Master Thesis

Development Support of Intercultural Collaboration Systems Based on Service Oriented Programming

Supervisor Professor Toru Ishida

Department of Social Informatics
Graduate School of Informatics
Kyoto University

Satoshi SAKAI

February 5, 2009

Development Support of Intercultural Collaboration Systems Based on Service Oriented Programming

Satoshi SAKAI

Abstract

Recently, various types of intercultural collaboration must be performed in the field of education, medical care and disaster prevention due to internationalization and growth of the Internet. However, it is difficult for participants who have various mother tongues or cultures to collaborate. Therefore, the need for intercultural collaboration systems has increased. In addition, the system should be customized for specific domains or end users. However, since there are many organizations which need support, it is impossible to construct custom-made systems for all organizations because of high cost. On the other hand, the technology of the web service, which can be invoked by standardized procedure, has developed. In addition, composing web services makes it possible to make functions which satisfy users' demand.

Thus, combining several web services becomes possible to construct customized intercultural collaboration systems. However, there are following two problems to construct systems by using diverse web services.

Difficulty of programming using web services Recently, in order to combine several web services, Business Process Execution Language (BPEL) is used. However, developers should be knowledgeable about web service to use this technique. Moreover, designing flows of web services and data requires professional skills peculiar to BPEL. Therefore, it is difficult for general developers such as members of NPOs or university students to combine several web services by using BPEL.

Low reusability of components of intercultural collaboration systems Creating a new component using web services and combining them make it possible to construct a customized system. However, if these components are created ad hoc, it becomes difficult to reuse them because they contains the structures and constants peculiar to specific domains or organizations. In addition, when many components are accumulated, it is difficult to discover components in need. Therefore, reusability of components becomes low and construction of

a customized system need high cost.

For the first problem, this research proposes a method to create light weight building blocks with service-oriented programming and a technique to construct systems by using these blocks. A building block is created as a service by using a general programming language such as PHP. Developers can create a new one easily by combining them. In addition, they can construct an intercultural collaboration system by building up several blocks and creating a graphical user interface. By using this approach, even developers who have little knowledge about web service can construct a system.

For the second problem, this research proposes a method to organize components by classifying the language services into four layers. Because of a constraint that a component can invoke other components only in the same layer or the lower layer, services customized for user domains or end users are accumulated in the upper layer and general-purpose services are accumulated in the lower layer. Thus developers can search available components in the layer beginning at the top. If there is no available component, they can create a new component by using components in the lower layer. Therefore, reusability of components becomes high.

In this research, we have developed the Language Grid Playground, an collection of intercultural collaboration systems, as a practice of our approaches. And we have published source codes of the systems and held workshops several times for developers such as university students as a practice of reusing building blocks. The contributions of this research are as follows.

Developing service-oriented building blocks General developers can construct a customized system for intercultural collaboration by using service-oriented building blocks. In fact we created 31 building blocks and constructed 14 systems in the Language Grid Playground.

Realizing layered architecture of language services Developers can find building blocks easily by classifying them into the four layers. Moreover reuse of building blocks makes it possible to reduce the cost of a system construction. In fact, we can bring several procedures together and reduce the code quantity by using classified building blocks.

サービス指向プログラミングに基づく
異文化コラボレーションシステムの開発支援

境 智史

内容梗概

現在，国際化やインターネットの普及により，教育や医療，防災など様々な分野で異文化コラボレーションがおこなわれている．しかし，言語や文化が異なる参加者同士がコラボレーションを行うことは難しく，支援システムの必要性が高まっている．異文化コラボレーションシステムは利用ドメインや利用ユーザにカスタマイズされたシステムでなくてはならない．しかし，世界中には支援を必要としている団体が数多く存在し，すべてに対してカスタマイズしたシステムを構築するには非常に大きなコストがかかってしまう．一方，標準化された手続きによって呼び出すことが可能なWebサービスという技術が発展してきており，それらを組み合わせることでユーザの多様な要求にこたえる機能を構築することができるようになってきた．

そこで，複数のWebサービスを組み合わせることで，カスタマイズした異文化コラボレーションシステムの構築が可能になると考えられる．しかし，多様なWebサービスを用いてシステムを構築するには次の課題が存在する．

Webサービスを用いたプログラミングの困難さ　現在，Webサービスを組み合わせる手法としてビジネスプロセス記述言語を用いる手法が提案されている．しかし，この手法を用いるにはWebサービスに関する知識が必要となる．さらに，実行するWebサービスフローやデータフローを構築する際にはビジネスプロセス記述言語特有のプログラミング方法の知識がなくてはならず，一般的な開発者やNPOのメンバー，学生などがそれらの言語を用いてWebサービスを組み合わせることは非常に困難である．

構築したコンポーネントの再利用性の低さ　Webサービスを用いて新しいコンポーネントを構築していき，それらを組み合わせることでカスタマイズしたシステムを構築することが可能となる．しかし，それらのコンポーネントをアドホックに構築してしまうと，利用ドメインやユーザ固有のWebサービス呼び出しや処理などがコンポーネントに含まれることにより他のシステムから利用することが困難となる．また，多くのコンポーネントが蓄積された際に，利用可能なコンポーネントの発見が難しい．よってコンポーネントの再利用性が下が

り，多くのカスタマイズシステムの構築に大きなコストが必要となってしまう．

前者の課題に対して，本研究ではサービス指向プログラミングを用いて軽量なビルディングブロックを構築し，それらを用いてシステムを構築する手法を提案する．PHP などの一般的なプログラミング言語を利用して一つのサービスを提供するビルディングブロックを構築していくことで，開発者はそれらを組み合わせて新たなビルディングブロックを容易に構築することが可能となる．また，それらを積み上げていき，システムの GUI を構築することで異文化コラボレーションシステムを構築することができる．この手法を用いることで，Web サービスに関する知識のない開発者でも容易にシステムを開発することが可能となる．

後者の課題に対して，本研究では言語サービスを 4 つのレイヤに分割し，整理を行う手法を提案する．各レイヤに同レイヤもしくは下位レイヤしか呼び出せないという制限を設けることで，上位レイヤにはドメインやユーザに特化したサービスを蓄積することができ，下位レイヤには汎用的なサービスを蓄積することが可能となる．これにより，開発者はまず，上位レイヤから順に利用可能なコンポーネントを検索し，もし利用可能なコンポーネントが存在しなければ，下位レイヤのコンポーネントを利用して新たなコンポーネントを構築することができ，コンポーネントの再利用性を向上させることができる．

さらに本研究では，上記のアプローチを実現する異文化コラボレーションシステム “言語グリッドプレイグラウンド”の構築を行うとともに，ビルディングブロック再利用の実践として，ソースコードの公開や学生の開発者への講習を行った．本研究の主な貢献は以下の 2 点である．

サービス指向ビルディングブロックの開発　ビルディングブロックを用いることで，一般的な開発者でもカスタマイズした異文化コラボレーションシステムを構築することが可能となった．実際にプレイグラウンドでは 31 個のビルディングブロックを構築し，14 個のシステムを構築した．

言語サービス多層アーキテクチャの実現　言語サービスを 4 つの階層に分類し，整理を行うことでビルディングブロックを容易に発見することが可能となり，発見したビルディングブロックを再利用することでシステム構築のコストを下げることが可能となる．実際にプレイグラウンドで構築したビルディングブロックを 4 階層に分類することで，処理の共通化を行うことができ，ステップ数を削減することができた．

Development Support of Intercultural Collaboration Systems Based on Service Oriented Programming

Contents

Chapter 1 Introduction

In recent years, the opportunities for international exchange and the number of multicultural communities have increased due to internationalization and the growth of the Internet. For example, there are collaborations such as health interviews for foreign patients in hospitals, guidance for foreign students, and communications with foreign parents in the field of education. In the field of education, a system with which users can communicate by reusing existing glossaries of educational terms or a system with which users can make handouts easily by using a machine translator are needed. Moreover, since an incorrect translation of a technical term makes serious situations in the field of medical care, systems should translate medical terms correctly. In fact, accumulated parallel text based support system for intercultural communication at medical receptions [3] has developed. Thus in intercultural collaboration, systems should be customized for the tasks in each field. However, systems available on portal sites provide only general functions such as translation and dictionary services. In other words, these systems cannot be customized for particular fields and consequently cannot solve collaboration problems in these fields.

On the other hand, the technology of the web service, which can be invoked by standardized procedure, has developed. In addition, it becomes possible to construct systems which satisfy users' diverse demand by combining several web services.

Therefore, combining several web services enables to construct intercultural collaboration systems customized for a specific organization. However, there are following two issues to construct systems by using web services.

- Difficulty of programming with web services

  As a method to combine several web services, workflow description languages such as business process execution language (BPEL) are developed. However, in order to construct workflows with these languages, professional knowledge about web service and the description language are needed. Thus, it needs high cost for general developers who have little knowledge about them to construct a system. Moreover, in many cases, organiza-

tions which need support for intercultural collaboration are lacking money and human resources. Therefore, architecture for general developers to construct a system easily with web services is needed.

- Low reusability of components for intercultural collaboration systems
  In the case of constructing general systems, it is possible to reduce the cost of construction by reusing components which are created in other system. Meanwhile, in the case of constructing intercultural collaboration systems, if components are constructed ad hoc, it is difficult to reuse them in other systems since they contains structures or constants peculiar to specific domains or end users. Moreover, when many components are accumulated, finding available ones is difficult. Then, since reusability of components becomes low, constructing intercultural collaboration systems needs high cost.

In this research, in order to solve these issues, we take following two approaches.

- Development of building blocks realizing service-oriented programming
  In order to solve the former issue, this research proposes a method of creating light weight building blocks realizing service-oriented programming and a technique of constructing intercultural collaboration systems by combining them. By creating a building block as single service, developers can create a new composite one easily. Moreover, they can construct systems easily by building up these blocks and creating a graphical user interface on them. In addition, since building blocks can be described in general programming language such as PHP or Java, developers who have little knowledge about workflow description language can construct systems with web services.
- Classification of language services into four layers
  In order to the latter issue, this research proposes an approach to organize language services by classifying them into four layers: resource adaptation layer, combination layer, application layer and user adaptation layer. This layered architecture has a constraint that a service in the upper layer can invoke another service in the same layer or the lower layer but a service

in the lower layer cannot invoke a service in the upper layer. Therefore, building blocks for general-purpose are accumulated in the lower layer and building blocks for limited-purpose are accumulated in the upper layer. Then, system developers can find necessary building blocks customized for specific domains or organizations. If there is no available building block, developers can create a new one easily by combining the blocks in the same or the lower layer.

In this research, we have developed the Language Grid Playground, a collection of intercultural collaboration systems, as a practice of our approaches by using the Language Grid as a group of web services and evaluated the approaches.

This paper is organized as follows: Chapter 2 describes an environment of intercultural collaboration as a background and introduces the Language Grid project. In chapter 3, we propose the method of system construction with building blocks as a first approach. Chapter 4 describes layered architecture of language services as a second approach. Chapter 5 describes an implementation of the Language Grid Playground. In chapter 6, we evaluate the cost of system construction and show the best practices of reusing building blocks. Finally, chapter 7 describes conclusion.

Chapter 2 Background

In this chapter, we introduce the current situations of intercultural collaboration activities and the Language Grid project. The Language Grid is an infrastructure for enabling users to create new language services by combining web services that represent wrapped language resources.

2.1 Intercultural Collaboration

Recently, intercultural collaborations have increased due to internationalization and the growth of the Internet. For example, there are medical inquiries for foreign patients in hospitals. Nowadays, many foreigners came to Japan to work and live there. However, since they are not able to understand Japanese well, they cannot have an appropriate medical practice when they catch some disease. Then, in order to support these collaborations, non-profit organizations such as Center for Multicultural Society [1)] send medical translation volunteers to hospitals. Moreover a multilingual communication support system which is called M3 (M cube) are developed for foreign patient who visit the front desk in the hospital [3]. This system can support reception procedures, communication with hospital staffs and navigations in multiple languages. A trial of intercultural collaboration using this system is started. In another example, NPO Pangaea[2)] performs intercultural collaboration activities. This organization's mission is to create a universal playground, where children around the world can play together. In addition, this organization holds web cam events by connecting several bases in Kyoto, Soul, and Vienna and so on. In order to increase the communication with each volunteer staffs who can understand only mother tongue in preparation for the event, the organization developed a community site which support exchange of information in multiple languages by using machine translators.

Thus, many activities of intercultural collaboration have increased in several fields. However, since language barrier and cultural difference are so big that

[1)] NPO Center for Multicultural Society Kyoto http://www.tabunka-kyoto.org/

[2)] NPO Pangaea http://www.pangaean.org/

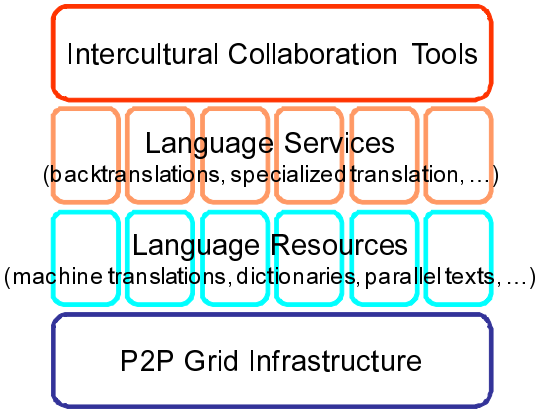

Figure 1: Service Layer of the Language Grid [1]

people cannot perform the activities smoothly. Therefore, many organizations need support for it.

2.2 Language Grid

The Language Grid [1] is an infrastructure for enabling users to create new language services by combining web services that represent wrapped language resources published on the Internet. In addition, the Language Grid Association[7] is organized as a user group to discuss issues about the Language Grid from various perspectives and accumulate knowledge to better utilize it. The Language Grid has two main structures. One, called the horizontal Language Grid, involves the combination of existing bilingual dictionaries or machine translation systems. The other one is called the vertical Language Grid. It involves concerns specific scenes of intercultural collaboration activities, which require new specialized language services. The first one combines language resources and a language processing system for standard languages by using a business process execution language. The second one enables the use of specific community dictionaries and parallel texts used in the field of intercultural collaboration. As shown in Figure 1, the Language Grid consists of four service layers.

- P2P Grid Infrastructure

  This layer allows access to material on multiple servers on the Internet to satisfy end users' request by coordinating the Language Grid core nodes and service nodes [2].

- Language Resources

  This layer provides various language resources as atomic services with a standardized interface such as dictionary services, parallel text services, and machine translation services. A language resource provider can easily add new language resources as web services.
- Language Services

  This layer allows various language services to be created by combining existing language resources for intercultural collaboration such as multi-hop translation, which is created by combining several machine translation services, and domain-specific translation, which is created by combining one or more public dictionary services with a machine translation service. The language services are created as workflows using business process execution language for web services (BPEL4WS) [4]. Thus, language service users can create new language services by themselves and add them.
- Intercultural Collaboration Tools

  The top layer provides intercultural collaboration tools that allow users to utilize the Language Grid, even if they have no programming skills. These tools are developed by combining the aforementioned language services and resources. The Language Grid Playground lies in this layer because its goal is to support intercultural collaboration by using language services and resources.

Chapter 3 Constructing a System Using Building Blocks

Most of general developers have little knowledge about web service. Therefore, it is difficult for them to program a system with several web services. Then, system construction needs big costs. In order to solve this problem, this research proposes a method of creating building blocks which realize the service-oriented programming by using general programming language such as PHP, and a technique to construct systems with web services by combining these blocks. In this chapter we introduce a paradigm of service-oriented programming. We propose how to create building blocks and how to construct systems by using these blocks. We also compare our method with existing method using business process execution language.

3.1 Service-Oriented Programming

Specific organization can be supported by a system which is constructed to satisfy the end user's demand. However, there are so many organizations in need of such systems which support their activities that it is impossible to make customized systems for each of them from scratch because of high cost. In addition, reusing systems is also difficult because functions of such systems are so complicated and specialized for the tasks of specific organizations.

Therefore, we incorporate a paradigm of service-oriented programming [8] in order to construct a system with several components and make them more reusable.

Service-oriented programming is a paradigm to create a component which represents a service and construct a system by using it. This "service" means that it is a group of software which can be invoked from outside by standardized procedure and it has a function which is meaningful to humans by itself. In other words, developers can create a component to represent a service and components can be combined to create a new complex component and, finally an application. In this paradigm, components can be seen as clients in that they are invoked by other components. On the other hand, they can be seen as servers for that

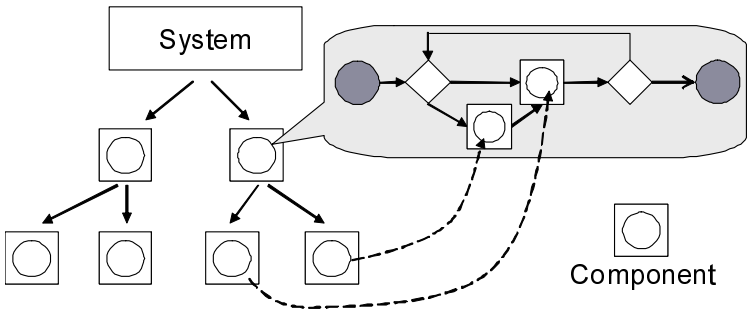

Figure 2: Service-Oriented Programming [8]

they invoke other components. Therefore, it becomes possible to break down a system which performs complex tasks into several components that are organized hierarchically as shown in Figure 2. Moreover accumulated components have appropriate grain size because this paradigm breaks down a procedure of a system hierarchically. In addition developers can understand the functions of components easily because one component provides one service. Therefore, the components are reusable since developers use existing components easily when they construct a new system. Moreover, creating components by breaking down the complex processing of tasks into several services allows the structure of the components to be greatly simplified. In other words, they become light weight.

3.2 Building Blocks

Currently, in order to realize the service-oriented programming, workflow description languages such as BPEL are developed. However, XML representation of BPEL is very verbose and can be read by only trained developers. Although support tools for describing workflows of web service such as BPEL editor are developed, knowledge about web service technology and data flow between several web services and techniques for programming peculiar to BPEL are need to create workflows. Therefore, general developers cannot describe it. Then, in this research, we propose an approach to realize the service-oriented programming by using general programming language such as PHP and Java.

In order to realize the function which is equivalent of integration workflow, a class which has a method that wrapped a web service can be created by using general programming language. Therefore, developers can use the functions of

web services by describing a general method call. Moreover, they can create a class which has a new method realizing a new service by combining these functions. Finally, developers can construct a system by building up the classes as building blocks created in this way. In this approach, it is possible to realize the service-oriented programming by using general programming languages. Figure 3 shows the procedure of creating the building blocks, and details of each step are as follows.

Step1 Search a web service which provides a service satisfying a demand

In order to create a building block, at first, developers search an available web service which realizes the service they want to use. If the web service exists, they can create a building block by describing a class with the method which invokes the web service.

Step2 Break down a service satisfying a demand into several subservices

If the service which developers want to use is not provided as a web service, developers break down the service into several subservices. For example, there is a service such as the back translation. Back translation service is realized by translating sentences from language A to language B and then translating the result from language B to language A again. Therefore, developers can break down the back translation service into two translation services.

Step3 Search the building blocks providing the subservices and combine them

First, developers search the building blocks providing the subservices which they break down the main service into. If all of building blocks exist, they can create a new building block described as a class with a method realizing the service by combining the blocks. If some of the building blocks providing subservices don't exist, they create several new building blocks and then, if all of the blocks are prepared, they can creates a new one.

Developers can construct a new intercultural collaboration system easily by building up the blocks created in this approach. However, there is a big cost to construct customized systems for all organizations which need support in the world. Therefore, we publish the building blocks used in our systems so that general developers can reuse them and create new ones easily.

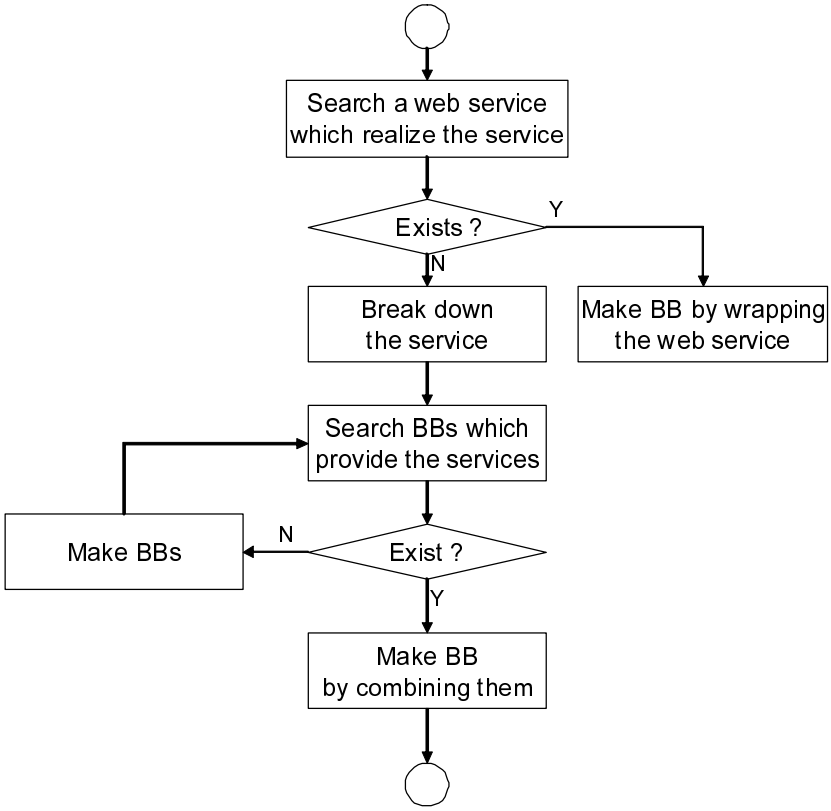

Figure 3: Procedure of Creating Building Blocks

3.3 Construction Method of Intercultural Collaboration System

Developers can construct a customized intercultural collaboration system by combining building blocks created in foregoing approach. Figure 4 shows the procedure of constructing the systems, and details of each step are as follows.

Step1 Define a function provided in the system

First of all, developers have to define a function which will be provided in the new system. In the case of intercultural collaboration systems, since the function should be customized to support specific domains or organizations, system developers have to build consensus with end users and then define a function.

Step2-1 Create a user interface

After the function is defined, developers discuss about how end users use the function and then construct a user interface. Since user interfaces

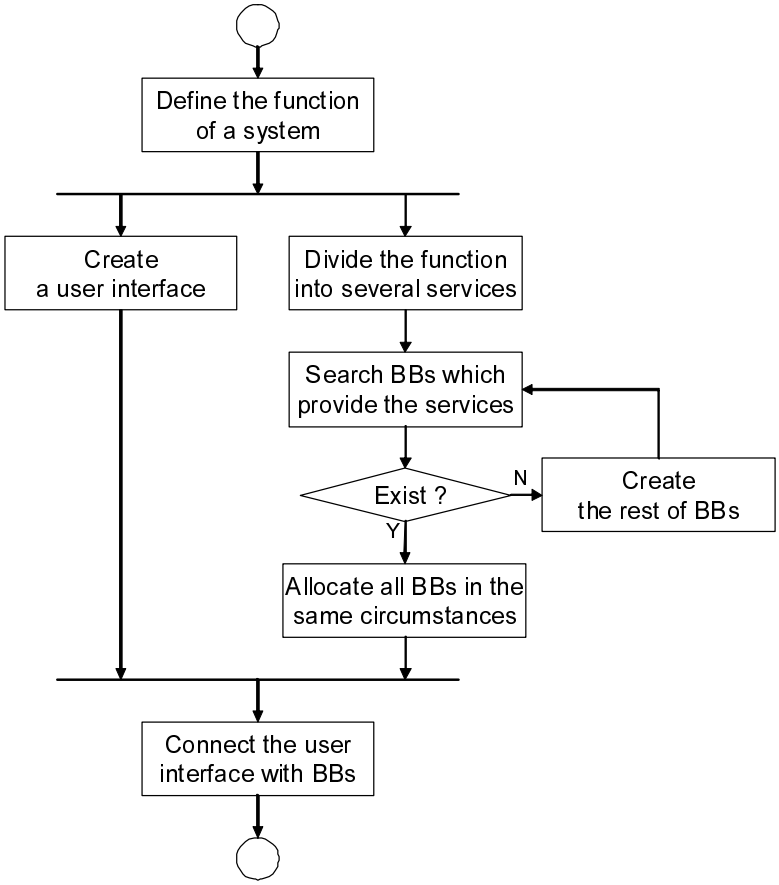

Figure 4: Procedure of Constructing a System

are different among organizations or domains which perform activities of intercultural collaboration, developers have to communicate with end users who need the support and then design and implement it.

Step2-2-1 Break down the function into several services

In order to realize the function defined in step1, developers find out necessary several services. At this time, the function should be broken down into the services in a big grain size instead of being broken down into small services. For example, in the case of constructing the system which translates sentences with user dictionary, the service editing user dictionary and the service translating sentences by using terms in the dictionary realize the function.

Step2-2-2 Search the building blocks realizing the services

Developers search the building blocks providing the services. If some of

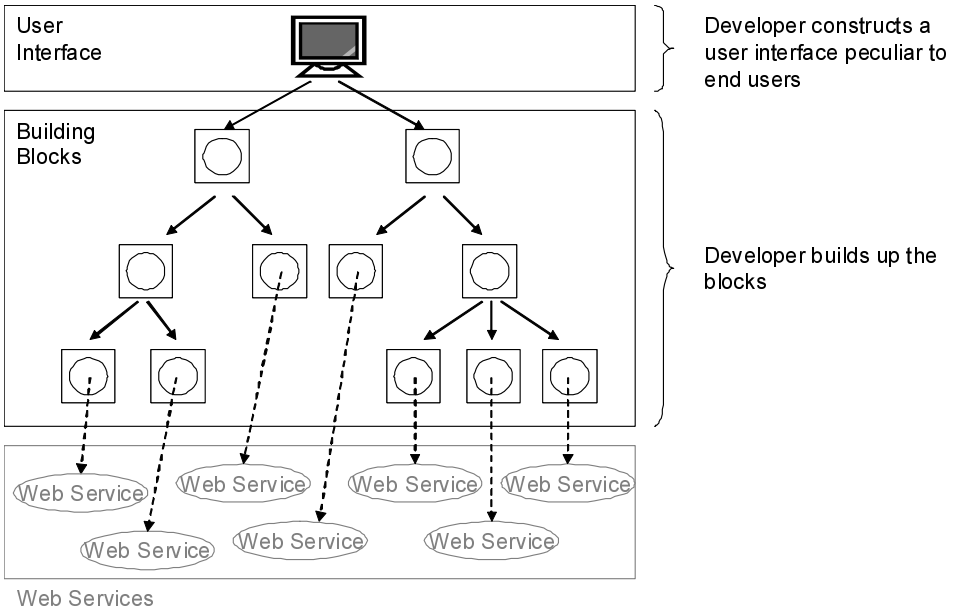

Figure 5: Construction of System

the blocks don't exist, they create new building blocks in the foregoing approach.

Step2-2-3 Allocate all building blocks in the same circumstances

If all of building blocks providing the services exist, in order to be able to invoke other building blocks, all of them should be allocated in the same circumstance. For example, in the case of building block realizing translation with user dictionary service, a building block with a method of editing user dictionary and a building block with a method of translating should be allocate in the same circumstance.

Step3 Connect user interface with building blocks

Finally, developers construct a part which gets the input from user interface, invokes several building blocks and returns the result to the interface. In order to create this part, developers can make various services by selecting and using building blocks or combining them.

Developers can construct an intercultural collaboration system with procedure mentioned above. Figure 5 shows the system architecture of the system in this approach. There are three layers in the system: the layer of web services, building blocks and the user interface. The layer of building blocks is organized

hierarchically in order to realize the service oriented programming. Terminals of building blocks invoke web services. Developers have only to combine the existing building blocks or create new building blocks by using general programming language such as PHP or Java and to create a user interface. Therefore, this procedure enables developers to construct a system using web services with no need to have knowledge and experience of describing workflows by using description language such as BPEL.

3.4 Trade-off Between Building Blocks and Web Services

We compare our approach that system is constructed by using building blocks with existing approach that system is constructed by using BPEL. Table 1 shows the result of comparison.

The advantage of system construction using building blocks is that several procedures can be described in general programming languages. Therefore, developers who have little knowledge about web services can develop, modify and debug the system. Moreover, another advantage is that processing speed of executing building blocks is higher than executing workflows described in workflow description language such as BPEL. However, since building blocks use other building blocks only in the same circumstances, all building blocks which are invoked by another block used in the system should be in there. Therefore, in the system construction, developers have to build up the blocks with considering that. Moreover, there is a problem that load is concentrated, since all building blocks are executed in the same circumstance. Transforming the building blocks frequently invoked by several systems into web services by taking advantage of the fact that a block provides a service can reduce the load. This transforming is taken by expert. If a building block is transformed into a web service, by modifying the procedure of the block to be able to invoke the web service without changing the interface, other building blocks invoking it don't need to be modified.

On the other hand, describing procedure in workflow description language such as BPEL makes components more reusable since web services can be published and then other systems can use them via networks. However, in order to

Table 1: Trade-off Between Two Approaches in the System Construction

	Building Blocks	**Web Services (BPEL)**
Advantage	-Describing and debugging are easy -Processing speed is high	-Web services can be reused by another system via networks
Disadvantage	-All building blocks used in the system need to be in the same circumstance	-Describing and debugging procedure are difficult because of abstrusity of description

create a component, knowledge about the web service and description method peculiar to workflow description language is needed. Therefore, there is a problem that it is difficult for general developers to describe the procedure, modify and debug it.

Thus the construction method with building blocks proposed in this research is very useful at the beginning of constructing systems using web services, especially intercultural collaboration systems. Although many organizations need support systems customized for them, most of them have little money employing efficient programmers and little human resource knowledgeable about web service. In such situations, it is very useful that developers who have experience to program something such as students studying computer science can construct a support system without the wealth of knowledge about web service.

Chapter 4 Layered Architecture of Language Services

Recently, many language resources are transformed into web services. However, it is difficult for single web service to satisfy the user's demand. Therefore, language services are created by combining web services and accumulated. This research also combines the web services by using building blocks. However, if language services are created ad hoc, it is difficult for developers to reuse them. Then, we propose a method of organizing language services into four layers. In this chapter, we introduce the reusability of language services and proposed the layered architecture.

4.1 Reuse of Language Services

In the Language Grid project, many language resources are wrapped and presented as web services. However, there is a big gap between the functions that the language resources provide and the functions end users need. For example, in the field of medical care, technical term should be translated accurately. Then, in order to solve it, bilingual dictionary resource of accumulated medical terms is transformed into web service by wrapping the resource. However, bilingual dictionary of medical terms cannot support the communications. On the other hand, a resource of machine translator cannot translate specific terms peculiar to medical field correctly. Therefore, combining these language resources can satisfy the user's demand. Then, recently, many composite services created by combining several web services are accumulated. In the example described above, the problem can be solved by creating the translation with bilingual dictionary service which can translate technical terms accurately with medical bilingual dictionary.

However, if language services are created ad hoc, there are following two problems of reducing the reusability. First problem is that a domain or organizations cannot use the language services created for supporting other field since the language services include procedures and invoke web services peculiar to specific domain such as medical care and education or specific organization.

For example, in the medical care domain, composite service searching several parallel texts is needed since inaccurate parallel text cannot be used. Another example is composite service combining pictogram and translator in order to communicate by using pictogram. These language services have procedure customized for specific domain or organization because the services are not used in other field. Therefore they are not reusable. Second problem is that it is difficult to search language service in need when many language services are accumulated. For example, when an organization needs the multi-hop translation service and cannot find the existing language service realizing it created by other organization, a new composite service are created and accumulated. These two problems increase the cost to create a new composite service and search existing one.

4.2 Four Layers of Language Services

Method of organizing language services into several layers is studied [4]. The research organizes the language services into four layers: system adaptation layer, combination layer, coordination layer, and user adaptation layer, by focusing translation service. In the system adaptation layer, the purpose of workflows is to adapt input sentences to machine translators. For example, the workflow replaces the abbreviations with standard nomenclature. In the combination layer, the workflow's purpose is to combine multiple translation services in order to translation between language pairs which no translation service supports. In coordination layer, the purpose of workflows is to coordinate multiple translation services in order to improve translation quality i.e. by sharing the context of translation selection with others. Finally, in the user adaptation layer, the workflow's purpose is to adapt output sentence of translation to users by combining translation service with community dictionary. Thus, classifying workflows into four layers makes the workflows whose purpose is to improve the quality of translation more reusable.

However, in the case of constructing intercultural collaboration systems, language services except translation services are accumulated. Therefore, these services should be made more reusable. Then, we organize language services

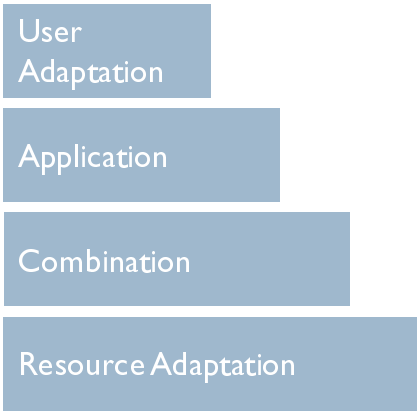

Figure 6: Layered Architecture of Language Resources

into different four layers shown in Figure 6. This approach enables users to search language services in need easily and use these services adapted to users by changing the invoked atomic services dynamically.

- Resource adaptation layer

  The goal of the resource adaptation layer is to resolve the problems unique to each language resources. For example, there is a translation web service which cannot translate the sentence including several linefeed codes since the service can translate only the sub sentence from the beginning to first linefeed code. Procedure deleting the linefeed codes as a pre-processing can solve this problem. Thus, resource adaptation layer accumulates the services which improve the input or output of web services. This layer contains concrete workflows because each language service in this layer corresponds to a specific language resource.

- Combination layer

  Services in this layer combine adapted language resources. This layer offers abstract workflows that have domain independent function, for example, multi-hop translation and translation with user dictionary. Multi-hop translation realizes the Japanese-German translation by combining Japanese-English translation service and English-Japanese translation service sequentially. Moreover translation with user dictionary improves the translation result by replacing the community terms with user dictionary.

- Application layer

Table 2: Comparison of the Four Layers

Layer	**Explanation**	**Examples of language service**
User Adaptation	Language services in the user adaptation layer provide functions specialized for end users by combining several language resources. A workflow realizing a language service is a concrete workflow containing structures and constants for them. The interface of language services is free.	- Translating Pangaea pictogram message service The service translates a sequence of pictograms to a sentence by combining the pictogram service of NPO Pangaea and machine translation.
Application	Language services in the application layer provide application domain specific functions by combining several language resources. A workflow achieving language service is an abstract workflow containing structures and constants for the domain. The interface of language services is free.	- Searching accurate parallel texts service The service searches a collection of medical parallel texts for the result of a collection of medical replay examples in order to get more accurate parallel texts.
Combination	Language services in the combination layer provide domain independent functions by combining several language resources. A workflow achieving language service is an abstract workflow. The interface of language services is free.	- Back translation service The service translates the translated sentences into the original language in order to check the accuracy of the translation. - Cross searching dictionaries service The service cross searches dictionaries at a time.
Resource Adaptation	Language services in the resource adaptation layer provide functions to adapt input data to language resources and correct errors in the processing result. A workflow realizing a language service has to bind a concrete language resource because the language service depends on the language resource. The interface of the language service has to be identical to the standardized interface.	- No-sentence-break translation service The service translates sentences after deleting all breaks in the sentence in order to make the result of a particular translator more accurate. - Auto switching search method service The service switches matching method to available one automatically in order to avoid invoking unavailable matching method.

The goal of this layer is to create composite services in order to solve the problems of specific domains. The service is abstract workflows. An example is a searching accurate parallel texts service that supports multilingual communication in hospitals. It retrieves medical question-and-answer pairs

from adjacency pair services and translates them using medical parallel text services since medical communications are unforgiving of the smallest mistake of translation.

- User adaptation layer

  The purpose of the user adaptation layer is to provide language services customized for the intended end users by combining language resources. For example, the pictogram translation service created by combining pictogram dictionary of NPO Pangaea and machine translation.

Classifying the language services with layered architecture makes the constraint that services in the upper layer can invoke a service in the same layer or the lower layer but services in the lower layer cannot invoke a service in the upper layer. Therefore, language services for general purpose are accumulated in the lower layer and customized language services for limited purpose are accumulated in the upper layer. Developers who want to use language services can search the language services customized for the organization which will be supported by the system in the user adaptation layer. If an available service does not exist, he can search language service in the application layer or lower layers. If an available service still does not exist, they can construct a system by creating a new language service combining existing general ones in the lower layer. Thus, creating language services according to the four layered architecture makes them more reusable. Table 2 shows the comparison of the layers.

Chapter 5 Language Grid Playground

We have constructed the Language Grid Playground as a practice of the two approaches described in chapter 3 and 4. The Playground is to support multilingual communities by using a group of web services on the Language Grid. In this chapter, we introduce the goal and architecture of the Language Grid Playground. We explain how the building blocks are used in the system and how the customized systems are constructed. Finally, we introduce the systems in the Language Grid Playground.

5.1 Goal of the Language Grid Playground

We constructed the systems to supporting multicultural communities according to two approaches: system construction with service-oriented building blocks and layered architecture of language services. Currently, there are many language resources such as machine translations, bilingual dictionaries, parallel texts and morphological analysis on the Language Grid. However, because these language resources are provided as web services, ordinary users who have no expertise of web service cannot use the resources. Therefore, we constructed a web site named "Language Grid Playground", which is a collection of systems to enable such ordinary users those language resources easily [5, 6]. In order to support multicultural communities, this site has four goals as described below.

- Making it easier to use language resources

  The system provides simple graphical user interfaces, which are available as web applications, to easily access language resources on the Language Grid such as machine translation and parallel texts.
- Providing new services

  New useful services can be composed by combining language resources. The system offers users new services that were created by combining language resources.
- Supporting specific organizations

  There are many organizations around the world which need support for their intercultural collaboration activities. We support such organizations

by creating customized web applications for them. The applications are constructed by combining useful services created by achieving the two targets described above. In addition, customized services are finally decomposed into reusable components, which are made available for composing new services.

- Supporting a system construction by using source codes of the systems
  This site publishes the source codes of building blocks and graphical user interfaces using these blocks. In addition, establishing the tutorial web page enables developers such as students to construct a system without knowledge about the web service.

In order to accomplish these first three goals: making it easier to use language resources, providing new services and supporting specific organizations, the Language Grid Playground provides systems which are classified into three categories corresponding to the three goals.

- BASIC category
  Systems in BASIC category provide graphical user interfaces that make it easy to invoke the language resources on the Language Grid. Currently, there are seven systems: dictionary system, concept dictionary system, pictogram system, parallel text system, morphological analyzer system, dependency parser system and translator system.
- ADVANCED category
  The ADVANCED category has systems combining the language resources, for example, dictionary creation system, composite translation system, document translation system and multilingual chat system.
- CUSTOMIZED category
  In the CUSOMIZED category, there are several systems customized to support intercultural collaboration activities in a certain community. Currently, in order to support Fujmi Junior High School and Kawasaki City Comprehensive Education Center, multilingual chat system, glossary viewer system and multilingual handout system are provided.

System use case in the Language Grid Playground is shown in Figure 7. There are following four actors.

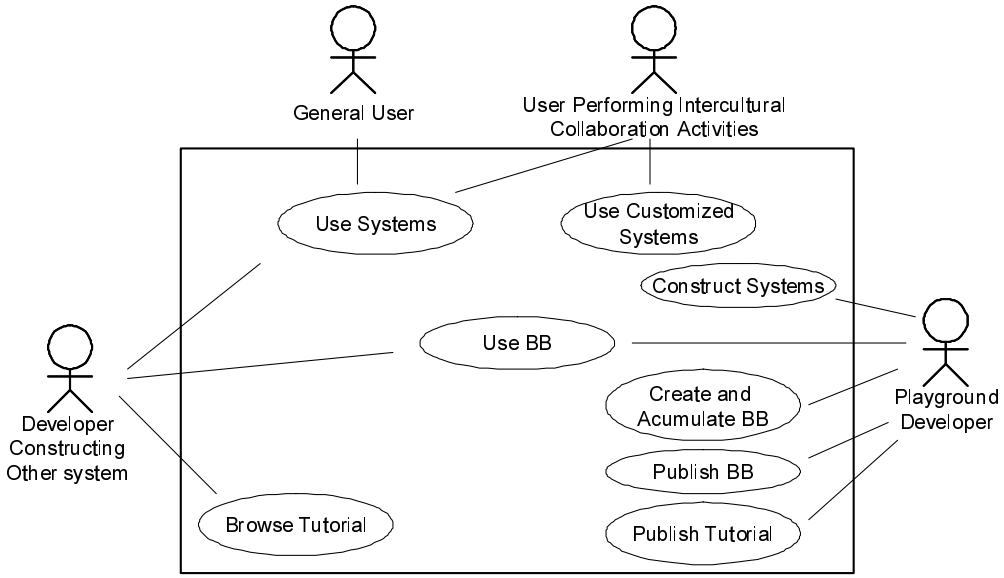

Figure 7: Use Case Diagram

- General user

  The system is published on the web. Therefore, all users who can browse web pages can try to use systems several times up to limited number.

- User performing intercultural collaboration activities

  Users who perform intercultural collaboration activities can use the customized systems constructed by playground developers for their organizations.

- Playground developer

  Playground developers can use the existing building blocks in each system construction. Moreover, they can create new building blocks by combining existing ones, accumulate and publish the blocks. In addition, they can show the system construction by publishing the tutorial web page.

- Developer constructing other system

  Developers who want to construct other intercultural collaboration systems can reuse published building blocks. In addition, they can study the system construction using building blocks with tutorial web pages.

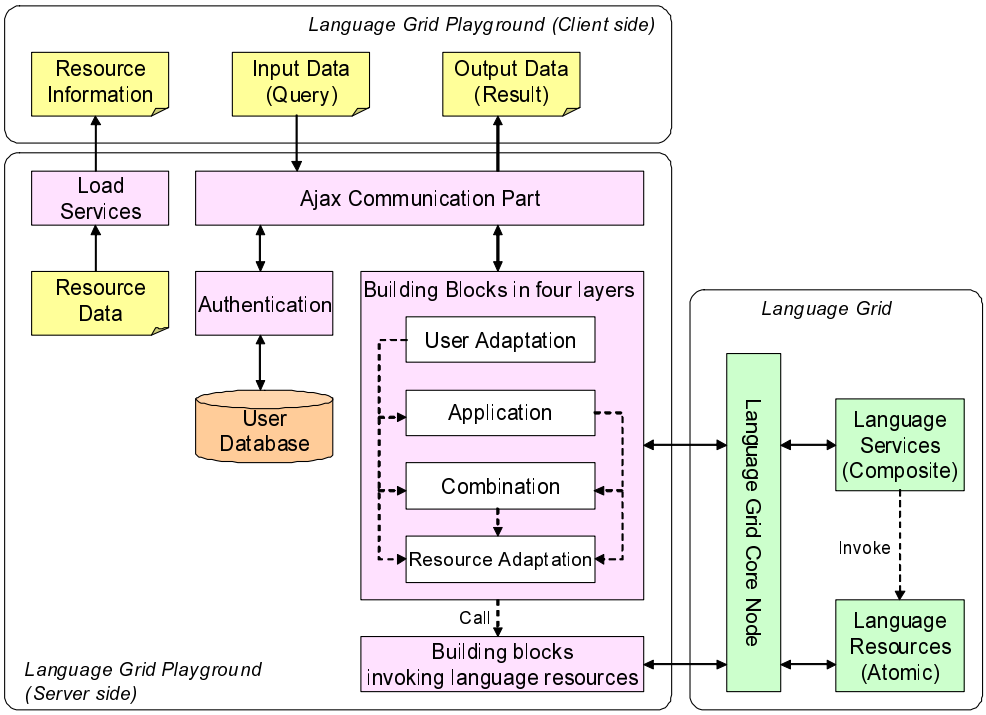

Figure 8: Architecture of the Language Grid Playground

5.2 Architecture of the Language Grid Playground

The Language Grid Playground provides graphical user interfaces (GUIs) available on the web browser since the purpose of the system is to enable general users to use language resources on the Language Grid. Therefore, the system plays a role of connecting the Language Grid and web browser. Architecture of the Language Grid Playground is shown in Figure 8.

Client side of the system creates GUI by using codes of HTML and JavaScript. Then, end users can select language resources, create the sentence which they want to translate and edit user dictionary and so on. First, when the end users load the web page of the system, server side loads the resource data in which available language resources on the Language Grid are described and displays the information to users. Users select the resources which they want to use among displayed ones. Then, in server side, ajax communication part receives the input data from web browser and authenticate users by using database of the playground user. Next, the part starts the processing by using building blocks accumulated in four layers. These building blocks use the language ser-

vices on the Language Grid or call building blocks invoking language resources. And then, ajax communication part receives the results from building blocks, transforms them into the data structure fitting the user interface and sends them to web browser. Web browser displays the result end users by transforms received data into HTML codes. Thus, users can use the language resources on the Language Grid easily.

Server side of the Language Grid Playground is described in PHP or Java mainly. Moreover, useful building blocks are created and accumulated in five layers which are added the lowest layer including building blocks invoking language resources to four layers of the language services. Therefore, building blocks have appropriate grain size and reusable. Combining these building blocks makes it possible to provide several functions to end users.

5.3 Example of Building Block

In the construction of the Language Grid Playground, many building blocks are created. We introduce the block of "Domain specific translator" as an example. This block is created to translate sentences including technical terms accurately by coordinating a multilingual user dictionary, bilingual dictionary and machine translation. Actual procedure of the block to translate is shown in Figure 9. This block uses the following three building blocks.

- Edit Dictionary

  This building block is to manage the user dictionary. This block can handle the dictionary, for example, editing terms, adding/deleting records and adding the languages by using several methods of this block. In addition, the extract() method can extract terms registered in the user dictionary from input sentence. It is used in the translation with user dictionary.

  - extract()

    Input arguments are dictionary name, source language and target language and sentences to extract terms. Output is array of extracted term pairs.

- Translation With Bilingugal Dictionary

  This building block is to translate sentences including technical terms with

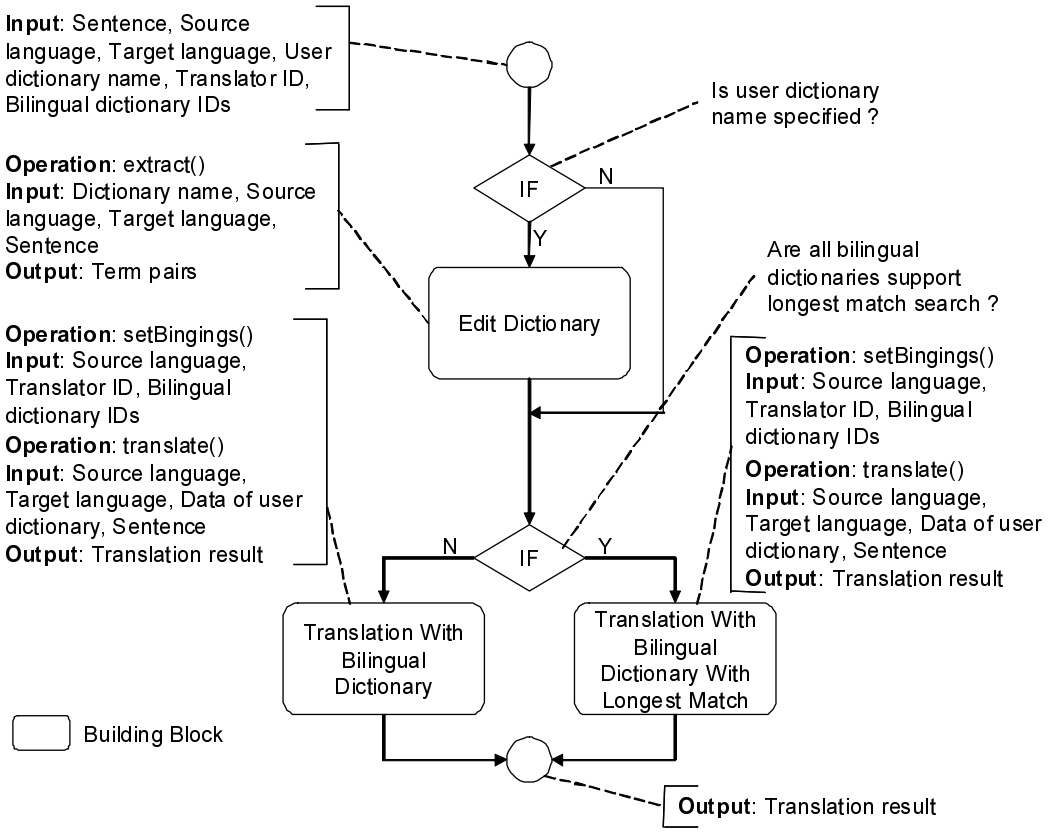

Figure 9: Domain Specific Translator

bilingual dictionaries by using composite web service "Translation Service Combined with Bilingual Dictionary" provided on the Language Grid. Since this composite web service is abstract workflow, creating the information about dynamic web service binding such as translator, morphological analyzer and bilingual dictionaries used in the web service by calling the method of setBinginds() is needed. This block realizes the translation with bilingual dictionary by invoking the web service with the information of bindings.

- setBindings()

  Input arguments are source language, language resource ID of translator and IDs of bilingual dictionaries. This function has no output.

- translate()

  Input arguments are source language, target language, user dictionary data to replace and sentence to translate. Output is the result of

Table 3: Specification of Domain Specific Translator

Assertion	__construct($dictionary)	
Argument	$dictionary	Object of the user dictionary which is used in translation to replace the community terms

Assertion	translate($source, $from, $to, $userDictName, $translator, $largedict)	
Argument	$source	Sentence to translate
	$from	The code of source language
	$to	The code of target language
	$userDictName	Name of user dictionary
	$translator	Language resource ID of translator
	$largedict	Language resource IDs of bilingual dictionaries which is used in translation to replace domain specific terms
Return value	Sentence of translation result	
Comment	This building block can translate sentences from source language to target language accurately by replacing the community and technical terms in the sentence with user dictionary and bilingual dictionary	

translation with user and bilingual dictionary.

- Translation With Bilingugal Dictionary With Longest Match
  This building block is to translate sentences including technical terms with bilingual dictionaries which support longest match search by using composite web service “Translation Service Combined with Bilingual Dictionary” on the Language Grid. Translation using composite web service supporting longest match is faster than using composite web service not supporting it. The interface of this building block is the same as the interface of “Translation with bilingual dictionary” block.
  - setBindings()
    Input arguments are source language, language resource ID of translator and IDs of bilingual dictionaries. This function has no output.
  - translate()

Input arguments are source language, target language, user dictionary data to replace and sentence to translate. Output is the result of translation with user and bilingual dictionary.

The processing procedure of "Domain Specific Translator" block is following.

Step1 If the name of user dictionary is specified in the input arguments, this block extracts terms in the user dictionary from input sentences by using "Edit dictionary" block. If the name is not specified, this block creates an empty array.

Step2 If all bilingual dictionaries in input arguments support longest match search, this block translates by using "Translation with bilingual dictionary with longest match" block. If some dictionaries do not support longest match search, this block translates by using "Translation with bilingual dictionary" block. Then, this block returns the translation result.

We construct the "Domain specific translator" block by describing these procedure. Table 3 shows the specification of the block.

5.4 Building Blocks in the Language Grid Playground

For constructing systems in the BASIC category, we create building blocks that enable to use the language resources easily. For example, "Multilingual dictionary" block enables multilingual search of dictionaries by combining multiple dictionary search functions and "Cross multilingual dictionaries" block enables cross search of dictionaries by combining the block described previously. These building blocks are classified in the combination layer. Another example, which is classified in the resource adaptation layer, limits the search methods to just full text match retrieval. It avoids that the corresponding resource from returning too many results for partial match retrieval that the Language Grid returns error because the limitation of use is exceeded.

In a dictionary service created by combining these blocks, users can select dictionaries and languages as they want and get the search results from several dictionaries at the same time. We created similar building blocks when developing parallel texts services, translation services and so on.

In ADVANCED category, we construct systems by combining building blocks

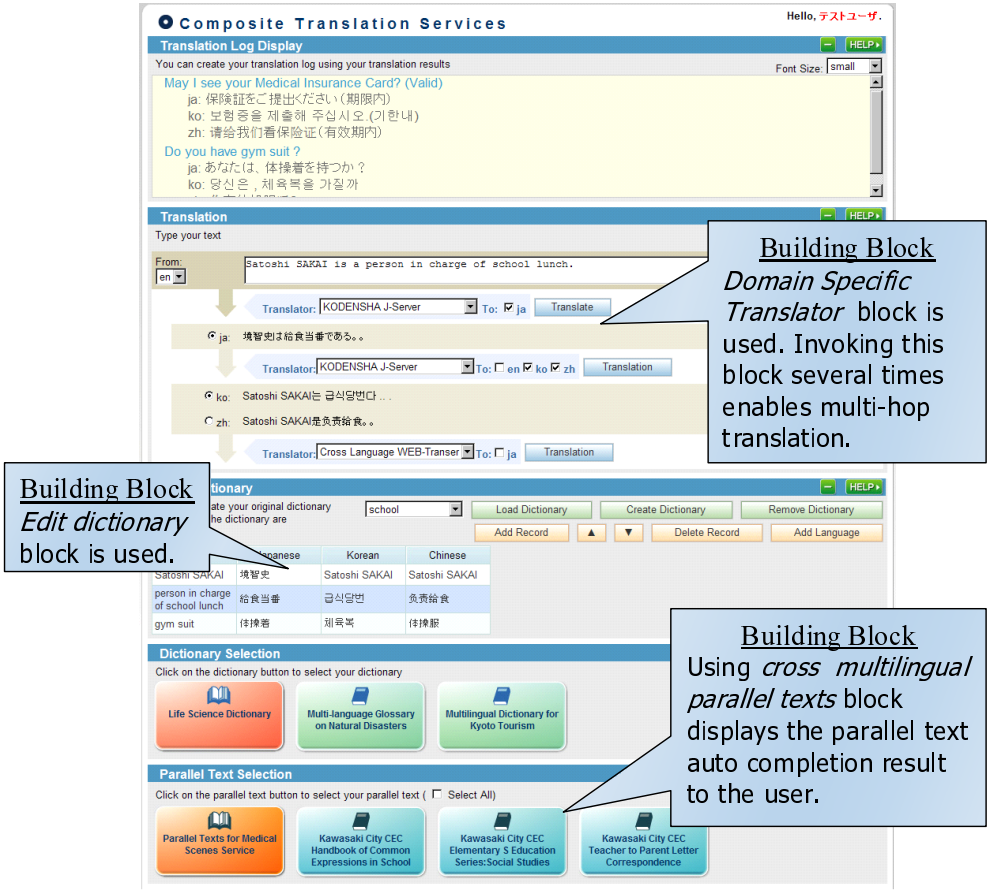

Figure 10: Building Blocks in the Composite Translation System

used in systems in BASIC category and other new building blocks. One example is the composite translation system shown in Figure 10. We create "Edit dictionary" block which enables to manage the user dictionary, for example, adding/deleting/editing terms and extract terms from sentence. Moreover, this system realizes the domain specific translation with user and bilingual dictionary by using "Domain specific translator" block. End users can raise the fluency and adequacy of translation with registering terms in the user dictionary and selecting bilingual dictionary. In addition, multi-hop translation is realized by using the block several times. Then, this system can translate between language pairs that no single translation service supported.

The composite translation system also provides auto completion by using "Cross multilingual parallel texts" block created in the system in BASIC category. Figure 11 shows the auto completion. When user input character string

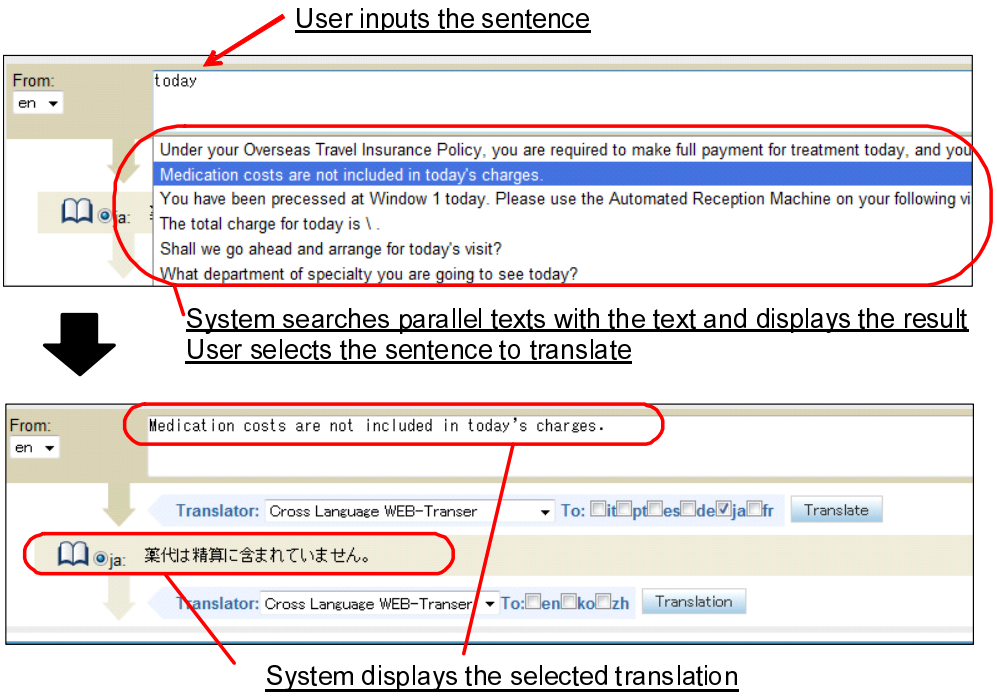

Figure 11: Auto Completion

in the text area, the system searches parallel texts with the string and displays the results to the user. If there is a sentence user wants to translate in the results, user can get the complete translation by selecting it. "Cross multilingual parallel texts" block enables multilingual search of multiple parallel texts. This block uses "Multilingual parallel text" block which can search a parallel text in multiple language. Moreover, the block uses the "Parallel text" block which invokes parallel text service on the Language Grid. The sequence of the auto completion is shown in Figure 12. First of all, when user inputs the character string into the text area on the GUI, user interface sends the string, language resource IDs of selected parallel texts and source language to ajax communication part. Then, the part calls the "Cross multilingual parallel texts" block. The block calls the "Multilingual parallel text" block to each selected parallel text. Moreover, this block calls the "Parallel text" block to each language pair. "Parallel text" block invokes web services with the string, language resource ID, source language and target language and returns the received result. After all searches are finished, ajax communication part receives the result. The part

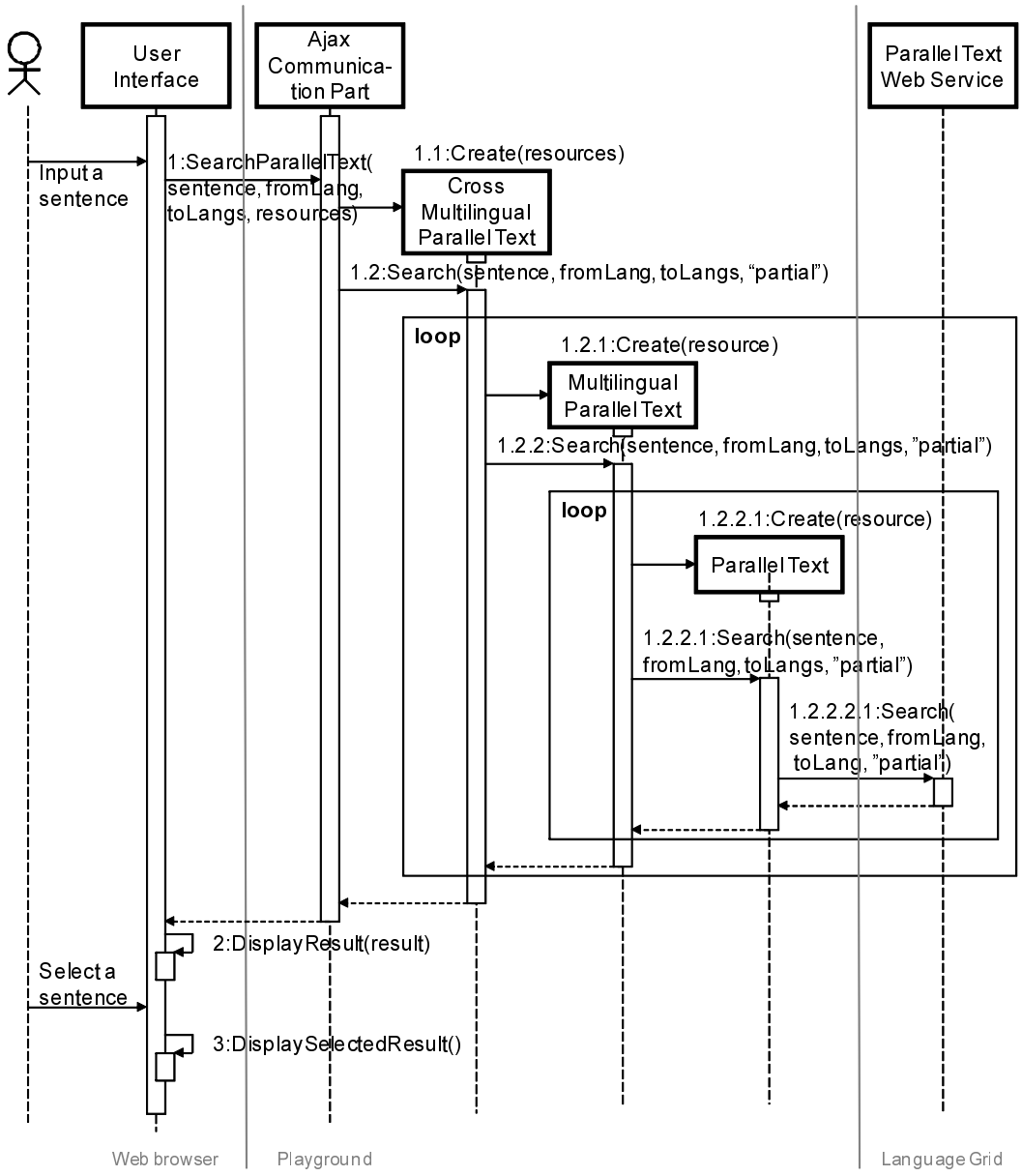

Figure 12: Sequence Diagram of Auto Completion

sends the all result to user interface and the interface displays them. If there is a sentence user wants to translate, user selects it. Finally, GUI displays the selected result into output area.

5.5 Constructing a Customized System

In the Language Grid Playground, in order to construct a customized system for specific organization, first of all, developer constructs an interface peculiar

to the organization. Since user interfaces are different according to the field which need support for intercultural collaboration, the interface needs to be construct individually. Then, developers select the building blocks which are needed to realize the function and create an ajax communication part. Ajax communication part is described to call building blocks and return the result whose data structure is transformed to fit the interface. Thus, developers can construct a customized system by building up the existing building blocks.

We constructed customized system by combining the building blocks that were developed in the construction of systems BASIC and ADVANCED and created for this system. The system was created to support Fujimi Junior High School. This school had 14 foreign students but few teachers can speak and write the students' mother tongue. Therefore, most teachers were unable to communicate with the foreign students and their parents. Moreover, because of the particular field as the school, the system should translate the terms peculiar to school such as classes, school life and career counseling. In addition, since there are the contents including students' privacy in guidance about school life and career, the third person cannot act an intermediary between teachers and students. Therefore, we construct a support system in which users can chat face-to-face in their mother language.

The constructed customized system is shown in Figure 13. In this system, students and teachers can chat by accessing the GUI in the same monitor. Interface consists of three parts. The top area displays logs of chat in two languages. In the middle area, users can input the sentence to chat and select the language which they want to input by clicking the tab. Moreover logs of chat in the selected language are displayed in the top area. Users can input sentences into text area in their mother tongue and create a translation with confirming the result of back translation. Since the back translation is the useful method to repair the input sentence [9], users can create good sentence for translation by using this function. If back translated sentences are equivalent to input sentence they want to speak, they can post it to the log area at the top of the system by clicking the post button. The bottom area enables users to edit community dictionary of Fujimi Junior High School. They can register the terms used in the

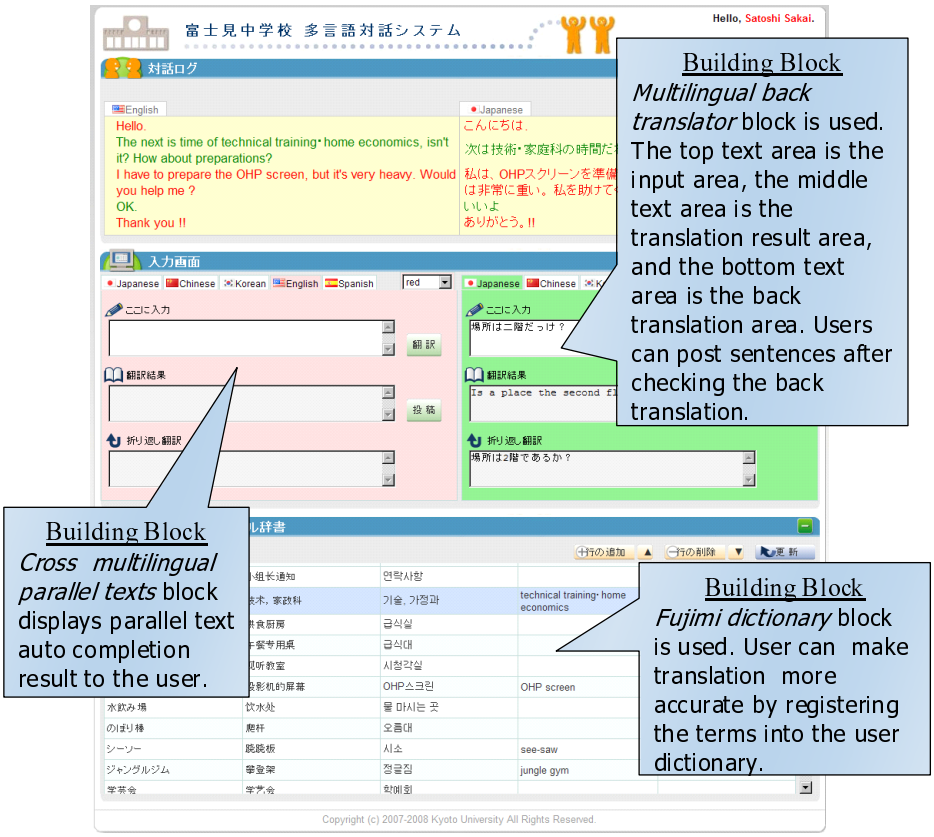

Figure 13: Customized System for Fujimi Junior High School

school in the community dictionary, which makes the translations more correct.

In order to construct this system, we create several building blocks and use them. The translation function is realized by "Multilingual back translator" block which is created by using "Back translator" block to all language pairs. "Back translator" block is to translate with user dictionary two times. Moreover, This system provides parallel text auto completion by using "Cross multilingual parallel texts" block as well as composite translation system. The parallel text resources used for auto completion are a glossary of the terms used by the school, glossary of the terms for society studies and other glossaries provided by Kawasaki city where Fujimi Junior High School is located. Moreover, "Fujimi dictionary" block enables users in the school to create their own community dictionary via an Internet browser.

Besides the customized system for Fujimi Junior High School, we construct a glossary viewer system and a multilingual handout system in order to sup-

port Kawasaki City Comprehensive Education Center. Glossary viewer system enables users to search several parallel texts at a time by selecting categories they want to search. In this system, "Multilingual parallel text with category" block is used. In addition, multilingual handout system enables users to create a multilingual handout easily. When users input the sentence in a section, translated sentence is displayed in a corresponding section. Because this system also provides auto completion, users can get the complete translation by selecting parallel text. In this system, "Back translator" block and "Cross multilingual parallel texts" block are used.

5.6 Systems on the Playground

In BASIC category, we construct following systems which have user interface that enable users to use several language resources on the Language Grid.

- Dictionary

  This system enables users to cross search dictionaries registered on the Language Grid and created by users in the dictionary creation system in multiple language.
- Concept Dictionary

  This system enables users to cross search several concept dictionaries on the Language Grid at a time. Moreover, Users can trace the concept.
- Pictogram

  This system enables users to cross search several pictograms on the Language Grid at a time. Pictogram is a dictionary that each pictogram has keywords in several languages. In this system, User can search the pictograms which have the keyword users input.
- Parallel Text

  This system enables users to cross search parallel texts on the Language Grid at a time in multiple languages.
- Morphological Analyzer

  This system enables users to cross analyze sentences by using several morphological analyzers on the Language Grid at a time.
- Dependency Parser

This system enables users to cross parse sentences by using dependency parser on the Language Grid at a time.

- Translator

  This system enables users to cross translate sentences by using several machine translators on the Language Grid at a time. Moreover the system can translate sentences to several languages.

In ADVANCED category, we construct several systems which support intercultural collaboration activities by using language resources provided as a web services on the Language Grid. Currently, there are following four systems.

- Dictionary Creation

  This system enables users to create a community dictionary and publish it as a web service. The dictionary published in this system can be searched in dictionary system and used in domain specific translation in other systems.

- Composite Translation

  Although the technology of machine translator has advanced, since sentence which users want to translate includes unknown words such as technical terms and terms peculiar end users in specific organization in many cases, the accuracy of translation becomes low. Therefore, this system enables users to combine the machine translator with bilingual dictionary on the Language Grid, community dictionary created in the dictionary creation system and temporal dictionary created in this system easily. Moreover, this system realizes the multi-hop translation by selecting the result of translation and translating it again. Therefore, users can translate between language pairs which no single translation service supports. In addition, this system provides auto completion by using parallel texts user selected. Then, users can get the complete translation easily.

- Document Translation

  This system enables to translate long article by combining machine translator with bilingual and community dictionary as well as composite translation system. Users can confirm the back translation in each sentence. By repairing original sentence or translated sentence, long article such as manual can be translated.

- Multilingual Chat

  This system enables users to chat in multiple languages. Users can translate sentences by using several machine translator, bilingual dictionary, community dictionary and temporal dictionary and confirm the accuracy of translation with the result of back translation as well as composite translation systems. Then, they can post the sentences. Therefore, they can communicate across the language barrier with their mother tongue. Moreover, by using auto completion, they are able to chat the existing complete translation.

In the CUSOMIZED category, we construct systems in order to specific organizations by using web services on the Language Grid. Currently, there are following three systems to support Fujimi Junior High School and Kawasaki City Comprehensive Education Center (CEC).

- Fujimi Junior High School Multilingual Chat

  In Fujimi Junior High School, since students whose mother tongue is not Japanese go to the school, most of teachers cannot communicate with the students or the parents of them. Therefore, we construct multilingual chat system. In this system, users can chat with their mother tongue. Details of this system are described in Chapter 5.4.

- Kawasaki City Comprehensive Education Center Parallel Text Viewer

  Kawasaki City CEC supports students whose mother tongue is not Japanese and schools which they go to. In this activity, the center has accumulated many resources of parallel texts. However, the resources are not used effectively. Therefore, Kawasaki City CEC parallel text viewer provides the function searching the parallel text resources, which are transformed as web services, with category. This function enables teachers in the school to refer several parallel texts when they make a document in foreign language. In addition, the function enables users to use parallel texts in multilingual communication.

- Kawasaki City Comprehensive Education Center Multilingual Handout Support

  When we heard in Kawasaki City CEC, we had information that a "Yu-

bisashi Card" is useful for the communication in the school. "Yubisashi Card" is a card in which parallel texts described horizontally or vertically in two or more languages. Then, users can communicate with others by pointing the sentence. For example, when a foreign student goes to the school nurse's office due to feel ill, the school nurses cannot figure out the symptom because they are unavailable to understand the student's mother tongue. In such cases, the card is used. However, the card can be created by only teachers who understand the languages which the card supports. Therefore, it takes a big cost to create cards for the general scenes except conventional ones such as nurse's office. Then, we construct Kawasaki City CEC multilingual handout system. In this system, users can create such handouts. First, users select template from "two language - eight column", "one language - eight column" and "two language - one line". Secondly, users input the sentence or word to the section in their mother tongue. Then, translated sentence is displayed in the corresponding section. Users can confirm the accuracy of translation with back translation. In addition, this system also provides auto completion.

We constructed 14 systems in the Language Grid Playground, create and accumulate 31 building blocks in the system construction.

Chapter 6 Discussion

In this chapter, we evaluate the cost for developing new web applications in the Language Grid Playground. We show the best practices of reusing building blocks. We also discuss the possibilities of applying building blocks and the layered approach to various service oriented systems.

6.1 Evaluation

Usually, constructing customized system such as multilingual chat system for Fujimi Junior High School requires a lot of programming. However, combining light weight building blocks makes easy to implement language processing parts of the system. Thus we accomplished in the construction of the customized system easily. In fact, time required for constructing the chat system was six man-weeks. Therefore, it shows that constructing by using light weight building blocks in our system is very useful.

Table 4 shows a list of building blocks used in the systems in ADVANCED and CUSTOMIZED category. This shows that several building blocks are reused in constructing six systems.

Moreover, we implement each language processing parts of the six systems according to the method of using building blocks and the method of using no building blocks. Then we compare the number of steps using our method with the number of steps of another method. The result of comparison is shown in Table 5. The result of deducting the number of steps describing building blocks from the number of all steps is very low. Therefore, it shows that six systems are constructed by little steps describing the language processing part. Moreover, although the number of steps to implement language processing parts of all six systems by using no building blocks is 2121 steps, the number of steps by using building blocks is 1563 steps including building blocks. The approach of building blocks has reduced the cost of 558 steps. This is the consequence of reusing the building blocks (Table 4.). For example, “Cross multilingual parallel texts” block is used in five systems except the glossary viewer system. In addition, in composite translation system and document translation system,

Table 4: Building Blocks Used in ADVANCED and CUSOMIZED Category

System	**Building Blocks**
Composite translation	-Domain specific translator -Translation with bilingual dictionaries with longest match -Translation with bilingual dictionaries -Cross multilingual parallel texts -Multilingual parallel text -Parallel text -Edit dictionary
Document translation	-Domain specific translator -Translation with bilingual dictionaries with longest match -Translation with bilingual dictionaries -Cross multilingual parallel texts -Multilingual parallel text -Parallel texts
Multilingual chat	-Cross multilingual parallel texts -Multilingual parallel text -Parallel texts
Multilingual chat (Fujimi)	-Multilingual back translator -Back translator -Cross multilingual parallel texts -Multilingual parallel text -Parallel texts -Fujimi dictionary
Glossary viewer	-Multilingual text with categories -Parallel texts with category
Multilingual handout	-Multilingual back translator -Back translator -Cross multilingual parallel texts -Multilingual parallel text -Parallel texts

"Domain specific translator" block is used. In Multilingual chat (Fujimi) system and multilingual handout system, "Back translator" block is used. Thus, it is possible to reuse more building blocks as the number of systems increases.

The relation between the number of systems and the number of steps described in language processing part is shown in Figure 14. When we constructed the composite translation systems, the steps described by using building blocks are more than steps by no building blocks since there are steps of super class and several useful functions of building blocks. However, in the construction of

Table 5: Comparison of Step Number (x) means number of steps of building blocks

System	With Building Blocks	Without Building Blocks
Composite translation	1076 (1048)	907
Document translation	7 (0)	403
Multilingual chat	17 (0)	116
Multilingual chat (Fujimi)	282 (185)	329
Glossary viewer	150 (107)	114
Multilingual handout	31 (0)	222
Sum	1563 (1340)	2121

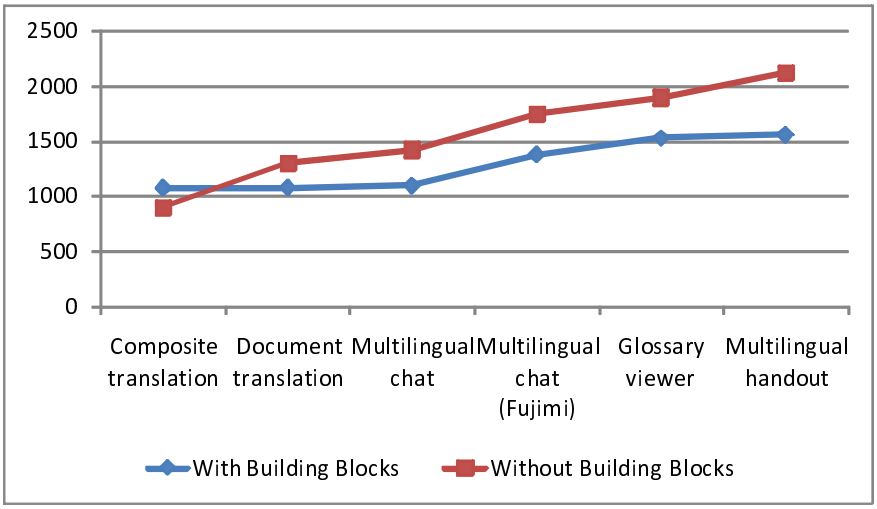

Figure 14: Transition of Step Number

the document translation system, building blocks are reused. Then, the number of steps described in our method is less than the number of steps in another method. After that, the difference of the number of steps is increased once a new system is constructed. Thus, in construction of the intercultural collaboration systems, there are many processing using language resources such as machine translator and parallel text. Therefore, providing building blocks which enables such processing and reusing them can reduce the cost of system construction. Hereby, it shows that the approach of using building blocks in

Figure 15: Example of the Web Page Displaying Source Code

system construction is very useful for intercultural collaboration systems.

In this research, because we have constructed only three customized systems, all building blocks are general and classified into combination layer and resource adaptation layer. Therefore, effectiveness of the upper layers of the layered architecture of language services is not showed. However, constructing systems in the Language Grid Playground is continued. When more building blocks are accumulated, we need discuss about the effectiveness of the upper layers.

6.2 Trial of Reuse of Building Blocks

In this research, we have created and accumulated many building blocks. Therefore, we have following activities in order to promote the reuse of building blocks on the Language Grid Playground for developers to construct other systems.

6.2.1 Open Source

First of all, we publish the source codes of Language Grid Playground on the web as a practice of reusing building blocks (Figure 15). Published source codes are codes of HTML and JavaScript realizing GUI and codes of ajax communication part and building blocks described in PHP. By using these source codes, general developers can construct a system using building blocks.

Moreover, source codes are provided on each system. Therefore, developers who have no knowledge about the web service can refer the system which has an equivalent function they want to construct. In addition, by reference the codes of system, they can learn how to build the blocks and construct a GUI using the blocks.

For example, developers can construct a system whose functions are translation and auto completion. First, they copies the source codes of "Domain specific translator" block, "Cross multilingual parallel texts" block and all building blocks called by them and configuration file and file of managing resources described in PHP. Then, they can invoke the web service with building blocks. Secondly, by copying the codes of HTML and JavaScript, they can realize the GUI. Finally, by modifying the code for their customized system, they can construct an intercultural collaboration system based on composite translation system.

6.2.2 Tutorial

We also create the tutorial site about system construction using web services as a practice of reusing building blocks (Figure 16). This site explains the construction of circumstance to use language resources on the Language Grid with building blocks. Then, the site introduces the system architecture of the Language Grid Playground. After the explanation, developers can construct three simple systems using building blocks with these examples. First example is to construct the system translating sentences. The system is constructed by using "Translator" block.

Second example is to construct the system realizing the back translation. The system is achieved by calling the "Translator" block two times. In order to construct this system, first of all, developers create the user interface by describing codes of HTML and JavaScript. Then, they describe the procedure of calling the blocks two times and returning the result to user interface.

Third example is to construct the system realizing the back translation with temporal dictionary and Kyoto tourism dictionary, which is bilingual dictionary. This system is constructed in the same way of second example. Developers can achieve the back translation by creating the user interface and describing the

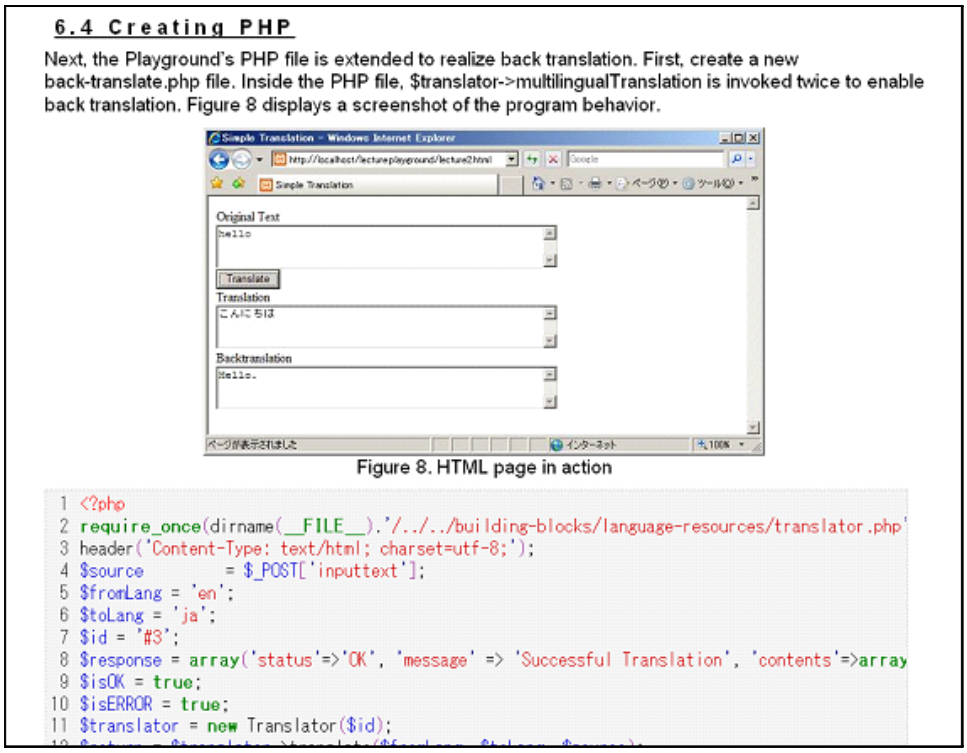

Figure 16: Tutorial Site

procedure of calling "Translation with user and public dictionaries with longest match" block two times.

By browsing the tutorial sites and constructing three systems as a example, general developers can learn the method of system construction using building blocks and then, they are able to construct an intercultural collaboration system using web services on the Language Grid easily.

Moreover, we had workshops using this tutorial several times. We have instructed many students of 12 laboratories of 9 universities: Kwansei Gakuin University, Future University - Hakodate, Tohoku University, Ritsumeikan University, Kansai University, Waseda University, Nagaoka University of Technology, Chiba Institute of Technology, Japan Advanced Institute of Science and Technology.

6.3 Applying Building Blocks to Other Field

In this research, we propose the method of constructing intercultural collaboration systems with the approach of service-oriented programming. Recently, many web services not only language services are provided around the world. Therefore, we discuss about the adaptive possibility of our approaches to construct systems in other field.

6.3.1 Concept of Building Block

Currently, various services such as hotel reservation, weather forecast and time table of transportation are transformed into web services. Combining these web services is able to satisfy the users' demand. For example, by combining web services of weather forecast and time table of bus and train, the service which informs users of best transportation is achieved. However, developers should create composite web services by using workflow description language in order to combine several web services.

The method of system construction using building blocks realizes the paradigm of service oriented programming. Moreover, the building blocks can invoke web services as method call of general programming. Therefore, the method can be applied the composite web services like the example described above. Then, our method is useful in all system construction using web services.

6.3.2 Concept of Layered Architecture

Currently, there are various web services around the world. Therefore, it is difficult to classify the composite web services consisted of such various web services into four layers of the language services. Moreover, since there is enormous number of combination, further classification method, for example, making category in each layer is needed.

Chapter 7 Conclusion

In order to support intercultural collaboration activities, systems should be easily customized to realize the tasks demanded by each field. Meanwhile, combining several web services enables to construct customized systems. However, since professional knowledge about web service and the workflow description language such as BPEL is needed to combine the web services, it is difficult for general developers to construct a system using web services. Then, we propose the method of system construction using web services with general programming language such as PHP and classification of language services. In addition, we have developed the Language Grid Playground to supporting multicultural communities according to these two approaches. The contributions of this research are as follows.

Developing of Service-Oriented Building Blocks

In order to achieve a paradigm of the service-oriented programming by using a general programming language, we proposed the method of creating the building blocks and the technique to construct systems using them. Moreover, through the construction of the Language Grid Playground, we have created and accumulated several useful building blocks. In fact, we have created 31 building blocks and constructed 14 systems including a customized system to support Fujimi Junior High School and two systems to support Kawasaki City Comprehensive Education Center.

Realizing Layered Architecture of Language Services

Creating and accumulating several building blocks using web services of language resources enables developers to construct a new system easily. However, if these building blocks are created ad hoc, it becomes difficult to reuse them because the blocks which cannot be used in other systems are accumulated and discovering the blocks is difficult. Therefore, we propose the layered architecture of language services. This architecture classifies the language services into four layers: resource adaptation layer, combination layer, application layer and user adaptation layer. By classifying language services according to this architecture, limited ones are accumulated in the upper layers and general ones are

accumulated in the lower layers. We have made building blocks in the Language Grid Playground more reusable with this architecture.

Moreover, we compared the cost of constructing systems using building blocks with using no building blocks. The result of comparison shows that system construction with building blocks is useful for customized intercultural collaboration systems since the number of steps with the method using building blocks is less than the number with another method. In addition, we have published the source codes of building blocks and user interfaces and the tutorial site to learn how to use building blocks. Moreover, we have held the workshop using this site several times and instructed many students of 12 laboratories of 9 university. Thus, we create a structure for developers to build up the blocks easily for intercultural collaboration systems.

Acknowledgments

The author would like to express his gratitude to the supervisor, Professor Toru Ishida at Kyoto University, for his consecutive instruction, pertinent advice and earnest argument.

The author would like to tender his acknowledgments to Associate Professor Shigeo Matsubara and Assistant Professor Hiromitsu Hattori at Kyoto University, for his technical and constructive advice.

The author would like to express his appreciations to the advisers, Professor Yoshiaki Kawata at Kyoto University, Researcher Yohei Murakami at National Institute of Information and Communications Technology, and Associate Professor Mizuho Iwaihara at Kyoto University for his valuable advice.

Finally, the author would like to thank all members of Ishida and Matsubara laboratory for their various supports and discussions.

References

[1] T. Ishida. Language grid: An infrastructure for intercultural collaboration. In *IEEE/IPSJ Symposium on Applications and the Internet (SAINT-06)*, pp. 96–100, 2006.

[2] R. Khalaf, N. Mukhi, and S. Weerawarana. Service-oriented composition in bpel4ws. In *Proceedings of the World Wide Web Conference*, 2003.

[3] M. Miyabe, K. Fujii, T. Shigenobu, and T. Yoshino. Parallel-text based support system for intercultural communication at medical receptions. In *Proceedings of The First International Workshop on Intercultural Collaboration (IWIC 2007)*, pp. 132–143, 2007.

[4] Y. Murakami and T. Ishida. A layered language service architecture for intercultural collaboration. In *The Sixth International Conference on Creating, Connecting and Collaborating through Computing (C5 2008)*, 2008.

[5] S. Sakai, M. Gotou, S. Morimoto, Y. Murakami, and T. Ishida. Language grid playground: Intercultural collaboration environment using light weight building blocks (in japanese). *Journal of Human Interface Society*, Vol. 11, No. 1, (to be published).

[6] S. Sakai, M. Gotou, Y. Murakami, S. Morimoto, D. Morita, M. Tanaka, and T. Ishida. Language grid playground: Light weight building blocks for intercultural collaboration. In *International Workshop on Intercultural Collaboration (IWIC-2009)*, (to be published).

[7] S. Sakai, M. Gotou, M. Tanaka, R. Inaba, Y. Murakami, T. Yoshino, Y. Hayashi, Y. Kitamura, Y. Mori, T. Takasaki, Y. Naya, A. Shigeno, S. Matsubara, and T. Ishida. Language grid association: Action research on supporting the multicultural society. In *International Conference on Informatics Education and Research for Knowledge-Circulating Society (ICKS'08)*, 2008.

[8] A Sillitti, T Vernazza, and Succi G. Service oriented programming: a new paradigm of software reuse. In *Seventh International Conference on Software Reuse (ICSR-7, Austin, TX, USA, April 2002)*, 2002.

[9] N. Yamashita, T. Sakamoto, T. Nomura, S. Ishida, Y. Hayashi, K. Ogura,

and H. Isahara. Analyzing user adaptation toward machine translation systems (in japanese). *Transactions of Information Processing Society of Japan*, Vol. 47, No. 4, pp. 1276–1286, 2006.

Appendix

A.1 Specification of Building Blocks

Cross dependency persers

Table A.1: Constructor

Assertion	__construct($resourceIds)	
Argument	$resourceIds	IDs of dependensy parser services

Table A.2: crossParse

Assertion	crossParse($language, $sentence)	
Argument	$language	The code of language to parse
	$sentence	A text to parse
Return value	The parse results	
Comment	This building block can parse a sentence with several dependency parsers	

Cross multilingual analyzers

Table A.3: constructor

Assertion	__construct($resourceIds)	
Argument	$resourceIds	IDs of morphological analyzer services

Table A.4: crossMultilingualAnalyze

Assertion	crossMultilingualAnalyze($fromLang, $source)	
Argument	$fromLang	The code of language to analyze
	$source	A text to analyze
Return value	The analyze results	
Comment	This building block can analyze with several morphological analyzers	

Cross multilingual concept dictionaries

Table A.5: constructor

Assertion	__construct($resourceIds)	
Argument	$resourceIds	IDs of concept dictionary services

Table A.6: crossMultilingualSearch

Assertion	crossMultilingualSearch($headLang, $headWord, $matchingMethod)	
Argument	$headLang	The code of language to search
	$headWord	The word used in search
	$matchingMethod	The code of matching method
Return value	The search results	
Comment	This building block can search by matching method with several concept dictionaries	

Cross multilingual dictionaries

Table A.7: constructor

Assertion	__construct($resourceIds)	
Argument	$resourceIds	IDs of dictionary services

Table A.8: crossMultilingualSearch

Assertion	crossMultilingualSearch($headLang, $targetLangs, $headWord, $matchingMethod)	
Argument	$headLang	The code of source language
	$targetLangs	The codes of target languages
	$headWord	A text to describe into the target language
	$matchingMethod	The code of matching method
Return value	Transltaions found with the head word by matching method	
Comment	This building block can search several bilingual dictionaries with the head word in multiple language	

Cross multilingual parallel texts

Table A.9: constructor

Assertion	__construct($resourceIds)	
Argument	$resourceIds	IDs of parallel text services

Table A.10: crossMultilingualSearch

Assertion	crossMultilingualSearch($headLang, $targetLangs, $headWord, $matchingMethod)	
Argument	$headLang	The code of source language
	$targetLangs	The codes of target languages
	$headWord	A text to describe into the target languaege
	$matchingMethod	The code of matching method
Return value	Translations found with the head word by the matching method	
Comment	This building block can search several parallel texts with the head word in multiple lanugage	

Cross multilingual translators

Table A.11: constructor

Assertion	__construct($resourceIds)	
Argument	$resourceIds	IDs of translation services

Table A.12: crossMultilingualTranslate

Assertion	crossMultilingualTranslate($fromLang, $toLangs, $source)	
Argument	$fromLang	The code of the source language
	$toLangs	The codes of target language
	$source	A text to translate
Return value	The transltaion results	
Comment	This building block can translate sentence to multiple language with several machine translator	

Domain specific translator

Table A.13: constructor

Assertion	__construct($dictionary)	
Argument	$dictionary	Object of the user dictionary which is used in translation to replace the community terms

Table A.14: translate

Assertion	translate($source, $from, $to, $userDictName, $translator, $largedict)	
Argument	$source	Sentence to translate
	$from	The code of source language
	$to	The code of target language
	$userDictName	Name of user dictionary
	$translator	Language resource ID of translator
	$largedict	Language resource IDs of bilingual dictionaries which is used in translation to replace domain specific terms
Return value	Sentence of translation result	
Comment	This building block can translate sentence from source language to target language accurately by replacing the community and technical terms in the sentence with user dictionary and bilingual dictionary	

Table A.15: translateWithDictionaryTable

Assertion	translateWithDictionaryTable($source, $from, $to, $dictionaryTableJSON, $translator, $largedict)	
Argument	$source	A text to translate
	$from	The code of source language
	$to	The code of target language
	$dictionaryTableJSON	Dictionary data with the JSON format（JSON 形式）

	$translator	Language resource ID of translator
	$largedict	Language resource IDs of bilingual dictionaries which is used in translation to replace domain specific terms
Return value	Sentence of translation result	
Comment	This building block can translate sentence from source language to target language accurately by replacing the community and technical terms in the sentence with dictionary data and bilingual dictionary	

Multihop userdictionary translator

Table A.16: constructor

Assertion	__construct($dictionary)	
Argument	$dictionary	Object of the user dictionary which is used in translation to replace the community terms

Table A.17: translate

Assertion	translate($source, $from, $langs, $dictName)	
Argument	$source	A text to translate
	$from	The code of source language
	$langs	Array of language code used multi-hop translation
	$dictName	Name of user dictionary
Return value	Translation sentence	
Comment	This building block can multi-hop translate with user dictionary	

Multilingual analyzer

Table A.18: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of morphological analyzer

Table A.19: multilingualAnalyze

Assertion	multilingualAnalyze($sourceLang, $source)	
Argument	$sourceLang	The codes of languages to analyze
	$source	A text to analyze
Return value	The analyze result	
Comment	This building block can analyze a sentence in multiple language.	

Multilingual back translator

Table A.20: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of back translator service

Table A.21: multilingualBackTranslate

Assertion	multilingualBackTranslate($sourceLang, $targetLangs, $source)	
Argument	$sourceLang	The code of source language
	$targetLangs	The codes of target languages
	$source	A text to translate
Return value	The results of trnaslation and back translation	
Comment	This building block can back translate in multiple languages	

Multilingual dictionary

Table A.22: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of dictioanry service

Table A.23: multilingualSearch

Assertion	multilingualSearch($headLang, $targetLangs, $headWord, $matchingMethod)	
Argument	$headLang	The code of source language

	$targetLangs	The codes of target languages
	$headWord	A text to describe into the target language
	$matchingMethod	The code of matching method
Return value	Translations found with the head word by the matching method	
Comment	This building block can search dictionaries in multiple languages	

Multilingual parallel text

Table A.24: constructor

Assertion	__construct($resourceId)	
Argument	$resourceIds	ID of parallel text service

Table A.25: multilingualSearch

Assertion	multilingualSearch($headLang, $targetLangs, $headWord, $matchingMethod)	
Argument	$headLang	The code of source language
	$targetLangs	The codes of target languages
	$headWord	A text to describe into the target languages
	$matchingMethod	The code of matching method
Return value	Translations found with the headWord by the matching method	
Comment	This building block can search parallel text in multiple languages	

Multilingual paralel text with category

Table A.26: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of parallel text service

Table A.27: multilingualSearchWithCategory

Assertion	multilingualSearchWithCategory($category, $headLang, $targetLangs, $source, $matchingMethod)	
Argument	$category	The category to search
	$headLang	The code of source language
	$targetLangs	The codes of target languages
	$source	A text to describe into the target languages
	$matchingMethod	The code of matching method
Return value	Translations found with the headWord by the matching method	
Comment	This building block can search parallel texts with category in multiple languages	

Multilingual translator

Table A.28: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of translate service

Table A.29: multilingualTranslate

Assertion	multilingualTranslate($sourceLang, $targetLangs, $source)	
Argument	$sourceLang	The code of source language
	$targetLangs	The codes of target languages
	$source	A text to translate
Return value	Translation result	
Comment	This building block can translate in multiple languages	

Parallel multilingual user dictionary translator

Table A.30: constructor

Assertion	__construct($dictionary)

Argument	$dictionary	Object of the user dictionary which is used in translation to replace the community terms

Table A.31: translate

Assertion	translate($source, $from, $langs, $dictName)	
Argument	$source	A text to translate
	$from	The code of source language
	$langs	The codes of target languages
	$dictName	Name of user dictionary
Return value	The results of translation	
Comment	This building block realize several multi-hop translations	

Translation With Bilingual Dictionary

Table A.32: constructor

Assertion	__construct()

Table A.33: setBindings

Assertion	setBindings($from, $translator, $largedicts)	
Argument	$from	The code of source language
	$translator	ID of translation service
	$largedicts	IDs of bilingual dictionaries which is used in translation to replace domain specific terms
Comment	This method sets bindings to invoke “Translation Service Combined With Bilingual Dictionary” service	

Table A.34: translate

Assertion	translate($from, $to, $userDict, $source)	
Argument	$from	The code of source language
	$to	The code of target language

	$userDict	Dictionary data
	$source	A text to translate
Return value	Translation result	
Comment	This building block can translation with user dictionary and bilingual dictionary by invoking “Translation Service Combined With Bilingual Dictionary”	

Translation with bilingual dictionary with longest match

Table A.35: constructor

Assertion	__construct()

Table A.36: setBindings

Assertion	setBindings($from, $translator, $largedicts)	
Argument	$from	The code of source language
	$translator	ID of translate service
	$largedicts	IDs of bilingual dictionaries which is used in translation to replace domain specific terms
Comment	This method sets bindings to invoke “Translation Service With Bilingual Dictionary With Longest Match Search” service	

Table A.37: translate

Assertion	translate($from, $to, $userDict, $source)	
Argument	$from	The code of source languages
	$to	The code of target language
	$userDict	Dictionary data
	$source	A text to translate
Return value	Translation result	
Comment	This building block can translation with user dictionary and bilingual dictionary by invoking “Translation Service With Bilingual Dictionary With Longest Match Search”	

User dictionary translator

Table A.38: constructor

Assertion	__construct($dictionary)	
Argument	$dictionary	Object of the user dictionary which is used in translation to replace the community terms

Table A.39: translate

Assertion	translate($source, $from, $lang, $dictName)	
Argument	$source	A text to translate
	$from	The code of source language
	$lang	The code of target language
	$dictName	Name of dictionary
Return value	Translation result	
Comment	This building block can translation with user dictionary	

EDR adaptation

Table A.40: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of bilingual dictionary service

Table A.41: search

Assertion	search($headLang, $targetLang, $headWord, $matchingMethod)	
Argument	$headLang	The code of source language
	$targetLang	The code of target language
	$headWord	A text to describe into the target language
	$matchingMethod	The code of matching method
Return value	Translations founc with the head word by the mathcing method	

Comment	This building block can search dictionary by the COMPLETE method

Pangaea pictogram adaptation

Table A.42: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of pictogram service

Table A.43: search

Assertion	search($language, $word, $matchingMethod)	
Argument	$language	The code of language to search
	$word	A text to search
	$matchingMethod	The code of matching method
Return value	Pictogram data to display in the web browser	
Comment	This building block can search Pangaea pictogram and transform the result to the structure to display in web browser	

Analyzer

Table A.44: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of analyzer service

Table A.45: analyze

Assertion	analyze($sourceLang, $source)	
Argument	$sourceLang	The code of language to analyze
	$source	A text to analyze
Return value	Analyze result	
Comment	This building block can analyze a sentence	

Back translator

Table A.46: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of back translation service

Table A.47: backTranslate

Assertion	backTranslate($sourceLang, $intermediateLang, $source)	
Argument	$sourceLang	The code of source language
	$intermediateLang	The code of intermediate language
	$source	A text to translate
Return value	The result of translation and back translation	
Comment	This building block can realize back translation	

Concept dictionary

Table A.48: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of concept dictionary service

Table A.49: search

Assertion	search($headLang, $headWord, $matchingMethod)	
Argument	$headLang	The code of lnaguage to search
	$headWord	A text to search
	$matchingMethod	The code of matching method
Return value	Search result	
Comment	This building block can search concept dictionary	

Dependency parser

Table A.50: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of dependency parser service

Table A.51: parse

Assertion	parse($language, $sentence)	
Argument	$language	The code of language to parse
	$sentence	A text to parse
Return value	Parse result	
Comment	This building block can parse a sentence	

Dictionary

Table A.52: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of bilingual dictionary service

Table A.53: search

Assertion	search($headLang, $targetLang, $headWord, $matchingMethod)	
Argument	$headLang	The code of source language
	$targetLang	The code of target language
	$headWord	A text to search
	$matchingMethod	The code of matching method
Return value	Translations found with the head word by the matching method	
Comment	This building block can search dictionary	

Edit dictionary

Table A.54: constructor

Assertion	__construct($organizationId)	
Argument	$organizationId	Organization ID of the Language Grid Playground

Table A.55: createDictionary

Assertion	createDictionary($dictName)

Argument	$dictName	Name of user dictionary
Return value	Success or failure	
Comment	This method can create a user dictionary	

Table A.56: createDictionaryWithEmptyRecord

Assertion	createDictionaryWithEmptyRecord($dictName, $aLanguage, $number)	
Argument	$dictName	User dictionary name
	$aLanguage	Languages used in the user dictionary
	$number	The number of redords to create
Return value	Success or failure	
Comment	This method can create a dictionary with specified languages and records	

Table A.57: removeDictionary

Assertion	removeDictionary($dictName, $time)	
Argument	$dictName	User dictionary name
	$time	Time at invoking this method
Return value	Success or failure	
Comment	This method can delete specified dictionary	

Table A.58: getDictionaryList

Assertion	getDictionaryList()
Return value	Dictionary list
Comment	This method can get the list of dictionary

Table A.59: getDictionaryData

Assertion	getDictionaryData($dictName)	
Argument	$dictName	User dictionary name
Return value	Data of the user dictionary	

Comment	This method can get the dictionary data

Table A.60: addRecord

Assertion	addRecord($dictName, $number, $time)	
Argument	$dictName	User dictionary name
	$number	The number of records to add
	$time	Time at invoking this method
Return value	Success or failure	
Comment	This method can add records to user dictionary	

Table A.61: removeRecord

Assertion	removeRecord($dictName, $aTermId, $time)	
Argument	$dictName	User dictionary name
	$aTermId	Term's id to delete
	$time	Time at invoking this method
Return value	Success or failure	
Comment	This method can delete records of user dictionary	

Table A.62: addLanguage

Assertion	addLanguage($dictName, $aLanguage, $time)	
Argument	$dictName	User dictionary name
	$aLanguage	The codes of languages to add
	$time	Time at invoking this method
Return value	Success or failure	
Comment	This method can add languages to user dictionary	

Table A.63: editTerm

Assertion	editTerm($dictName, $termId, $aLanguage, $aTerm, $time)	
Argument	$dictName	User dictionary name
	$termId	Term's id to edit
	$aLanguage	The code of language to edit

	$aTerm	A new word of editing term
	$time	Time at invoking this method
Return value	Success of failure	
Comment	This method can edit term	

Table A.64: setPriority

Assertion	setPriority($dictName, $aTermId, $aPriority, $time)	
Argument	$dictName	User dictionary name
	$termId	The list of term's id
	$aPriority	The list of priority
	$time	Time at invoking this method
Return value	Success of failure	
Comment	This method can change the term's priority	

Table A.65: extract

Assertion	extract($dictName, $from, $to, $source)	
Argument	$dictName	User dictionary name
	$from	The code of source language
	$to	The code of target language
	$source	A text to be extract
Return value	The result of extract	
Comment	This method can extract the terms in the user dictionary from input sentence	

Table A.66: getLanguage

Assertion	getLanguage($dictName)	
Argument	$dictName	User dictionary name
Return value	The list of languages supported in the user dictionary	
Comment	This method can get the list of languages supported in the user dictionary	

Fujimi dictionary

Table A.67: constructor

Assertion	__construct()

Table A.68: callGetSupportedLanguageByDictionary

Assertion	callGetSupportedLanguageByDictionary()
Return value	The list of supported languages
Comment	This method can get the list of supported language in the dictionary

Table A.69: callGetDictionaryByLang

Assertion	callGetDictionaryByLang($lang)	
Argument	$lang	The code of language
Return value	Dictionary data	
Comment	This method can get dictionary data in the language	

Table A.70: callRegisterTerm

Assertion	callRegisterTerm($lang, $term)	
Argument	$lang	The code of language
	$term	A new word
Return value	Success or failure	
Comment	This method can add a new term in specified language	

Table A.71: callDeleteLine

Assertion	callDeleteLine($term)	
Argument	$term	Term's id
Return value	Success or failure	
Comment	This method can delete the specified record	

Table A.72: callUpdateTerm

Assertion	callUpdateTerm($termId,$lang,$term)	
Argument	$termId	Term's id
	$lang	The code of language
	$term	A new word
Return value	Success or failure	
Comment	This method can update the term with a new term	

Table A.73: callSetPriority

Assertion	callSetPriority($termId, $priority)	
Argument	$termId	Term's id
	$priority	The priority
Return value	Success or failure	
Comment	This method can set priority to the record	

Parallel text

Table A.74: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of parallel text service

Table A.75: Search

Assertion	search($sourceLang, $targetLang, $source, $matchingMethod)	
Argument	$sourceLang	The code of source language
	$targetLang	The code of target language
	$source	A text to describe into the target language
	$matchingMethod	The code of matching method
Return value	Translations found with head word by matching method	
Comment	This building block can search parallel text	

Parallel text with category

Table A.76: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of parallel text with category service

Table A.77: searchWithCategory

Assertion	searchWithCategory($categories, $sourceLang, $targetLang, $source, $matchingMethod)	
Argument	$categories	The category to search
	$sourceLang	The code of source language
	$targetLang	The code of target language
	$source	A text to described into the target language
	$matchingMethod	The code of matching method
Return value	Translations found with source by matching method	
Comment	This building block can search parallel text with category	

Pictogram

Table A.78: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of pictogram service

Table A.79: search

Assertion	search($language, $word, $matchingMethod)	
Argument	$language	The code of language to search
	$word	A text to search
	$matchingMethod	The code of matching method
Return value	The result of search	
Comment	This building block can search pictogram	

Translator

Table A.80: constructor

Assertion	__construct($resourceId)	
Argument	$resourceId	ID of translation service

Table A.81: translate

Assertion	translate($sourceLang, $targetLang, $source)	
Argument	$sourceLang	The code of source language
	$targetLang	The code of target language
	$source	A text to translate
Return value	Tranlsation result	
Comment	This building block can a translate sentence	