properties. Horizontal Web service composition will become more and more important as the number of Web services increases and standardization of Web service interfaces is achieved.

2.1.2 Controlling composite Web services

Composite Web services have been mainly established within companies or among companies that have common goals. The objective is to improve the efficiency of their business by implementing existing business processes as composite Web services. In such a circumstance, the manager and users know what Web services are available and how they are used.

However, Web services for unspecified number of users have recently become available in open environment. Even the manager of the Web services cannot know how and when their Web services are used beforehand in the environment. Moreover, service providers in various organizations can join and publish their service in the open environment. This is the reason the number of available services in the environment increases day by day.

Web services which different organizations provided are not designed to be used together. This may cause runtime failure which is not expected by the designer of a composite Web service. For example, an output of a Web service in a composite Web service is often given to another Web service as an input. But the given data is not always valid for the latter Web service.

Even if the same input is given to the same Web service, the Web
service may differently work. This is because a Web service is always controlled by its provider. And the provider may stop his Web service when he monitors the too many accesses to the Web service. In this case, any user, not only the user who gives too many accesses, cannot use the Web service any more.

If any Web service in a composite Web service fails due to the reasons, the execution of the composite Web service also fails and expected result is not given.

However, modifying composite Web service makes the designer’s work more complicated because it is difficult for the designer to expect the bound concrete Web services and the runtime environment.

Another solution is to control the behavior of a composite Web service. Therefore some previous works have tries to add processes without modifying a composite Web services. AO4BPEL[Charfi 07] introduced Aspect-Oriented Programming (AOP) into BPEL. It allows the composite service designer to insert any process before/after an activity specified as a pointcut. Dynamo[Baresi 07] is also based on the concept of AOP. It monitors exchanges of messages between a BPEL process and external Web services and checks if the messages satisfy constraints.

But these works realizes only adding processes to an existing composite Web service and they are not enough to realize various controls required in open environment, such as switching concrete Web services and monitoring/changing the state of the running instance of the composite Web service.

### 2.2 Collective Intelligence for Supporting Intercultural Collaboration

The Language Grid project aims at realizing collective intelligence by combining language resources on the Web based on services computing technologies described in the previous section. In this section,
we explain some important elements for the construction of collective intelligence based on services computing technologies.

2.2.1 Language resources on the Web

Recently various language processing programs and contents which are useful for multilingual communication such as machine translators, dictionaries and parallel texts have become available. We refer to such programs and contents as language resources. Scale, usage and conditions of use depend on each resource. We overview the current state of language resources on the Web.

General dictionaries which are not specific to any domain usually have a lot of entries. The contents are stored in the database at back-end and the user interfaces are provided on the Web site. For example, Merriam-Webster Online* belongs to this type of language resources. Such a language resource is originally compiled by experts and published as paper dictionary. This is the reason why construction of such a language resources costs a lot. Web site which provides the interface for such a language resource aims at obtaining profit by advertisement.

On the other hand, there is another type of language resources which provide translations of technical terms and sentences peculiar to a special domain. The scale of such language resources is usually much smaller than a general dictionary. A paper dictionary often has several tens of thousands of entries, but this type of language resources has from a few dozens to the thousands translations of terms or sentences. Such language resources are often constructed in each field of intercultural collaboration in order to support the activities in the field. This is the reason why the author of the language resources are usually a department in a local municipality, an NPO or an individual, but not commercial companies.

The language resources which belong to this type are provided in form of HTML or PDF on the Web. Therefore they do not provide user

*http://www.m-w.com/
interface for the search via a Web browser. This is because they do not have enough staffs and budget for development of Web application which has a graphical user interface and a database.

A language processing program such as a machine translator provides an interface on the Web to access the program. Development of a machine translator costs a lot as well as large-scale general dictionaries. Therefore machine translators are developed by the company which has experts and language processing technologies. The Web site which provides machine translators on the Web usually purchases the license from such companies and provides it as a part of the contents of the Web site for advertisement.

We also consider a program which is more specialized to natural language processing like a morphological analyzer as a language resource. Such a program is developed mainly by universities and research institutes. Some institutes provide the interface to access their program on the Web in order to show the performance. But most of them are designed to be executed on local machines because program specialized in natural language processing is difficult to use as stand-alone application.

The language resources have been independently developed by various organizations. This is the reason why the schema of the data which the language resources handle and the interface to access the language resources are different with each other.

Moreover, language resources which have interfaces to access them have different features from legacy software components. They are provided on the server of providers and unspecified number of users can access them at the same time. This leads to changes of the behavior of the resources. For example, Merriam-Webster Online limits the access when too many requests are given in a certain period of time and become temporarily unavailable.
2.2.2 Language service

We can make it easy to develop an application which supports inter-cultural collaboration by combining various language resources if the language resources are invoked from our application. However, language resources on the Web are provided on different remote servers. Thus we need to access the language resources via the internet. Another problem is that most of language resources on the Web are designed to be used on a Web browser by humans. Therefore they do not provide APIs as software components.

Therefore we have wrapped language resources and publish them as Web services based on standard technologies of services computing. The Web services have interfaces which access language resources such as machine translators and dictionaries and the interfaces are described in WSDL. We can invoke the Web services via the standard protocol SOAP. This allows us to combine multiple language resources. We refer to such a Web service which can access the language resource as a language service.

The Language Grid project have collected language resources from various organizations and wrapped them as language services. Available service types are dictionaries, parallel texts, machine translators, morphological analyzers and so on. The interfaces of the services are standardized according to the types. This standardization leads to reuse of composite Web services. Assume we have a composite Web service which combines a machine translator Web service and a technical term dictionary to improve the translation quality in a special domain. When the interface of the dictionary Web service is standardized, we can easily change the dictionary Web service to another dictionary Web service in a different domain only by changing endpoint of the Web services.

If a language resource has the interface on the Web site, we can wrap the language resource by transforming a request from a client following SOAP into a request for the interface. As we described before,
however, dictionaries and parallel texts which have been accumulated in the field of intercultural collaboration are usually provided as Web pages in HTML or PDF. We need to manually extract required contents of the language resources and input them into the database of the wrapper program. But the extraction of the contents costs a lot.

To allow small communities to publish their own language resources as language services is crucial to realize collective intelligence for supporting intercultural collaboration. Therefore we need to reduce the cost of constructing language services from existing language resources.

Automatic or semi-automatic approach on information extraction may be a help for reducing the cost. There have been many previous works on information extraction from Web page such as [Ashish 97, Kushmerick 97]. However, in order to apply the previous works to language service construction, we still have a problem due to diversity of the structures and the contents of language resources on the Web.
Chapter 3
Ontology Extraction from Tables

3.1 Introduction

There are various language resources on the Web. We can classify them into two categories; programs such as machine translators and static contents such as dictionaries and parallel texts. The programs are usually products of commercial companies. Research organizations also often provide natural language processing programs such as morphological analyzers. But not many companies or research organizations have produced such programs because advanced expertise is required to develop the programs. On the other hand, static contents are easy to create for non-expert users. This is the reason why there are many dictionaries and parallel texts which are specialized in a specific domain.

We can create a new functionality by coordinating these language resources if they are wrapped as Web services. For example, we can coordinate a machine translator Web service and a technical term dictionary Web service based on services computing technologies. This improves the quality of translation of sentences in a special domain.

The combination of a machine translator Web service and a tech-
Technical term dictionary is quite effective in fields of intercultural collaboration. This allows us to realize such a customized translation service only by changing invoked technical term dictionaries.

However, we still have a problem with realizing the combination. The language resources such as technical term dictionaries or parallel texts developed in the field of intercultural collaboration are not usually machine-readable. Most of the language resources provide from several hundred to thousands of terms or sentences on the Web pages and no interface for searching is provided.

The Web pages are directly created by humans, not generated from database at backend. The authors of the language resources do not have enough staff or budget for developing and maintaining the database because they are often a department in a local municipality, an NPO or an individual, not commercial company. The language resources in communities of intercultural communication and data provided by enterprises such as price lists differ in this point.

Constructing a language service from a language resource requires the wrapper program which returns the contents in the language resource according to a user’s request. But the language resource is not readable for the wrapper program because the Web page of the language resource is designed for humans. Therefore we need to transform the language resources in the Web page into machine-readable format and input them into the database of the wrapper program.

The scale of the language resources developed for the specific domain are relatively small, but such language resources are released by many organizations or communities. Therefore it is impossible to construct language services from them by the centralized approach. But it is also difficult for people in each community to make machine-readable data due to the lack of staff and budget.

In this work, we aim at construction of language services by extracting machine readable contents from static language resources such as dictionaries and parallel text.

We focus on dictionaries and parallel texts in table structures avail-
able on the internet because the language resources are represented as tables. We extract ontology which describes relation between contents in the resource based on the semantics of table structures. Once the ontology is acquired, we can construct a language service only by inputting the ontology to the database of the wrapper program. Moreover, ontology based on shared vocabulary is needed for integrating a language resource with other resources.

Some previous works have tried to extract information from tables. In [Chen 00, Wang 00], they interpret tables according to predefined typical table structures. The recognition model for tables[Pivk 04] or domain-specific database[Embley 02, Tijerino 03] is also used to analyze tables.

But we cannot adopt the previous works for extracting ontology from language resources due the following reasons.

**Ambiguous semantics of structure** The relation which a table structure represents depends on tables. This is the reason it is difficult to extract the detailed information based on a prior model of the structure.

**Interpretation according to contexts** Our goal is to extract ontology from language resources in various domains. This make it difficult to analyze tables based on domain-specific database.

Our solution to these problems consists of three steps as follows:

**Interpretation of table structures by humans** First the user interprets the table structures and gives an example which shows the relation between contents in the table.

**Generalize the table structure** Next the table structure which is given an interpretation by humans is generalized based on some features of table structure, such as adjacency of cells and iterative structure.
**Interpretation of tables** The given table is finally interpreted by generalized structure and the semantics of contents are extracted as ontology.

The rest of this chapter is organized as follows. In 3.2, we first describe the construction of language services from language resources. Next we show the observation of table structure of language resources on the internet, propose a formalization for the tables structures and a method for extracting ontology based on the formalism in 3.3. The evaluation of the result is discussed in 3.4. After we introduce some related works in 3.5, we conclude this work in 3.6.

## 3.2 Constructing Language Services from Language Resources on the Internet

As we explained in the previous section, language resources which have been accumulated in the field of intercultural collaboration are often provided in HTML or Excel. This is because the formats are easy for humans to handle.

Take Fig. 3.1 as an example of a language resource developed in the field of intercultural collaboration. This language resource is developed by Council of Local Authorities for International Relations (CLAIR) in order to support foreigner’s lives and available on the Web site. Parallel texts in various languages including English, Chinese, Korean, Portuguese, Spanish and Tagalog are classified based on situations.

To wrap this language resource as a Web service, we first need to define the interface. As for language services on the Language Grid, they standardized the interfaces according to the types of the language services, such as dictionaries, parallel texts and machine translators and so on. The interface for the parallel texts service takes a source language, a target language, a word to be searched for, and a search method (complete match, partial match, regular expression etc.).
Figure 3.1: Multilingual living information provided by Council of Local Authorities for International Relations (CLAIR).

In case of the language resource in Fig. 3.1, we may need to extend the interface for parallel text service because this language resource has some additional information such as categories of parallel texts and situations in which they are used. The required information depends on the application. If an application which shows parallel texts with categories integrates this language resource by invoking the language service, we need the extension.

The wrapper program for language resources are deployed on a server for Web service. The wrapper program returns the contents of the language resource according to the given request from a client by accessing the database at backend.

Therefore we need to design the schema of contents in the database according to language resources. Then we input contents into the database selecting and copying required information in the language resources.

But the most work for the input is done by humans. This is because language resources on the Web are often directly edited as Excel or HTML by humans without using regularized database. Therefore even the author does not have the contents in the machine-readable format.

In the Language Grid project, they have wrapped such language resources as language services. In order to do this, they had to extract the required contents and input them into the database of the language ser-
vices. Therefore they tried to reduce the cost of extracting the contents by regular expression, but the structures are often too complicated or not regular enough because they are directly edited by humans. This leads to manual extraction from large scale language resources, which costs a lot.

Another problem is that the usage of contents in language resources depends on contexts. This is the reason why previous researches on automatic information extraction do not worked effectively.

For example, most of the previous works on information extraction from tables have tried to pairs of attributes and the corresponding values. In case of the language resource shown in Fig. 3.1, the name of languages are value and the expressions in the languages are the corresponding attributes. The categories are also attributes, but the expressions in the two languages are the corresponding values.

However, the extracted pairs are not enough to construct language services. The wrapper program for the language resource returns the parallel texts which the user specifies based on languages or keyword. Thus it has to be able to recognize which part is a language, a parallel text or a category.

So we describe relations between contents in the language resources in RDF/OWL. If we can define the semantics of each part in the language resource, we can show which part is what the user requested.

Figure 3.2 shows the relations between words described in Fig. 3.1 using RDF. pt is the namespace for contents of parallel texts. translations is a tuple of sentences which have the same meaning in different languages. translation is an expression which is an element of translations. category is a category which a parallel text belongs to.

Once we have the semantic information as RDF graph shown in Fig. 3.2, it is clear to the wrapper program which part should be returned to the user according to his request. This allows us to construct language services only by giving these information to the wrapper program when the interface of the language service is standardized and components of
SOAP message exchange are established.

3.3 Generalizing Table Structures

In this section, we show our formalization of structure of tables on the Web. Our goal is to generalize table structures for extracting ontologies based on the formalization. First we explain our observation about features of table structures and relations which the structures represent.

We focus on tables whose shape is a rectangle and whose cells inside are also rectangles. We also handle the cells which span over some rows or columns. This class of tables is the same as those which we can produce by using table and related tags in HTML or a spread sheet application like Microsoft Office Excel. This class of tables are often used for language resources because it is easy for the user in the field of intercultural collaboration to directly edit the table.

Table structures represent relations between data in the table. We refer to class-instance relation, various property-property value relation in ontology as the “relation” in this thesis. It is reasonable to suppose that a relation represented by a structure is consistent in one table. We call the relations represented by a table structure the semantics of the table structure.

We take the language resource in Fig. 3.1 as an example. The structure of the language resource is a table we defined above. In the
first row, the languages of parallel texts and possible situations of the parallel texts as attributes.

We first consider that we represent the contents as ontology described in RDF. In this table, a record which describes parallel texts in a row corresponds with an instance of the class of parallel texts. The expressions in Japanese and English are property values of property which the instance has. The names of the languages, “Japanese” and “English, can be considered to be property values of the property which shows the language of parallel texts”. IDs for parallel texts can be ignored when our goal is to integrate this language resource into other tools.

We can explain the semantics of the structure of the table as,

- The cells spanning over multiple rows represents a class of instances of parallel texts.
- The cells which follow the class in left hand represent an instance of parallel texts.
- The region which represents an instance of parallel texts can be divided to cells which represent properties of the instance.

If we know the semantics of the table structure as shown above, we can extract relations between contents in the table. But we need general semantics as much as possible in order to extract relations between lots of contents in the table.

Therefore, once we can know the above semantics of the table by generalizing the correspondence between the semantics of “some” contents and the structure, we can extract relations between all contents in the table.

In order to consider how we can generalize the structure and its semantics, we assume features of table structures based on our observation of table structures.

**Adjacency of cells** A table often has cells which span over multiple rows or columns as the language resource in Fig. 3.1 does. We
assume the two cells which are adjacent and whose edges met have
different length contain different types of contents. For example,
in the language resource in Fig. 3.1, the cells in the second column
span over more than one rows. On the other hand, cells in the
right hand of the column do not. In this case, the cells in the
second column represent classes of parallel texts and cells which
follow them in the right hand represent instances of parallel texts.

Iterative structure We assume that the cells (or blocks which consist
of cells) contain the same type of contents when they have the
same features and iteratively appear. In the language resource in
Fig. 3.1, cells in the right hand of the second column, but all of
them are properties of instances of parallel text.

We formalize table structures which represent relations between con-
tents in tables based on the above assumptions.

3.3.1 Formalization of table structure

We formalize table structures focusing on features of adjacency of
cells based on our assumption. Then we defined a grammar which
represents a pattern of occurrence of cells for our formalization.

We introduce a “box” as an element corresponding to a cell, and
adjacency of cells are represented as links between boxes. We also
introduce a uni-directional link and a bi-directional link between boxes.
A link between boxes represents the adjacency of two cells as is shown
in Fig. 3.3.

The three types of adjacency of two cells are shown in the upper
part of Fig. 3.3, and two boxes corresponding to the cells and the
link between them are shown in the lower part of Fig. 3.3. “Box A”
corresponds to “Cell A” and “Box B” corresponds to “Cell B”. The link
between “Box A” and “Box B” represents the adjacency between “Cell
A” and “Cell B”. When the edges of two adjacent cells meet completely,
the two boxes corresponding to the cells are linked bi-directionally as
shown in Fig. 3.3(a). When the longer edge contains the shorter edge, the two boxes corresponding to the cells are linked uni-directionally as shown in Fig. 3.3(b). When the two edges meet partially, two boxes are not linked as shown as shown in Fig. 3.3(c).

In table structure, which cell is upper than (or left to) another one is important feature. Therefore we define four types of links (up, down, left, right).

Assume the some of contents which appear in the language resources shown in Fig. 3.1 is given as the RDF graph in Fig. 3.4. We can represent the table structure of the language resource as shown in Fig. 3.5. In the figure, we labeled boxes which are given the semantics according to the RDF graph in Fig. 3.4.

Then we define a grammar which represents a pattern of occurrence of cells based on the above formalism. First we consider boxes which are constituents of a row or a column as one symbol. We also consider the links between the rows or columns as a symbol. When we interpret the structure in Fig. 3.5 in vertical orientation, the symbols appear in the structure are $b_1, e_1, b_2$ shown in Fig 3.6. $b_1$ and $b_1$ can be divided by interpreting them as a horizontal sequence of boxes and links. For example, we can divide $b_2$ into symbols $b_{21}, b_{22}, e_{21}, e_{22}, e_{23}$. 

Figure 3.3: Model of adjacency of cells.
Figure 3.4: RDF graph representing relationship between words in a table.

Figure 3.5: Structure representing relationship between words in a table.

Figure 3.6: Symbols for modeling a pattern of a table structure.
GenerateSymbol(table)
  \( r \leftarrow 1, i \leftarrow 1, j \leftarrow 1 \)
  repeat
    \( h \leftarrow \text{split}(r, table) \)
    if box-symbol(r, h, table) \( \notin \{b_m\} \)
      \( b_i \leftarrow \text{box-symbol}(r, h, table), i \leftarrow i + 1 \)
    if \( r + h \)th row (column) is the tail of table
      return \( \{b_m\}, \{e_n\} \)
    if edge-symbol(r, table) \( \notin \{e_n\} \)
      \( e_j \leftarrow \text{edge-symbol}(r, table), j \leftarrow j + 1 \)
    \( r \leftarrow r + h + 1 \)

Figure 3.7: Procedure for generating symbols.

Figure 3.7 shows the procedure for generating these symbols from tables. \( \text{split}(r, table) \) is a subroutine and returns the minimum number \( h(\geq 0) \) with which there is no spanning cell over \( r + h \)th row (column) and \( r + h + 1 \)th row (column). box-symbol\((r, h, table)\) returns boxes and links which represent rows or columns between \( r \)th and \( r + h \)th. edge-symbol\((r, table)\) returns links between \( r \)th row and \( r + 1 \)th column. \( \{b_m\} \) and \( \{e_n\} \) are sets of symbols which corresponds to boxes and links respectively. The orientation of the table is determined by the method proposed in [Chen 00]. When a symbol contains multiple boxes, this procedure is recursively applied to the symbol.

The structure shown in Fig. 3.1 is transformed into a sequence \( b_1e_1b_2e_1b_2e_1b_2... \) by applying this procedure.

Then we show the procedure for generalization of table structure based on the sequence. As we described above, we assume cells which have the same feature and iteratively appear contain the same type of contents.

According to this assumption, we fold symbols of boxes and links which have the same feature into another symbols. Thus we introduce a notation “+” to represent structures that contain iterative boxes that have the same links.
We can revise the structure shown in Fig. 3.1 as Fig. 3.8 by introducing the “+”. The boxes and links in brackets with “+” can iteratively occur. The links besides “+” represents the links between iterative boxes in the brackets.

The boxes differently labeled contain different types of contents. This is the reason why we fold boxes only if they have the same labeled or one of them is not labeled. We call the pattern of table structure a generalized structure.

The generalized structure shown in Fig. 3.8 is represented by a grammar for the defined symbols. The grammar for the symbol in Fig. 3.6 is defined as $G = \langle P, S \rangle$, where $P$ is a set of production rule, and $S$ is the start symbol.

$$
P = \{ S \rightarrow b_1e_1E_0, \\
    E_0 \rightarrow b_2e_1E_0|b_2 \\
    \ldots \} 
$$

In this research, we propose a method for extracting relations between contents in tables by discovering the grammar for the tables based on an example of the relations given by humans.

### 3.3.2 Procedure for extracting ontologies

Our method for extracting contents in tables as ontology consists of the following three steps.
1. Give an interpretation of table structure

2. Generalize the table structure which is interpreted

3. Extract new relations between contents

In this section, we detailed each step.

**Give an interpretation** First our method requires humans to describe relations between contents in tables as RDF statements. The RDF statements represent a schema of contents to be extracted. For example, the RDF graph in Fig. 3.4 can be an example of interpretation of contents in Fig. 3.1. Boxes are labeled according to given RDF graph.

**Generalize the table structure** Next our method discovers iterative structures which are represented by boxes and generates production rules.

GenerateGrammar shown in Fig. 3.9 is a procedure which generates a grammar $G = \langle P, S \rangle$ from a sequence of symbols of a table structure and given RDF statements. This procedure first labels symbols which corresponds to boxes. Next it calls GenerateRule with the parameters set to the sequence of symbols and empty set respectively. GenerateRule returns production rules. In GenerateRule, a set of subsequence of $s_{box}$ is obtained ($Seq$). The element $\alpha$ in $Seq$ are sorted in ascending order of the length. Then it searches for a sequence of symbols which matches with $\alpha(e\alpha)^+$. If such a sequence is found, two rules $E_i \rightarrow E_i e\alpha$ and $E_i \rightarrow \alpha$ are added to $P$ and it makes $s'_{box}$ by replacing the longest subsequence that matches $\alpha(e\alpha)^+$ in $s_{box}$ with $E_i$, which is a new non-terminal symbol. If it found some symbols which can be replaced, it chooses one which appear in the latter in the sequence. When the replacement is performed, it recursively calls GenerateRule with the parameters set to $s'_{box}$ and $P$ which is updated.
by adding new production rules. However, it does not add rules nor replace symbols when the two adjacent symbols have different labels.

**Extract new relations** Using the grammar obtained by the previous step, our methods finally parses the given table structure and associates labels of boxes with contents in the corresponding cells. Therefore our method records the correspondence between boxes and labels when adding production rules. When the given table structure is accepted by the grammar, we can acquire new RDF statements according to the correspondence between boxes and labels.

ExtractRelation shown in Fig. 3.9 is a procedure which extracts RDF statements from a table table and the initial set of RDF statement $R$, which is given by humans. The procedure first transforms the given table into a sequence of symbols, and next it obtains the grammar for the table by GenerateGrammar. Then it analyzes the sequence according to the grammar $G$ by calling procedure Parse($s_{box}$, $G$) which returns extracted RDF statements.

### 3.4 Evaluation

In this section, we show the result of applying our method to language resources in tables on the Web.

We applied our method to dictionaries, which contain words in two or more languages and their meaning, and parallel texts, which contain tuples of expressions in multiple languages.

We focused on language resources developed by a department in a local municipality, an NPO or individuals because our goal is to allow people in communities of intercultural collaboration to publish their language resources as Web services.
Figure 3.9: Procedure for extracting relationships from a table.

We collected dictionaries and parallel texts on the Web linked from the Web site “Translation and Dictionaries”, that provides a number of links to language resources on the Web. Then we tried to apply our method and check how much language resources our method effectively works for. The result is shown in Table 3.1.

Completely applicable means that all the contents in the language resource are extracted and the extracted contents do not require revision by humans. We assume an example given by humans is one entry of a dictionary or parallel texts. Partially applicable means that some

*http://www.kotoba.ne.jp/
Table 3.1: Applicability to language resources.

<table>
<thead>
<tr>
<th>Applicability of generalization</th>
<th># of language resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completely applicable</td>
<td>28</td>
</tr>
<tr>
<td>Partially applicable</td>
<td>17</td>
</tr>
<tr>
<td>Not applicable</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3.2: Precision and recall

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>All applicable tables</td>
<td>97.6 %</td>
<td>97.7 %</td>
</tr>
<tr>
<td>Partially applicable tables</td>
<td>80.9 %</td>
<td>81.3 %</td>
</tr>
</tbody>
</table>

parts of the extracted contents require revision by humans or some parts of the table structure cannot be generalized by our method. Not applicable means that all of the extracted contents require revisions by humans or the whole structure cannot be generalized by our method.

The major reason of requirement of revision by humans is that cells in the table contain some different kinds of information. For example, expressions shown in a language resource often have notes. The expressions and notes are splited by delimiters such as white space and comma. But our method cannot recognize the notation because it relies only on table structures. We need to remove the notes before inputting the extracted contents into the database of the wrapper program.

Such notes are sometimes described with the expressions in all cells in some language resources. Our method does not work for such tables.

We also checked the precision and recalls of the result of our method. We consider the extracted entry which does not require any revision by humans as appropriately extracted. The result is shown in Table 3.2. The numbers in the table is average of tables which are shown in Table 3.1.

The second row in Table 3.2 shows the result of applying our method
to tables which are completely and partially applicable in Table 3.1. The
third row shows the result of applying our method to only tables
which are partially applicable.

Table 3.2 shows the result in the case we applied our method with
an example of one entry in a dictionary or parallel texts. But the
result about some of tables can be improved by more examples. This
is because the tables have multiple patterns of the table structure. For
the tables, we can give an example for each pattern.

Our method requires RDF statements as an example to extract
from a table. Once the example is given, the rest of processes are
automatically performed. The goal of our method is to reduce the cost
of extraction from tables. This is why the ratio of the cost of giving the
example and the amount of extracted contents is important. Therefore
we applied our method to general tables, not only language resources,
and checked how much examples are required and how much contents
are extracted.

The example given by humans and the extracted contents are des-
cribed in RDF statements. Thus we counted the number of the RDF
statements of the examples and the extracted contents. In this experi-
ment, we give a minimum number of RDF statements which can extract
all contents in the table as an example.

The result depends on the complexity of table structures. We follow
the classes of layout of tables that the previous works proposed[Pivk 04,
Wang 00] as follows.

**1-dimensinal table** Tables belonging to this class have one or
more rows of attributes above rows of attribute values. Attribute values
correspond to an attribute in the same column. Table 3.10(a) belongs
to this class.

**2-dimensinal table** Tables belonging to this class have a rect-
angular area in which attribute values are described. Attributes are
described in one or more rows above the rectangular area and in one
or more columns at the left side of the area. Table 3.10(b) belongs to
this class.
Figure 3.10: Classes of tables.

**Complex table** Tables belonging to this class have various features. On the basis of some features, the following three subclasses of this class are shown in the previous works.

**Partition label** In a table belonging to this class, Special labels make several partitions of the table. Each partition shares the same attributes. The table shown in Fig. 3.1 is an example of this class.

**Over-expanded label** In a table belonging to this class, some attributes and some attribute values can expand over multiple cells. The table shown in Fig. 3.10(c) is an example of this class.

**Combination** A table belonging to this class consists of several smaller tables. The table shown in Fig. 3.10(d) is an example of this class. The first two rows and the lower seven rows can be interpreted as two structurally independent tables.

Most of language resources belong to 1-dimensional, and some belong to over-expand according to the above classes.

The features of table structures also depend on the domains of the tables. Therefore we gathered tables from three different domains: price list, timetable and statistics. 25 tables were gathered in each domain and 5 tables for each table class were gathered. In order to gather tables, we used a search engine giving keywords “price list”, “timetable” and “statistics” and chose tables which actually belong to these domains.

We can find all the classes of table structure shown above for price lists. They usually contain a lot of records especially in a simple struc-
Table 3.3: Number of extracted RDF statements.

<table>
<thead>
<tr>
<th></th>
<th>price list</th>
<th>timetable</th>
<th>statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-dimensional</td>
<td>93.6 (3.6)</td>
<td>160.8 (4.4)</td>
<td>163.2 (7.2)</td>
</tr>
<tr>
<td>2-dimensional</td>
<td>36.0 (3.2)</td>
<td>84.0 (3.2)</td>
<td>169.0 (3.0)</td>
</tr>
<tr>
<td>Partition Label</td>
<td>215.8 (4.4)</td>
<td>136.6 (5.4)</td>
<td>193.8 (20.0)</td>
</tr>
<tr>
<td>Over-expanded Label</td>
<td>104.8 (4.8)</td>
<td>106.2 (8.0)</td>
<td>184.4 (6.0)</td>
</tr>
<tr>
<td>Combination</td>
<td>51.0 (6.8)</td>
<td>154.8 (5.6)</td>
<td>269.8 (4.6)</td>
</tr>
</tbody>
</table>

Timetables often have structures which belong to over-expand because the same time or schedule are merged. The most of structures of tables for statistics are 1-dimensional or 2-dimensional. But some of them have hierarchical structure in the attributes.

Table 3.3 shows the numbers of extracted RDF statements and RDF statements of given example in brackets for each table class 1-dimensional table, 2-dimensional table, Partition label, Over-expanded label, Combination in the three different domains.

The number of extracted RDF statements varies on the scale of tables. Our method works well for the table which contains simple iterative structures because it requires a small example and extracts lots of contents from such a table. On the other hand, a few RDF statements are required as an example for the most of tables.

In the following case, however, a lot of RDF statements are required as examples. First, we need to give many RDF statements if the table has many attributes although the structure is simple. We also need to give many examples in the case that the cells which are adjacent and have the same value are merged anywhere in the table. This is because the structure does not present the relation between data in the table.
A row or a column which has only blank cells is often used to represent separations. But our method cannot recognize the semantics as the separation. This is the reason we need to explicitly show the semantics by giving more examples.

We also collected 50 price lists by selecting tables which are actually price lists from the results of search by keywords “price list”. But 27 of them required human’s revision and 7 were not applicable. This is because tables designed by commercial companies focused on visual presentation using various colors and special effects more than those created by small communities of intercultural collaboration.

3.5 Related Works

In this research, we proposed a formalization of table structure based on adjacency of cells and iterative structure. Our goal is to extract information which iteratively occur in tables.

Many previous works have proposed the formalization according to their goals. We first introduce the formalizations by the previous works in this section.

In [Watanabe 95], the representation of table structures are divided into two types; global structure and local structure. Both structures are represented as a tree following the given rule. The goal of this research is to analyze a form printed on paper and extract information in the form.

Similar representation is proposed in [Duygulu 00] and the authors adopted XY-Tree[Nagy 84] in order to represent table structures. A table is recursively divided and the structure is represented as a tree. Nodes in the tree represent a block which consists of cells. A parent node represents a block which contains cells corresponding to child nodes. The root node represents the whole table and a leaf node represents a cell. This representation divides table into parts based on spanning cells. The authors insist that this representation is appropri-
ate for representation of forms because forms often contain spanning cells.

In [A.Amano 02], the authors proposed a representation of form in TMFL, which is an extension of XML. By using the language, a form is represented as a graph based on adjacency of cells. Nodes of the graph contain information on roles of the corresponding cell and whether the corresponding cell has value or not. These information are used to analyze the table structure. The authors insist that their grammar based on XML and graph make it possible to change the view of forms preserving consistency of contents described in the form. Graph representation for forms and tables are also proposed by [Rahgozar 96].

Such representations for tables are summarized in [Lopresti 00]. An application which is appropriate for each representation is also explained in this paper.

In our work, we have manually found the tables which have contents as language resources. But we need automatic method to find wide range of tables from Web. This work is not trivial because table tag in HTML is often used to locate contents on the Web page.

Previous works have tried to use heuristics about structure by HTML tags[Chen 00], machine learning[Wang 02b, Wang 02a]. The proposed in [Chen 00] method determine if the given table tag is a real table or not based on similarity between data in adjacent cells and types of data in cells. In [Wang 02b, Wang 02a], the author adopted decision tree or SVM taking characteristics of layout of the table and types of data as features for the algorithms.

When we try to automatically find language resources on the Web, we have to adopt these works as a preprocess of our algorithm.

As for interpretation of tables for information extraction, many previous works have tried various approaches. A major approach is to match the given table to a similar typical table structure[Chen 00, Lim 99, Wang 00]. In [Pivk 04], the interpretation is performed based on recognition model for tables[Hurst 99]. In [Cohen 02], inductive learning was adopted to learn wrappers for tables. Some works have
used the domain-specific knowledge base to determine the types of data in tables [Embley 02, Tijerino 03].

In [Cohen 02], the proposed method detects real tables in HTML and extracts attribute-value pairs from the detected tables. Their method first determines the orientation (vertical or horizontal) of the table based on similarity between data in adjacent cells. According to the assumption that an attribute and a set of values located in adjacent region, the correspondence is found based on the locations. The method also extracts relations between attributes when attributes are hierarchically described. However, the types of tables that their method covers are limited because the algorithm of analysis is simple. Moreover, they have not shown the result of extraction of attribute-value pair.

The method proposed in [Lim 99] focuses on extraction of hierarchical data from tables. But the interpretation on which parts contain what kind of contents follows expected table structure which is given assumption. For example, they assume the key of a record is located in the most left column. This limits the types of table that they can apply their method. The authors insisted that the advantage of this work is to allow us to give a complex query and to transform documents HTML into XML based on extracted hierarchy.

In [Wang 00], the authors showed that extracted attribute-value pairs are transformed into the format for database by their method. They used knowledge base in order to interpret the semantics of table structure. Using the knowledge base, their method determines the types of data in cells and matches the given table structure with one of predefined typical structure. They also proposed table classes for evaluation, and we adopted their classes as well as [Pivk 04].

The approach shown in [Cohen 02] is based on inductive learning. Based on the given examples for learning, the method learns rules about HTML tags and their contents. They conducted an experiment and shows that their method works with a small number of examples. But the result of learning is not useful for different tables.

Some works have tackled ontology extraction from tables, not only
attribute-value pairs [Pivk 04, Embley 02, Tijerino 03]. As we described
in the previous section, the method proposed in [Pivk 04] interprets
tables following the recognition model of tables [Hurst 99]. The method
first classifies the region into two types; regions of attributes and regions
of values. The correspondences between them are determined based on
their locations. They assume that attribute are located on upper or
left region of the corresponding values. The classification of the type
of region is based on types of data (string, number, date, etc.) and the
locations. The result of extraction is given as frames.

In [Yoshida 01], the authors proposed a method for integrating
tables which have the same type of contents. They predefine expected
types of tables and determine the type of the given table using EM
algorithm. For the integration of tables, they used clustering algorithm
based on co-occurrence of words and similarity of words of attributes.

The goal of the method proposed in [Embley 02] is to acquire the
mapping between schemas of tables. They adopted the method pro-
posed in [Lim 99] to extract attribute-value pairs. They also used do-
main specific wrapper in order to extract contents following the given
schema.

In [Tijerino 03], the authors showed the way of extending ontology
which is first given by human based on the extracted information from
tables. The extension exploits the information which is already ac-
quired. Various sources such as functional dependency, domain specific
knowledge base and similarity between values are used for the exten-
sion. They also showed an example which is an extension of geograph-
ical ontology by extraction from two tables. But they just proposed a
general framework for extending ontology, not a process for automatic
acquisition of extracting ontology.
3.6 Summary

In this research, we proposed a method for extracting ontology from tables. This aims at construction of language services from the language resources on the internet. The contributions of this work are as follows.

**Generalization based on interpretation by humans**  The semantics of the table structure depends on each table. Our method extracts relations from tables according to the semantics of each table by introducing interpretation given by humans.

**Applicability to tables in various domains**  Language resources on the internet vary in various domains. Our method is easy to apply to tables in various domains because it uses interpretations given by humans and generalize table structures instead of a domain-specific knowledge base.

We applied our method to dictionaries and parallel text in tables. The result shows that our method is applicable to most of the language resources and the extracted contents require few human revisions. We also applied our method to tables which belong to various classes of the structure and whose contents are in general domain. As the result of our experiment, we confirmed that our method can extract a lot of RDF statements by giving a small example.
Chapter 4

Predicting Connectability of Web Services

4.1 Introduction

We can construct language services from language resources based on the method proposed in the previous chapter. Once we standardize the interface of the language services based on the types of the language services, we can develop and reuse a composite Web service which consists of the services in the following way.

A composite Web service, described in WS-BPEL\[bpe 03\] or OWL-S\[owl 04\], can contain atomic services whose endpoints are not set (abstract services). Only the interfaces and basic functions are defined for the atomic services. Web services actually executed (concrete services) are specified by setting endpoints for the abstract services. This enables us to develop and use a composite Web service in the following way. Various service providers release their services on the Web as concrete atomic services. A designer of a composite Web service provides a composite Web service which is composed of abstract services (an abstract composite service). The developer of an application that uses an abstract composite service sets endpoints for the service in his composite service (a concrete composite service); the goal is to realize
the functions required for his application.

For example, various organizations provide different language services such as dictionaries or machine translations for Language Grid, an infrastructure for Web services. They have standardized the interfaces of the services based on a language service ontology [Hayashi 07]. This makes it possible for a developer to easily modify the functions of his application such as supported language sets simply by changing the endpoints of atomic services.

However, a concrete composite Web service developed in the above way may suffer runtime failure. This is due to the following reason. Although an abstract composite Web service can be designed by combining abstract atomic Web services, WSDL, the standard language for definition of interfaces of Web services, does not represent the full variety of specifications possible with concrete atomic Web services. For example, an abstract hotel reservation Web service is defined to take the date of the reservation from a client as a request message. The valid range for the date depends on the concrete atomic Web service. Unfortunately, WSDL definitions for the abstract atomic Web service do not include this specification. Thus the execution of a concrete composite Web service which contains atomic Web services may fail for some request messages if it is configured by setting endpoints for the abstract atomic Web services. When a request message triggers execution failure of any atomic Web service in the composite Web service, the cost of executing all prior atomic Web services that is wasted.

Previous works have proposed methods with verification techniques such as petri net [Narayanan 02] and model checking [Ankolekar 05, Fu 04, Nakajima 06] in order to prevent execution failure. Their goal is to prove that a composite Web service is proof against execution failure regardless of the request message. However, these previous works are not useful because providers of concrete atomic Web services, designers of abstract composite Web services, and application developers reside in different organizations.

Even if an application developer verifies a concrete composite Web
and determines that it might suffer runtime failure, he/she can modify
neither the concrete atomic Web services nor the abstract composite
Web service. Moreover, the application developer cannot always per-
form the verification because the specifications of concrete Web services
required for the verification are often unavailable.

To solve the above problems, we addressed the following issues.

• To ensure that Web services in a composite Web service are con-
nectable, we need to predict the connectability of the Web ser-
vices for each request message based on as much specifications as
is known.

• To improve the accuracy of connectability prediction, we need to
acquire the specifications of concrete atomic Web services based
on a success/failure of execution for each request message.

To predict connectability, we model atomic and composite Web ser-
vices in a formal specification. This makes it possible to predict the
connectability of all atomic Web services in a composite Web service
by using a theorem prover. Moreover, we apply a constraint acquisition
algorithm[Bessière 07] to acquiring the input specifications of concrete
atomic Web services. We can improve the prediction of connectability
by updating the models of atomic Web services based on the acquisition
results.

4.2 Formal Specification to Model Web Services

This section introduces a formal specification to model of Web ser-
vices with the goal of predicting connectability. The model is based
on CafeOBJ[Futatsugi 97], which is an algebraic formal specification
language[Ehrig 85]. While the complete specifications of Web services
are not always known, CafeOBJ allows us to describe the specifications
as far as is known.
To predict the connectability in a composite Web service, we need the following specifications of its constituent atomic Web services.

- Request message to response message mapping (input/output relation)
- Request message validity (input specification)

The former is required to create the chain that starts with the request message to the composite Web service, runs through each atomic Web service, and ends with the response of the composite Web service. Note that the request messages to most atomic Web services will be the response messages of other atomic Web services.

The latter is used to predict whether each atomic Web service in the composite Web service is connectable or not for the request messages that could be passed to the atomic Web service given its situation, i.e. its position in the composite Web service. If all atomic Web services are connectable, the composite Web service is executed.

To allow the above specifications to be checked, we model an atomic service as a module in the formal algebraic specification that consists of the following two operations.

`domain-service-name` This operation represents the input specification. This takes a request message to the Web service and returns true or false. True means executable and false means not executable.

`execute-service-name` This operation represents the input/output relation. This takes a request message to an atomic Web service and returns the response message of the atomic Web service.

We note, however, the input specification and input/output relations are not always completely known. Moreover, in general, it is impossible to describe the input/output relations completely because the response message of a composite Web service (its output) given one particular request message, is unknown before the service is executed.
mod TRANSLATOR-REQUEST {
    -- Input is (sourceLanguage, targetLanguage, expression)
    pr(3TUPLE(C1 <= view to LANGUAGE { sort Elt -> Language },
               C2 <= view to LANGUAGE { sort Elt -> Language },
               C3 <= view to EXPRESSION { sort Elt -> Exp })
        * { sort 3Tuple -> TranslatorRequest })
}

Figure 4.1: Definition of request message.

This is why we describe constraints on values or types of elements of request/response messages as far as are known.

We detail the above process using the example of a machine translator Web service. Figure 4.1 shows that the request message to the Web service is the 3-tuple named TranslatorRequest; it consists of source language, target language and the string to be translated in this order. The first two elements have type Language and the third has type Exp.

Figure 4.2 shows the specifications of the machine translator Web service.

mod TRANSLATOR {
    pr(LANGUAGE + TRANSLATOR-REQUEST + TRANSLATOR-RESPONSE)
    op domain-translator : TranslatorRequest -> Bool
    op execute-translator : TranslatorRequest
                           -> TranslatorResponse
    var e : TranslatorRequest
    eq domain-translator(e) = 1*(e) == english or 1*(e) == japanese .
    eq get-language(execute-translator(e)) = 2*(e).
}

Figure 4.2: Specification of a machine translator service.

First the definitions of data types and messages are imported (line
2). The response message named TranslatorResponse is a string which is the result of the translation into the target language. Next, the two operations which represent input specification and input/output specification are declared (lines 4-5). Finally, axioms are described as equations following eq (lines 10-11,13). In this example, the first axiom states that the first value of the request message (source language) must be English or Japanese. The second axiom states that the Web service translates a given string into the language that is specified by the second value of the request message (target language). “n*” is an operation on N-tuple which extracts the nth value.

We extended the representation of program specifications shown in Problem Theory[Smith 90] to define our model; it allows constraints to be set for specific request/response messages.

WSDL provides definitions of data types for inputs and outputs in a similar way. Therefore, we can easily transform definitions in WSDL into specifications in CafeOBJ.

Note that WSDL doesn’t provide a definition of constraints, which corresponds to axioms of operations. Thus a designer of a composite Web service needs to define the specifications according to the abstract service. In the case of OWL-S, hasPrecondition and hasEffect properties may help in defining axioms.

Predicting the connectability in a composite Web service requires the specifications of the composite Web service, in addition to the specifications of its constituent atomic Web services. This allows us to update the specifications of the composite Web service simply by updating the specifications of atomic Web services, when the atomic Web services are replaced or updated.

Our approach is to create the specifications of a composite Web service by combining the specifications of its constituent atomic Web services in a bottom-up way using control constructs. The features of the control constructs are represented as axioms in formal specifications.

In OWL-S and WS-BPEL, a composite Web service has nested
structures. A control construct block contains atomic Web services or other control construct blocks. We follow this and recursively define the specifications of control construct blocks. To allow this, we need a uniform specification for both atomic Web services and control construct blocks. Thus we consider a control construct block as a Web service and define it using the request/response message, the input specification, and the input/output relation. The block that contains all other blocks and atomic Web services corresponds to the composite Web service.

Here we define two operations to represent the input specification and the input/output relation in the specification of each control construct block.

Furthermore, we have to represent dataflows between blocks of control constructs and atomic Web services and constraints based on the features of the control constructs. Dataflows establish the correspondence between the elements of request/response messages. Constraints based on control constructs define the order of atomic Web service execution and the blocks inside them and the conditions of execution. For example, the Sequence control construct requires the atomic Web services and the blocks inside them to be executed serially, and all atomic Web services and blocks must be connectable. The While control construct, on the other hand, ties execution of the atomic Web services and blocks inside them to satisfaction of the condition. If the condition is not satisfied, they do not need to be connectable. Dataflows and constraints based on features of control constructs are represented as axioms in the specification.

We explain the specifications of a composite Web service by taking Fig. 4.3 as an example. Figure 4.3 shows a composite Web service consisting of two services, Machine Translator A and B, in a Sequence control construct block. The string in a request message given to this composite Web service is first translated into another language (target language), and then translated into the source language in order to estimate the quality of translation in the target language.
Elements of a request message and a response message are shown to the above left and above right of the block and each atomic Web service, respectively. The request message to the composite Web service is the 3-tuple that consists of source language, target language, and the string to be translated. The response message of the composite Web service is the string output by Machine Translator B. Links between the elements show the dataflows.

Figure 4.4 shows the specification of the Sequence block in Fig. 4.3. The declarations of operations follow those of the atomic Web services. Constraints based on control constructs are represented as axioms. The first axiom is defined as the conjunction of operations of input specifications and states that both machine translator Web services must be connectable. The second axiom is defined as the input/output relation operation of Machine Translator B which accepts the response message from Machine Translator A. This axiom states that the response message of the block is the response message of Machine Translator B. Dataflows are represented by operations on the N-tuples of messages.

To predict the connectability of a composite Web service, we reduce the operation `domain-service-name` of the block that corresponds to the
overall composite Web service. The request message to the composite Web service is given as the parameter of the operation. Reduction rewrites terms based on equations defined as axioms using a theorem prover. When the result of reduction is \textbf{true}, our method predicts that Web services in the composite Web service can be connectable for the request message and starts execution. If the result of reduction is \textbf{false}, our method cancels execution.

Our method allows specifications of Web services to be described as far as are known. This may lead to execution failure even if the prediction result is “connectable”. As more input specifications are described, more request messages that trigger execution failure can be detected monotonically by our method as long as the described input specifications are correct.

During the execution of a composite Web service, our method can

Figure 4.4: Specification of the block in Fig. 4.3.

```
mod SEQUENCE {
  ...
  op domain-sequence : SequenceRequest -> Bool
  op execute-sequence : SequenceRequest -> SequenceResponse
  ...
  var request : SequenceRequest
  ...
  eq domain-sequence(request) =
    domain-translator-a(
      1*(request), 2*(request), 3*(request))
  and domain-translator-b(
    2*(request), 1*(request),
    execute-translator-a(
      1*(request), 2*(request), 3*(request))).

  eq execute-sequence(request) =
    execute-translator-b(
      2*(request), 1*(request),
      execute-translator-a(
        1*(request), 2*(request), 3*(request))).
}
```
predict the connectability of atomic Web services and blocks that have not been executed yet. It can detect more request messages that will trigger failure because we can use response messages from atomic Web services and blocks which have already been executed for the prediction. If any of the prediction results for atomic Web services and blocks is unconnectable, our method immediately cancels further execution.

Prediction during execution increases the computational load. However, reducing an operation by using a theorem prover is usually much faster than XML message exchange among Web services. Therefore, prediction does not degrade execution performance.

Figure 4.5 shows the specification of the control construct If-Then-Else, which represents conditional execution. We suppose two services A and B are contained in the block. If a given condition is true, service A is executed. Otherwise, service B is executed.

```plaintext
mod IF-THEN-ELSE {
  ...
  op condition : IfThenElseRequest -> Bool
  ...
  eq condition(request) = ...
  ...
  ceq domain-if-then-else(request) =
    domain-a(compose-request-a(request))
    if condition(request).
  ceq domain-if-then-else(request) =
    domain-b(compose-request-b(request))
    if not condition(request).
  ceq execute-domain-if-then-else(request) =
    execute-a(compose-request-a(request))
    if condition(request).
  ceq execute-domain-if-then-else(request) =
    execute-b(compose-request-b(request))
    if (not condition(request)).
}
```

Figure 4.5: Formal specification of If-Then-Else.

We introduce an operation condition in order to represent a condi-
tional. This takes the input to the If-Then-Else block as its input and returns true or false. `ceq` defines an axiom expressed by a conditional equation.

If the condition is true, the input to service A must be valid and the output of this block is composed of the output of service A. If the condition is false, the input to service B must be valid and the output of this block is composed of the output of service B.

“If” without an “Else” block can be defined by composing the output of this block of the input to the block when the condition is false. WS-BPEL provides Switch and Case control constructs instead of If-Then-Else, but they can be used in the same way.

Figure 4.6 shows the specification of the control construct Split+Join, which represents concurrent execution. We suppose two services A and B are contained in the block.

```plaintext
mod SPLIT-JOIN {
  ... 
  eq domain-split-join(request) =
      domain-a(compose-request-a(request))
      and domain-b(compose-request-b(request)).
  eq execute-split-join(request) =
      compose-response-split-join(
        execute-a(compose-request-a(request)),
        execute-b(compose-request-b(request))).
}
```

Figure 4.6: Formal specification of Split+Join.

The declarations of the two operations are omitted in Fig. 4.6. The axioms state that both the input to Service A and that to B must be valid and that the output of this block is composed of the output of service A and that of B. WS-BPEL provides Flow control constructs instead of Split+Join, but the specification of Flow can be described in the same way.

Figure 4.7 shows the specification of the control construct Choice, which represents the execution of one of a given set of services. We
suppose two services A and B are contained in the block. The service with higher priority is executed if the service is connectable. Otherwise, the other one is executed if connectable.

Choice is a control construct of OWL-S. WS-BPEL has no control construct corresponding to Choice.

\[
\text{mod CHOICE} \{
\begin{align*}
\text{eq domain-split(request) = } & \text{domain-a(compose-request-a(request))} \\
& \text{or domain-b(compose-request-b(request)).} \\
\text{ceq execute-split(request) = } & \text{compose-response-choice(execute-a(} \\
& \text{compose-request-a(request)))} \\
& \text{if domain-a(compose-request-a(request)).} \\
\text{ceq execute-split(request) = } & \text{compose-response-choice(execute-a(} \\
& \text{compose-request-b(request)))} \\
& \text{if (not domain-a(compose-request-a(request)))} \\
& \text{and domain-b(compose-request-b(request)).}
\end{align*}
\]

Figure 4.7: Formal specification of Choice.

The first axiom states that the input to at least one of the services in this block must be valid. The second axiom states the output of this block is composed of the output of the service executed.

Figure 4.8 shows the specification of the control construct While, which represents the iterated execution of a service. We suppose that service A in this block is iteratively executed and the output of service A affects the iteration process.

condition operation is defined in the same way as If-Then-Else. If the condition is true, the input to service A must be valid. Otherwise, any input to this block is valid because service A is not executed.

We introduce \text{compose-request-while}, which defines dataflow, in order to take the output of service A as a condition. Axioms of \text{domain-while} and \text{execute-while} are recursively defined using this.
mod WHILE {

  op condition : WhileRequest -> Bool
  op compose-request-while :
    AResponse -> WhileRequest

  eq condition(request) = ...

  ceq domain-while(request) = true
    if not condition(request).
  ceq domain-while(request) =
    domain-a(compose-request-a(request)) and
    domain-while(compose-request-while(
      execute-a(compose-request-a(request))))
    if condition(request).

  ceq execute-while(request) =
    compose-response-while(request)
    if not condition(request).
  ceq execute-while(request) =
    execute-while(compose-request-while(
      execute-a(compose-request-a(request))))
    if condition(request).
}

Figure 4.8: Formal specification of While.

operation.

OWL-S provides several more control constructs: Split, Iterate, Repeat-While, Repeat-Until and Any-Order. These, except for Any-Order, can be used in the same way as the control constructs described above. We note that it is impossible to represent Any-Order in our specification. WS-BPEL also provides repeatUntil and forEach. They are essentially the same While.

4.3 Acquiring Input Specifications

Complete specifications of Web services are not always known, especially in the case of Web services on the Web. The prediction of connectability described in the previous section works even if the given
specifications of atomic Web services are incomplete, but its effectiveness declines. Therefore, we propose a method that acquires the input specifications of atomic Web services to improve prediction accuracy.

In our model described in the previous section, input specifications of a Web service are represented as a logical formula. Thus we adopt the constraint acquisition algorithm [Bessi`ere 07] to acquire the input specifications because the result of the acquisition can be represented as a logical formula in the formal specification.

The constraint acquisition algorithm is based on version space learning. Given predefined predicates, which represent the possible constraints, and a set of training examples, it acquires the conjunction of the predicates for which only positive examples satisfy.

Version space learning acquires a set of hypotheses from positive and negative examples. The following notation represents that a hypothesis $h$ is consistent with a set of examples $D$. $c(x)$ represents if an example $x$ belongs to a concept, and $h(x)$ represents if $x$ belongs to a concept.

$$\text{Consistent}(h, D) \equiv (\forall x, c(x) \in D) h(x) = c(x)$$

Version space is a set of hypotheses which satisfy the above condition and defined as shown below.

$$VS_{H,D} \equiv \{h \in H | \text{Consistent}(h, D)\}$$

Figure 4.9 shows an example of version space. This is a set of hypotheses which determine if a set of values about weather belongs to a concept or not. ? represents the attribute does not care the value. Given the examples, we can define the most general set of hypotheses ($G$) and the most specific set of hypotheses ($S$) because partial order between the hypotheses can be defined. Version space is the set of hypotheses between $G$ and $S$.

In our method, a request message to an atomic Web service and the success/failure of the execution of the message are given to the constraint acquisition algorithm as a training example. The acquisition
result can be easily transformed into descriptions in the formal specification by defining logical formulas in the formal specification that correspond to predefined predicates.

We explain below how to model the input specifications to apply the constraint acquisition algorithm.

First we define a request message to a service which has $k$ elements as follows.

$$I = \{x_1, \ldots, x_k\}$$

Next we define predicates which represent input constraints. For the sake of simplicity, we assume that the predicates have one or two variables. We refer to a unary constraint on an input value $x_i$ to $i$th parameter as $b_i$. We also refer to a binary constraint on input values $x_i$, $x_j$ to $i$th and $j$th parameters as $b_{(i,j)}$.

$b_i$ and $b_{(i,j)}$ can be defined as follows:

$$b_i : x_i \in \text{class} \text{ or } x_i \notin \text{class}$$

$$b_{(i,j)} : \{x_i, x_j\} \in \text{class}_1 \times \text{class}_2 \text{ or } \{x_i, x_j\} \notin \text{class}_1 \times \text{class}_2$$

$class$, $\text{class}_1$ and $\text{class}_2$ represent any class. $x_i \in \text{class}$ indicates that $x_i$ is an instance of $\text{class}$. $\text{class}_1 \times \text{class}_2$ represents a Cartesian product set of $\text{class}_1$ and $\text{class}_2$. Constraint library $B_s$ contains $b_i$ and $b_{(i,j)}$ for all known classes or pairs of classes.

The constraint acquisition algorithm works based on the above formalization. When a positive example is given, the constraint acquisition algorithm adds to formula $K$ ($K = \text{true}$ in the initial state) the
conjunction of negation of all constraints in the constraint library that the example does not satisfy. When a negative example given, it adds to $K$ the disjunction of all constraints in the constraint library that the example does not satisfy. The acquisition result is the conjunction of literals that should be set to true in order to satisfy $K$.

We explain the process of constraint acquisition using the following request messages to the machine translator Web service described in the previous section.

- (Positive) $I_1 = \{ \text{ja, en, “Kon-nichiha” (Japanese)} \}$
- (Negative) $I_2 = \{ \text{ja, ko, “Hello”} \}$
- (Positive) $I_3 = \{ \text{ja, ko, “Kon-nichiha”} \}$

“ja”, “en” and “ko” represent “Japanese”, “English” and “Korean” respectively. The service can be executed by positive examples, but execution fails when a negative example is given.

Table 4.1 shows the process of updating $K$. $K_i$ is the result from $I_i$. Literals in $K$ are predicates defined in the constraint library*. $l_s$, $l_t$ and $l_e$ represent the source language, the target language and the actual language of the string to be translated respectively. Moreover we used a notation $l_1 = l_2$ to represent that languages $l_1, l_2$ are same as $l_1 = l_2$ for simplicity.

In this example, only $(l_s = l_e)$ is true and other literals of $K$ are false. Thus this literal is added to the axiom in formal specification. Figure 4.10 shows the updated axioms of specifications shown in Fig. 4.2. “:is” is a predicate which checks whether a term on a right-hand side is a sort of a term in the left-hand side. SameLanguagePair is a class which has pairs of the same two languages as the instances.

In general, the set of literals that satisfy $K$ is not unique. Thus we consider the possible sets of literals that satisfy $K$ as a set of hypotheses,
Table 4.1: Process of acquiring input constraints.

<table>
<thead>
<tr>
<th>$K_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg(l_s \in en) \land \neg(l_t \notin ja) \land \neg(l_s \in ko) \land$</td>
</tr>
<tr>
<td>$\neg(l_t \notin en) \land \neg(l_t \in ja) \land \neg(l_t \in ko) \land \neg(l_s \neq l_e)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$K_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_1 \land {(l_s \in en) \lor (l_s \notin ja) \lor (l_s \in ko) \lor$</td>
</tr>
<tr>
<td>$(l_t \in en) \lor (l_t \in ja) \lor (l_t \notin ko) \lor (l_s = l_e)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$K_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_2 \land \neg(l_t \in en) \land \neg(l_t \notin ko)$</td>
</tr>
</tbody>
</table>

eq domain-translator(e) =
(1*(e) -- english or 1*(e) -- japanese)
and (language-pair(1*(e),
get-language(3*(e))):
is SameLanguagePair).

Figure 4.10: An updated axiom.

$H$, and define a partial order $\prec$ between hypotheses in $H$ as follows:
$h_i \prec h_j \equiv (\forall x \in X)[h_i(x) = true \rightarrow h_j(x) = true]$. $X$ is a set
of possible values of a request message. $h_i(x) = true$ means that the
atomic Web service is connectable for request message $x \in X$ under the
hypothesis $h_i$. Our method performs prediction by reducing operation
domain-service-name under all $h_g$ defined as follows: \{h_g \in H|h_g \not\prec (\forall h \in H)\}. Our method cancels execution only if the results of reducing
under all $h_g$ are false. This is because our prediction of connectability
involves detecting request messages that would cause service execution
to certainly fail.

As the number of predefined predicates in the constraint library in-
creases, the computational complexity of acquisition by the constraint
acquisition algorithm also increases. However, acquisition can be per-
formed after execution of a composite Web service finishes or is can-
celled. Therefore, our method does not degrade execution performance.

Figure 4.11 shows the process of the prediction by using specifica-
tions and constraint acquisition described above.
When an input is given, the process starts with a prediction of connectability. The prediction is done by reducing the term “domain-composite-service(input)”, where input is an input to the composite service defined in the formal specification language, and domain-composite-service is an operation for checking connectability defined in the specification of the composite Web service.  

As described before, the constraint acquisition algorithm can obtain several possible specifications. The prediction is done using all possible specifications. If the result of the reduction is false with any specification, the process exits. Otherwise, execution of the composite service is started. If any atomic service fails during execution, the input to the service is given to the constraint acquisition algorithm as a negative example. The updated specifications are accumulated for future execution.  

Figure 4.12 shows the architecture of our prediction mechanism. Components in the dashed box are responsible for the processes shown.
in Fig. 4.11. Theorem Prover takes an input to a composite service and specifications and returns the result of prediction. Execution Engine interprets the composite service described in WS-BPEL or OWL-S and executes it using the input. Constraint Acquisition takes the inputs given to atomic services and the result of the execution and gives input constraints of the services.

Figure 4.12: The architecture for predicting and learning connectability.

4.4 Experiment

We conducted an experiment to show how much our method can improve the efficiency of executing a composite Web service.

We applied our method to a composite Web service for translation used in Language Grid. The composite Web service shown in Fig. 4.13 aims at translate sentences in a special domain. When the given string to be translated includes technical terms, the corresponding words in the string translated by the machine translator Web service are replaced by words found in a technical term dictionary.

The details of the process of the composite Web service are as follows:
1. Split the string given as a request message into words using Morphological Analyzer (MA)

2. Concurrently execute the followings:
   
   (a) Translate all the words by Technical Term Dictionary (Dic)
   
   (b) Translate all the words by Machine Translator 1 (MT1)
   
   (c) Translate the string given as a request message by Machine Translator 2 (MT2)

3. Term Replacement (TR) replaces words in the string translated by MT2 with corresponding words translated by Dic

MT1, MT2, and Dic in the composite Web service handle request/response messages whose forms are the same as that of the machine translator Web service described in Section 4.2. Morphological Analyzer (MA) takes a string, which is assumed to be a sentence, and its language as a request message, and returns an array of words, which the given string consists of, as a response message. TR takes the string and the two word arrays as a request message. It outputs a string in
which the words in the given string match an element in one array are replaced with the corresponding elements in the other array.

Suppose MA, MT1 and MT2 have the following input specifications.

- **MA**: Fail if any language other than Japanese or English is specified for the language of the given string.

- **MT1, MT2**: Fail if the specified source language is different from the actual language of the given string or the given string is longer than 100 characters.

These input specifications lead to the possible failure of execution of the composite Web service as shown below.

- When the source language is neither Japanese nor English, MA fails and the cost of MA execution goes counted as waste.

- When the given string is longer than 100 characters, MT2 fails and the cost of one execution of MA and MT2 and the iterated execution of MT1 and Dic is counted as waste.

- When the given string has words in multiple languages (e.g. Japanese sentences often contain English words.), MT1 fails because the actual language of some of the given words differ from the source language specified. In this case, the cost of one execution of MA and MT2 and the iterated execution of MT1 and Dic are counted as waste as in the previous case.

We applied our method under the conditions described above. We assume that all input specifications of the atomic Web services are unknown in the initial state. Moreover, we defined predicates for the constraint acquisition algorithm. The predicates involve the input specifications of MA, MT1 and MT2. They represent classes for the source language, the target language, and the actual language of the given
string (three classes for each; English, Japanese and Korean) as described in Section 4.3. In this experiment, we also defined predicates to represent given string length (two classes: short and long).

We generated request messages of all combinations of the classes for each element and executed the composite Web service by giving the messages in random order. The number of generated request messages was 54, because we have three classes for each of source language, target language and actual language of the string and two classes of string length ($3 \times 3 \times 3 \times 2$).

We counted failure of the execution of the composite Web service due to failure of any of the atomic Web services as one of the wasteful executions of the composite Web service. Similarly, we counted the sum of the executions of atomic Web services until one of the atomic Web service failed as the number of wasteful executions of atomic Web services. Figure 4.14(a)(b) compares the numbers of wasteful executions of the atomic/composite Web services shown in Fig. 4.13 before and after applying our method, respectively.

The figures show that the rate of increase in the number of wasteful executions saturates as the number of execution increases and more input specifications are acquired. In particular, Fig. 4.14(a) shows that our method works well in our example because it prevents some atomic Web services from being iteratively executed after the failure of some atomic Web service. This is very effective in reducing the cost of executing atomic Web services. However, we can see in Fig. 4.14(b) that wasteful execution continues to occur even after our method acquires as many input specifications as possible. This is because a failure of MT1 can be predicted only after the execution of MA. This limitation comes from the fact our method considers combinations of specifications of atomic Web services.

Moreover, the result also suggests that our method can detect most of the request messages which cause failures in rather early stages of acquisition. In our experiment, input constraints which can be learned using the given predicates were acquired after execution of 12 request
messages. The reason is that we defined predicates which properly represent dependencies between elements in a request message. Although a predicate for any combination of elements can be defined, we used one binary predicate for the combination of a source language and an actual language of the given string and unary predicates for each element. This makes it possible for our method to acquire the common features of request messages which cause execution failure without testing values of any combinations of the elements.
4.5 Related Works

Some papers have examined the verification of composite Web services based on logic, process algebra, petri net, and finite state machine. In this section, we briefly overview previous works and describe the difference between their objectives and ours.

Narayanan et al. modeled composite Web services in DAML-S, which is a predecessor of OWL-S, as petri nets. Their work made it possible to validate the reachability to the goal state and to detect deadlocks [Narayanan 02].

Ankolekar et al. proposed a method which detects inconsistency between control flows and dataflows by transforming OWL-S into a language for model checking [Ankolekar 05]. The method proposed by Fu et al. is also based on model checking and validates interaction protocols between Web services [Fu 04]. Nakajima proposed a method which extracts behavioral specification from BPEL and translates it into descriptions in language for model checking [Nakajima 06]. The methods by Nakajima and Fu et al. can flexibly adopt various combinations of languages for composite Web service and languages for model checking. The works we mentioned here adopt SPIN model checker [Holzmann 04].

The major difference between these works and our work is the assumption about how to develop composite Web services and the relationships among stakeholders. Most of the previous works assume that specifications of Web services are sufficiently well known to permit validation, and that an application developer can modify Web services to configure the functions for the application if needed. On the other hand, our work assumes that providers of concrete atomic Web services, designers of abstract composite Web services, and application developers are in different organizations, and so cannot modify the products made by others. In such a situation, it is important to predict the connectability in composite Web services for each request message.
4.6 Summary

In this paper, we proposed a method for predicting the connectability of Web services in a composite Web services in order to reduce the cost of wasteful executions of atomic Web services. Although standardization of the interfaces of Web services makes it easy to configure composite Web services, the complexity of the specifications of atomic Web services often causes execution failure of composite Web services. The major contributions of our method are as follows:

- We showed a model of Web services in a formal specification and applied it to predict the connectability in a composite Web service for each request message by using a theorem prover.

- We applied the constraint acquisition algorithm in order to acquire input specifications of atomic Web services and showed that it improves the prediction of connectability.

We conducted an experiment in which our method was applied to a composite Web service in practical use. The results showed that our method could detect almost all request messages that would cause execution failure. Moreover, the results also showed that input specifications useful in raising prediction accuracy are acquired rather early in the acquisition process. The experiment also showed that our method was quite effective in reducing the cost of executing atomic Web services by identifying, and then terminating, the wasteful iterative execution processes created by the failure of some of the atomic Web services.

Our method is applicable to a wide variety of composite Web services because our model can represent most of the control constructs provided in OWL-S or WS-BPEL. Moreover our method allows us to define predicates which represents input specifications as far as are known. This also increases the applicability of our method.

Compared to previous works, we assume that the application developer cannot modify concrete atomic Web services or abstract composite Web services because the providers of concrete atomic Web services,
designers of abstract composite Web services, and application developers are in different organizations. Therefore, the application developer has to assume that a composite Web service may fail for some request messages. This paper is the first work that focuses on the point and tries to reduce the wasteful execution of Web services by predicting the connectability of each request message.

Although input/output relations can be unique to each concrete atomic Web services, our work acquires only input specifications of the atomic concrete Web services. Therefore we will consider acquisition of input/output relations of concrete atomic Web services in future work.
Chapter 5

Meta-level Control of Composite Web Services

5.1 Introduction

Once language services which wrap language resources of various organizations are published in open environment, anyone can develop a new composite Web service which combines them.

For example, programs for language processing such as machine translators and morphological analyzers, and contents like dictionaries and multilingual corpora are now available on Language Grid as Web services. The programs and contents have been widely collected from various organizations like companies and research institutes. This leads to the creation of new composite Web services created by fusing the elemental Web services provided by various organizations.

However, the programs, the contents and the servers of the elemental Web services continue to be managed by their providers. Moreover, an elemental Web service may be accessed from many users via different composite Web services. For this the reason the behaviors of a composite Web service can change with the runtime environment.

The loose grouping of elemental Web services can lead to execution failure of a composite Web service due to changes of the behavior of
elemental Web services. Suppose a composite Web service is concurrently accessed by many users, and one of its elemental Web services has limit in terms of the number of invocations in a certain period of time. If the number of invocations exceeds the limit, the provider of the elemental Web service blocks further access. This results in failure of all running instances of the composite Web service.

We can avoid failures such as this by coordinating the Web services. One solution is to replace a limit-sensitive elemental Web service with a functionally equivalent alternative before the number of invocations exceeds the limit. We can implement this solution by counting and sharing the number of invocations among the instances of the composite Web service and changing elemental Web services as necessary. Generally speaking, the problems that can be triggered by the runtime environment are extremely diverse. This prevents the designer of a composite Web service from adding functions that can avoid the problems.

The users of a composite Web service may know details of the runtime environment, but they also have difficulty in coordinating Web services to suit the runtime environment for the following reasons.

**No authority to modify composite Web service**  In the open environment, the user generally does not belong to the organization that created the composite Web service that he/she is accessing. Therefore, the user may not be authorized to modify the composite Web service under terms set to protect the intellectual rights of the creator of the composite Web service.

**Load of adding processes for coordination**  Even if a user is authorized to modify a composite Web service, the user has to add processes to realize coordination. This burden is too heavy for most users.

Our solution is the concept of Web service coordination; instances of a composite Web service are monitored and controlled from outside
the composite Web service. This does not demand modification of the original composite Web service. We call this concept “Service Supervision”. Service Supervision makes the following three points possible.

First, Service Supervision makes it possible to separate the control processes needed for coordination from the composite Web service. This allows the user to describe only those processes needed for the coordination demanded by the user without modifying the composite Web service.

Next, Service Supervision allows a user to create a general control pattern for coordination. Once created, the control pattern can be used against other composite Web services. This reduces the load of describing control processes for coordination.

Finally, Service Supervision makes it possible to simulate the execution of a composite Web service. Service Supervision controls the execution of the composite Web service. Thus it allows users to execute a composite Web service without invoking the elemental Web services. This is useful because the user can inspect the behavior of the composite Web service in any runtime environment desired without paying the cost incurred by invoking the elemental Web services.

This paper aims at realizing the first two points described above.

Some previous works also aim at changing the behavior of a composite Web service at runtime by extending a standard language or execution engine for composite Web services. For example, Language Grid[Ishida 06] provides dynamic binding, which allows a user to specify endpoints (addresses for accessing Web services) when invoking the composite Web service. In this work, a composite Web service is designed based on only the interfaces of the elemental Web services. AO4BPEL[Charfi 07] and Dynamo[Baresi 07] allow a user to add processes at certain points in a composite Web service based on the concept of AOP (aspect-oriented programming). This technique is not enough for realizing the monitoring and controlling required for Service Supervision.

In this paper, we combine the following two approaches to realize
Service Supervision. First we introduce meta-level control of a composite Web service, which monitors and changes the state of execution of a composite Web service. We can change the behavior of an active instance of a composite Web service and collect information for coordination through the meta-level control. Moreover, we define a protocol that defines the interactions among elemental/composite Web services by adopting choreography described in WS-CDL[wsc 05]. We can control the interactions needed for coordination through choreography.

5.2 Meta-level Control for Coordinating Web Services

Unfortunately, most programs, contents and servers of public Web services in the open environment, like Language Grid, are managed by their original providers as is shown in Fig. 5.1. That is, the provider of a Web service can monitor and control access to his/her service, such as limiting the amount of access and load balancing.

![Figure 5.1: Web services in open environment](image)

In this section, we describe meta-level control of composite Web
services for realizing Service Supervision. The use of meta-level control allows us to change the behavior of a composite Web service in accordance with the runtime environment without modifying the composite service. Moreover, we introduce the technique of control patterns for Web services; once created, a pattern can be applied to other composite Web services.

We first explain the coordination of Web services in the open environment below, and then describe the meta-level control functions that we are proposing.

### 5.2.1 Coordinating Web services in open environment

Several standard languages exist for describing composite Web services, such as WS-BPEL[bpe 03] and OWL-S[owl 04]. We can define the order of Web service invocation and other processes by setting the control constructs provided by the languages. We can also combine Web services by writing programs in languages such as Java. However, we focus here on composite Web services as a business process. Thus we assume composite Web services are described in WS-BPEL, the standard language used to define a composite Web service as a business process. Note that the use of another language, like OWL-S, raises no essential difference.

The languages for Web service composition usually provide some control constructs, such as sequence execution, conditional execution, and looping. They have variables in which the inputs and outputs of Web services and the results of other processes are stored.

Invocations of elemental Web services inside a composite Web service are defined based on WSDL[wsd 01]. A WSDL document defines operations that a Web service provides, messages exchanged between the Web service and its client, and the endpoint address of the Web service.

Figure 5.2 shows an example of a composite Web service. In the
figure, a square containing a circle represents an invocation of a Web service. Arrows represent the order of invocation. A rhombus labeled \texttt{while} represents a control construct for looping. To simplify the figure, we omitted the processes of copying variables between input and output of the Web service, receiving a request, and returning a response from/to the client.

Figure 5.2: Composite Web service for translating documents.

The composite Web service shown in Fig. 5.2 is for document translation. Our goal is to prevent the translation of an extremely long document from exceeding the limit set for the original machine translation Web service. This composite Web service first splits the given document into sentences (\texttt{split}). Next it translates the sentences by the machine translation Web service in the loop (\texttt{translate}). Finally, it merges the results of the machine translation Web service (\texttt{merge}) and returns the translation result of the given document. An instance of the composite Web service is created for each request from the client. The instances are executed independently.

In this example, we assume the string functions \texttt{split} and \texttt{merge} are provided as Web services because the languages used to describe composite Web services do not usually provide embedded string functions.

The definition of a composite Web service and WSDL documents for the Web service, which are constituents of the composite Web service, are deployed on the execution engine for the language describing the composite Web services before execution. The composite Web service is interpreted by the execution engine.

The execution engine creates an instance of the composite Web service when the composite Web service is invoked by other Web service
or a client application and executes the instance.

However, in the open environment, the execution of the document translation composite Web service shown in Fig. 5.2 may fail due to the runtime environment.

One possible point of failure is the limit set by the owner of the machine translation Web service. Assume that each user group that can access the machine translation Web service is assigned an ID, and that the provider of the machine translation Web service limits the number of invocations that each user group can make.

In this case, the number of invocations can exceed the limit. Once the number of invocations exceeds the limit, all instances of the document translation composite Web service invoked by the users in the group fails.

In another case, the provider of the machine translation Web service limits the concurrent accesses to his/her service in order to decrease server load. In this case, service fails when the number of concurrent accesses exceeds the limit.

We can solve these problems if we can modify the document translation composite Web service. One solution to the first case is to switch to a different machine translation Web service when the number of invocations approaches the limit. When the interfaces are standardized and support all machine translation Web services, it is simple to switch to another machine translation Web service just by changing the endpoint.

To realize this solution, we need to modify the document translation composite Web service as shown in Fig. 5.3.

The composite Web service shown in Fig. 5.3 uses an external service which records the number of invocations. Before invoking the machine translation service, the composite Web service invokes the external service to increment the recorded number of invocations (\texttt{count}). This external service returns the number of invocations. When the number of invocation approaches the threshold, the endpoint of the machine translation Web service is changed.
Figure 5.3 shows the control construct *if* for conditional execution as a rhombus. The step for changing endpoints is represented as a square which does not contain a circle because this step is involves copying variables, not as service invocation.

### 5.2.2 Meta-level control of composite Web services

As we described in Section 5.1, it is often difficult to create the composite Web service shown in Fig. 5.3 due to the protection of the intellectual rights of the composite Web service designer and the cost of revising the existing composite Web service.

In this paper, we propose functions for controlling the execution state of a composite Web service in order to realize Service Supervision. This makes it possible to change the behavior of a composite Web service without changing its definition. Moreover users can create and reuse control patterns for coordination of various composite Web services; this reduces the overhead of describing processes for each coordination.

We provide the following functions for controlling the execution state of a composite Web service.

- Step execution
- Suspend and resume execution of a composite Web service
- Get and set variables
- Get and set endpoints of elemental Web services
- Get and set execution point in a composite Web service
- Set a breakpoint
- Set a Web service which is invoked when execution stops at a breakpoint (callback Web service)

These functions control the execution of a composite Web service. Thus we call these functions meta-level control functions. The functions are implemented by extending the execution engine and called when executing a composite Web service.

We also provide an interface to access these meta-level control functions as Web services. Therefore, the functions can be invoked from any platform from which we can invoke a composite Web service.

We use these meta-level control functions to allow one composite Web service to control the execution of another composite Web service. This allows a user to describe a composite Web service that can control execution of another composite Web service. We refer to the former as the supervision composite Web service.

For example, we show a supervision composite Web service for the document translation composite Web service in Fig. 5.4. This supervision composite Web service counts the number of invocations of the machine translation Web service and changes the endpoints to another machine translation Web service when needed.

Before invoking the document translation composite Web service, the supervision composite Web service is invoked. First the supervision composite Web service sets a breakpoint (setBP) before the invocation of the machine translation Web service translate in the document translation composite Web service. It also sets invocation of count in the supervision composite Web service as the callback Web service for the breakpoint.

Next, the supervision composite Web service stops at count (receive) in the figure waiting for an invocation from the document
translation composite Web service. When the document translation composite Web service is invoked and stops before translate according to the breakpoint set by the supervision composite Web service, count is invoked as a callback Web service.

The execution of the supervision composite Web service is then resumed. The supervision composite Web service increments the recorded number of invocations of the machine translation service (increment). The number of invocations is recorded as a variable in the supervision composite Web service.

If the number of invocations of the machine translation Web service exceeds the limit, the endpoint of the machine translation Web service is changed (setEndpoint). Different from Fig. 5.3, setEndpoint in Fig. 5.4 is an invocation of Web service, which is one of the meta-level control functions.

The above steps are repeated according to the control construct for looping until all sentences of the given document have been translated.

The invocation of count from the document translation composite Web service is not a step included in the definition of the document translation composite Web service. This is virtually realized by the
meta-level function that sets a breakpoint and a callback Web service. Therefore, this invocation is represented by a square and a circle with dashed line in Fig. 5.4. In the figure, a dashed arrow represents control of or setting of the state of execution by the meta-level control functions. A solid arrow is an invocation virtually set by the meta-level control functions.

The supervision composite Web service shown in Fig. 5.4 can be applied to various composite Web services that have some limit on the number of invocations. The configurations that depend on the composite Web service to be controlled are locations of breakpoints and the location of Web service invocation whose endpoint should be changed. These configurations can be parameters of the supervision composite Web services. Therefore, we can specify the configurations before invocating the supervision composite Web service and do not have to embed the configurations.

This leads to the reuse of a supervision composite Web service as a general control pattern for coordination. Assuming that several supervision composite Web services are available, all the user has to do is choose the appropriate service and set its parameters. This reduces the user’s overhead in describing processes for coordination.

5.3 Coordination Control Based on Choreography

In this Section, we show the method for coordinating multiple instances of composite Web service. This realizes the information sharing between the instances and timing of the internal processes.

5.3.1 Choreography for Meta-level Control

The control using the supervision composite Web service shown in Fig. 5.4 may fail due to the timing of invocation from the document
translation composite Web service.

Assume that two instances try to invoke count in the supervision composite Web service at almost the same time. The supervision composite Web service receives the request for count that arrives first and starts to increment the number of invocations. If the supervision composite Web service receives the request from another instance while incrementing the number of invocations, the invocation of count fails because the supervision composite Web service is not waiting for the request at count (receive).

In this example, the invocation of a machine translation Web service takes much longer than increment of the number of invocation. This is the reason the control works if only one instance of the document translation composite Web service is running. However, the control may fail if there are multiple instances of the document translation composite Web service which is to be controlled by one instance of the supervision composite Web service.

To solve this problem, we introduce choreography, which defines the protocol of interactions between a supervision composite Web service and the composite Web service being controlled.

Choreography in the Web service area defines a protocol of interaction between Web services. A standard language for describing choreography, WS-CDL (Web Service Choreography Description Language)[wsc 05], has been established.

Choreography is normally used to define a protocol between composite Web services produced by different organizations. Each organization can ensure that its composite Web service correctly works with those produced by other organizations by sharing a choreography and designing its composite Web service according to the choreography.

WS-CDL provides the following elements as constituents of a choreography.

**sequence, parallel, choice** Control constructs which fines the order of execution of activities defined in a choreography.
interaction activity Message exchange among participants. Different from BPEL, WS-CDL allows the definition of message exchange for distributed systems which do not have a central view.

workunit activity Guarded activity which is executed only if a condition is satisfied. This activity allows interactions to wait for arrival of required data.

perform activity Import and execute choreographies defined in other part.

assign activity Copy data stored in variables.

We show an example of choreography in Fig. 5.5, which defines the supervision composite Web service and the document translation composite Web service shown in Fig. 5.4. The repetition is represented as conditions of the workunit.

We show an example of choreography in Fig. 5.5, which defines the supervision composite Web service and the document translation composite Web service shown in Fig. 5.4.

A square which contains arrows inside represents an interaction between Web services. The choreography shown in Fig. 5.5 defines the order of count (receive) and count (invoke) which are invoked by the supervision Web service and the document translation composite Web service respectively.

This definition prevents the failure which is caused by invocation of count during the execution of increment.
The choreography shown in Fig. 5.5 can be applied to various composite Web services containing a Web service whose invocation is limited to a certain number. Therefore we can reuse the choreography as well as the supervision composite Web service in Fig. 5.4.

### 5.3.2 Architecture

We describe the process and the architecture for controlling the interactions between a supervision composite Web service and the composite Web service to be controlled according to the given choreography.

After a supervision composite Web service and the composite Web service to be controlled are invoked, all interactions among the composite Web services and Web services invoked by them are monitored, and checked if the sequence of the interactions is accepted by the given choreography. If the choreography does not accept the sequence, we change the order of the interactions.

To realize the above, we implemented the architecture shown in Fig. 5.6.

![Figure 5.6: The architecture for realizing Service Supervision.](image)

The architecture shown in Fig. 5.6 consists of two parts: Composite Web service execution engine and protocol control engine.
When the composite Web service execution engine receives a request for invoking a composite Web service, the engine creates an instance of the requested service. If the invoked service requires some control, the appropriate supervision composite Web service must be invoked beforehand.

The supervision composite Web service and the composite Web service to be controlled interact with each other at breakpoints and callback Web services set by the supervision composite Web service via meta-level control functions.

The supervision coordinator in the protocol control engine monitors all the interactions defined in the given choreography. To process the interactions according to the choreography, the supervision coordinator performs the following steps when it observes that an interaction is going to occur:

1. Before the interaction is processed, put the information of the interaction into a queue which records the set of information of interactions.

2. Check the queue and process the interactions in the queue acceptable by the given choreography.

3. Remove the information of the interactions that have been processed from the queue.

If any sequence that consists of interactions in the queue is not accepted by the given choreography, execution of both the supervision composite Web service and the composite Web service to be controlled are halted and the control based on the choreography fails.

To check if an interaction can be processed under the context of the previous interactions, we can use a simulator of the interpreter of WS-CDL. To obtain the information of interactions and to process the interactions that are accepted by the given choreography, we use the meta-level control functions.
5.4 Discussion

In this section, we show examples of applying Service Supervision in order to discuss its applicability and the performance. We then consider some of the features of Service Supervision.

We take the document translation composite Web service as an example and apply the following two controls.

- Assume the provider of the machine translation Web service limits the total number of invocations within some period. When the number of invocations exceeds the limit, change the endpoint of the machine translation Web service to select another machine translation Web service. (Control 1)

- Assume the provider of the machine translation Web service limits the number of concurrent accesses. When the number of concurrent accesses exceeds the limit, change the endpoint to select another machine translation Web service. When the number of concurrent accesses falls under the limit, change the endpoint back to that of the initial service. (Control 2)

To implement Control 1, we used composite Web services shown in Fig. 5.4 and the choreography shown in Fig. 5.5.

To implement Control 2, we designed a supervision composite Web service that records the number of concurrent accesses to the machine translation service. This supervision composite Web service sets breakpoints before and after the invocation of the machine translation Web service in the document translation composite Web service. The callback Web services set for the breakpoints invoke the supervision composite Web service.

The supervision composite Web service waits for the invocation of the two kinds of callback Web services set before and after invocation of the machine translation Web service. It increments the recorded number of concurrent access at the first invocation, and then decrements the
number at the second invocation. The choreography is similar to one shown in Fig. 6, but it selectively processes one of the two interactions that occur before or after invoking machine translation Web service.

The supervision composite Web service waits for the invocation of the callback Web services. It increments the recorded number of concurrent access at the first invocation, and then decrements the number at the second invocation. The choreography is similar to one shown in Fig. 5.5. But it selectively processes one of the two interactions which occur before or after invoking machine translation Web service.

First we inspected the execution time of the document translation composite Web service for each control in order to show that our method is practical. The input to the document translation composite Web service is a document consisting of 5 Japanese sentences to be translated into English. The limits placed on the total number of invocations and the concurrent access number limit are both 3.

Figure 5.7 shows the results. In this experiment, we compared the execution time of the control based on Service Supervision with the execution time of the equivalent control realized by modifying the document translation composite Web service like Fig. 5.3.

Moreover, we inspected the execution time of 1 - 7 instances of the document translation composite Web service. The time shown in Fig. 5.7 represents the average of 10 trials. The time for more than 1 instance is the average per instance.

The result shows that Service Supervision yields faster control than modifying the composite Web service in the case of 1 instance. This is because the interactions between composite Web services in WS-BPEL take longer than the access to information of other composite Web service via the meta-level control functions.

In the case of more than 3 instances, modifying the composite Web service yields faster control than Service Supervision. This is because the queue of the interactions which are not processed tends to have relatively many interactions and it takes time to check if the interactions can be processed or not by the WS-CDL interpreter.
The results show that our method has a disadvantage in scalability of the number of processes to be controlled. But the performance does not decline much for the small number of instances.

Next we compared the control complexity of supervision composite Web services to that of modified composite Web services in order to assess the cost of applying Service Supervision. We used the number of activities (atomic process step in a composite Web service including service invocation, copying variables, and interactions with other composite Web service) and containers of control constructs, and maximum depth of nested structures as metrics. To choose the metrics, we consulted a previous work on business processes that addressed such metrics [Gruhn 06].

Table 5.1 shows the complexity comparison results. For the supervision composite Web services, the sum of the numbers of activities and containers in the supervision composite Web services and the document translation composite Web service to be controlled in brackets because they are used together.

When using Service Supervision, the sum of the number of activities and containers exceeds that of the document translation composite Web service modified to realize the same controls. One reason for this is that the supervision composite Web services have activities for interacting with the client and the composite Web service to be controlled. Another
Table 5.1: Complexity of composite Web services.

<table>
<thead>
<tr>
<th></th>
<th>Activities</th>
<th>Containers</th>
<th>Nest</th>
</tr>
</thead>
<tbody>
<tr>
<td>No control</td>
<td>9</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Service Supervision (Control 1)</td>
<td>8 (17)</td>
<td>5 (7)</td>
<td>4</td>
</tr>
<tr>
<td>Modifying service (Control 1)</td>
<td>12</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Service Supervision (Control 2)</td>
<td>10 (19)</td>
<td>6 (8)</td>
<td>5</td>
</tr>
<tr>
<td>Modifying service (Control 2)</td>
<td>15</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

reason is that some of the processes that require a control construct are separated into the supervision composite service and the composite Web service to be controlled. However, the difference is not significant because the difference in numbers of activities and containers does not increase with the complexity of the supervision composite Web service and composite Web services.

The supervision composite Web services for control 1 and 2 described above can be applied to various composite Web services that use a Web service with a limit on the number of invocations. The choreography for the supervision composite Web services can also be applied to such composite Web services. Note that not all supervision services can be applied as a general control pattern for coordination.

As we mentioned in Section 5.3, the major reason for failure of a reuse of a supervision composite Web service is conflict between the processes by the supervision composite Web services and composite Web services to be controlled. When timing of changing variables and endpoints varies depending on composite Web services to be controlled, it is difficult to define common control patterns for coordination.

Multiple supervision composite Web services can be also applied to a composite Web service at the same time. This is especially useful for absorbing the differences in the behaviors of Web services when the endpoint of a composite Web service is changed during execution.

Suppose the endpoint of the machine translation Web service in the
document translation composite Web service is changed, and that the character set of the new machine translation Web service is different from that of the previous one. This may cause failure of subsequent Web services. In this case, we can add a post process by the supervision composite Web service which converts the character set of output of the machine translation Web service. This feature is effective in keeping the modularity of supervision composite Web services.

5.5 Related Works

Web services require to be monitored because they often show unexpected behaviors. The importance of the monitoring is pointed by previous works such as [Mahbub 05, Barbon 06]. But not many works have addressed the issue which is about controlling the behaviors of Web services, not only monitoring.

Several researches have tried to change behaviors of a composite Web service without modifying the composite Web service. Most of the works have adopted the concept of AOP (Aspect-oriented Programming).

Mosincat et al. proposed a method for dynamic service selection and error recovery[Mosincat 08]. Their method transforms a BPEL process transparently to both the designer of the BPEL process and the execution engine.

Moser et al. addressed that BPEL lacks functionalities for dynamic adaptation and monitoring[Moser 08]. They proposed a method for dynamic service selection and managing QoS information by intercepting SOAP messages.

Dynamo[Baresi 07] is also based on the concept of AOP. It monitors the messages exchanged between a BPEL process and external Web services and checks if the messages satisfy constraints.

The works mentioned above introduce their own languages or non-standard languages for defining the controls of composite Web services.
However, there are already many systems which are working based on the current standard specifications. This may be a problem when we adopt the methods proposed in the previous works. Our framework proposed in this paper uses standard languages such as WS-BPEL and WS-CDL. Therefore we can exploit the existing tools like GUI editors and expertise of SOA engineers. On the other hand, our work requires modification of execution engine. This can be a disadvantage over the approach[Mosincat 08] to the execution engine.

AO4BPEL[Charfi 07] introduced Aspect-Oriented Programming (AOP) into BPEL. It allows the composite service designer to insert any process before/after an activity specified as a pointcut. Inserted processes are also described in BPEL. A pointcut is specified by XPath.

This work allows us to define controls as BPEL processes. But it only adds a process into an existing BPEL process and cannot realize some controls such as stopping/restarting execution of a composite Web service which the meta-level control can perform.

Program Supervision[Thonnat 92, Nethercote 03] tries to compose legacy program components, not Web services. It automatically finds a plan that can appropriately process the given data according to requirements at runtime by combining program components.

Adaptive Workflow[Aalst 99] focuses on workflows which are mainly executed by humans. It aims at adapt to unexpected exceptions and changes in the environment. Some previous works adopt case-based reasoning, rule-based systems, and planning to realize the adaptation [Weber 04, Casati 00]. However, they considered neither the policies of stakeholders nor interaction between Web services and human tasks.

The above two works aim at adapting programs or workflow to a given requirement or an environment as our proposal does. However, neither considers the coordination of multiple programs or workflows.
5.6 Summary

The behavior of a public Web service in the open environment can change with changes in the runtime environment. Users must, therefore, coordinate Web services when they use a composite Web service that consists of elemental Web services in the open environment. However, in the open environment, users of a composite Web service are generally not authorized to modify the composite Web service, and the overhead of adding processes required for coordinating Web services can be too heavy for the users.

This paper proposed the concept of Service Supervision, which monitors and controls active composite Web services in order to coordinate Web services in the open environment.

The contributions of our work are as follows:

- We made it possible for users to change the behavior of a composite Web service for coordinating the elemental Web services in the open environment without modifying the composite Web service by using meta-level control.

- We reduced the overhead of describing processes for coordinating elemental Web services in the open environment by allowing users to apply control patterns for coordination to various composite Web services.

We also showed that our work achieves Web service coordination without a significant performance penalty or making composite Web services too complex.

In future work, we will extend the mechanism proposed in this paper and coordinate composite Web services based on the policies of various stakeholders such as users and service providers.
Chapter 6

Conclusion

In this research, we aim at constructing language services from language resources on the internet and supervising execution of composite services which consist of them. In this chapter, we first describe the contributions of our research and then conclude this thesis by showing the future works.

6.1 Contributions

We describe contributions of each work included in this thesis below.

1) Simplifying process of language service construction

In this thesis, we first proposed a method for constructing language services from language resources on the internet such as dictionaries and parallel texts. The contribution of this work is as follows.

Our method makes it possible to extract relations between data described in tables based on given human’s interpretation. This allows us to extract contents required for constructing language services from language resources.

In various fields of intercultural collaboration, they have compiled dictionaries for their own domains. We have observed that the language services constructed from the language resources improve the quality of translation by combining them with other Web services such as machine
translator Web services.

However, extracting contents from language resources often costs a lot because we need to manually copy the required contents or describe complicated regular expressions over again and again. This requires the advanced knowledge of regular expression and manually editing parts that regular expression does not cover by people who wants to construct language services.

Although our method is limited to extraction from language resources in tables, it allows users to extract contents required for language services only by giving an example of the required contents.

We assume that tools which support intercultural collaboration will be more useful by collecting language resources from small communities in intercultural collaboration and combine with other language resources using services computing technologies. Our method which allows users to construct language services only by showing examples will be useful for achieving this goal.

(2) Reducing cost of wasteful service execution

We also proposed a method for predicting connectability between Web services which are provided by various organizations.

In our work, the proposed method predicts execution failure of a composite Web service caused when an atomic Web service outputs a response message which is not valid for another atomic Web service as an input. In this case the cost of execution of atomic services which have been already executed goes to waste. Therefore we can reduce the wasteful execution of the atomic Web services by cancelling the execution of the composite Web service when we can predict such execution failure.

The new business models based on SaaS (Software as a Service) or cloud computing have rapidly spread and some Web services require monetary costs. Some Web services available on the Language Grid also limit the amount of traffic during a certain period of time.

If execution of a composite Web service which consists of such Web services failed due to failure of any of atomic Web service in the com-
posite service, the user would be put at a disadvantage. But it is diffi-
cult for him/her to ask service providers to modify their Web services
because the Web services are correctly working.

Although the number of available Web services has been increasing,
interfaces, specifications and usage of the Web services depend on the
service provider and we don’t still have the framework for coordinating
the heterogeneous Web services. In such environment, the method
proposed in our research will work well for reducing the cost of wasteful
execution.

(3) Providing execution control framework for open environ-
ment

In this work, we proposed a method for modifying the behavior of
a composite Web service without changing the composite Web service
itself. Our method is based on the concept Service Supervision, which
controls execution of composite Web services.

The model of a composite Web service is defined in a language such
as WS-BPEL, which describes control flows and dataflows. However,
in open environment where new Web services become available day by
day and the status of the Web services is always changing, the existing
frameworks for the composite Web services is not enough.

Some previous works assume that we have Web services which
are functionally equivalent but have different non-functional properties
such as throughput, response time and quality of the contents. They
tried to improve the QoS of the composite Web services by appropri-
ately configuring them[Menasce 02, Menasce 04]. But it is difficult to
implement such QoS control on the existing frameworks as well as our
method for predicting connectability between Web services. Thus we
have to implement them by modifying a composite Web service to re-
alize the controls.

However it costs a lot to modify composite Web services to imple-
ment such controls due to frequent changes of open environment. Ser-
vice Supervision proposed in our work provides a general framework in
which various control components can be flexibly implemented.
6.2 Future Directions

The works described in this thesis show basic ideas and possibility for constructing language services from language resources and coordinating them. Thus we need some more works shown below in future in order to use the result of our works in real world, such as on the Language Grid.

Mixed-initiative framework for language service construction

Our work proposed a method for constructing language services from language resources on the Web based on examples given by humans. But our algorithm sometimes has difficulty in generalizing structures of tables based on the given example. Thus we need a mixed-initiative framework which interleaves human’s interpretation and correction by giving new examples and generalization.

Learning input/output relation

Our method for predicting connectability of Web services needs input specifications and input/output relations in order to be applied. Although input specifications can be acquired by using constraint acquisition algorithm, input/output relation must be defined by users. This is one of the major issues of our work. Thus we also need a method for learning internal models of the atomic Web services based on the behavior of the Web services.

Solving conflicts between controls

Meta-level control functions and coordination based on choreography allow us to describe a composite Web service called a supervision composite Web service, which controls other composite Web services. In open environment, various stakeholders including users and service providers need to apply the supervision composite Web services. But multiple supervision composite Web services may cause conflicts when those which are described by different stakeholders are applied at the same time. This is the reason we need to introduce
a coordination mechanism of controls and allow stakeholders to
describe control based on their policies.

We need a mechanism which gives a energy to the cycle of con-
struction and use of language services in order to satisfy a wide variety
of requirements in the field of intercultural collaboration. Supporting
intercultural collaboration based on services computing will spread via
constructing language services and use of tools which invoke the ser-
vices. This leads to further participation of various communities and
construction of new language services.

Therefore our work will make sense by finally implementing the re-
sults in the practical systems. The Language Grid already has many
users and services which wrap language resources collected from vari-
ous organizations. Thus we will work on organizing the results so far
and show the contributions of our work via empirical activities on the
Language Grid.
Bibliography


[Chen 00] Chen, H., Tsai, S., and Tsai, J.: Mining Tables from Large Scale HTML Texts, in 18th International Conference Computational Linguistics, pp. 166–172 (2000)


[Embley 02] Embley, D., Tao, C., and Liddle, S.: Automatically Extracting Ontologically Specified Data from HTML Tables with Un-


104


105


Publications

Major Publications

Journals


International Conferences


3. Masahiro Tanaka and Toru Ishida. Ontology Extraction from
Other Publications

International Conferences


Conventions
