Open-Source Platform for Language Service Sharing

Yohei Murakami, Masahiro Tanaka, Donghui Lin
National Institute of Information and Communications Technology
{yohei, mtnk, lindh}@nict.go.jp

Toru Ishida
Department of Social Informatics
Kyoto University
ishida@i.kyoto-u.ac.jp

Abstract

The Language Grid is an infrastructure for enabling users to share language services developed by language specialists and end user communities. Users can also create new services to support their intercultural/multilingual activities by composing various language services. In the Language Grid, there are several stakeholders with different incentives: service users, service providers, and a Language Grid operator. For enhancing the language service sharing, it is significant that the Language Grid can coordinate them to match their incentives. However, their incentives vary with the operation model of the Language Grid. To support the various operation models, the Language Grid should employ not a general platform dealing with various types of operation models, but a customizable platform. To this end, we have developed an open-source platform consisting of two types of components: core components and optional components. The former assures interoperability of Language Grids, while the latter provides flexibility of system configuration. It allows developers to extend the platform, and each operator to adapt the platform to his/her operation model by selecting the components. To validate the customizability, we have constructed the private Language Grid for Wikimedia using the same platform as public Language Grid.

1 Introduction

Although there are many language resources (both data and programs) on the Internet (Choukri, 2004), most intercultural collaboration activities still lack multilingual support. To overcome language barriers, we aim to construct a novel language infrastructure to improve accessibility and usability of language resources on the Internet. To this end, the Language Grid has been proposed (Ishida, 2006). The Language Grid takes a service-oriented collective intelligence approach to sharing language resources and creating new services to support intercultural/multilingual activities by combining language resources.

In previous work, many efforts have been made to combine language resources, such as UIMA (Ferrucci and Lally, 2004), GATE (Cunningham et al., 2002), D-Spin (Boehlke, 2009), Hart of Gold (Callmeier et al., 2004), and CLARIN (Varadi et al., 2008). Their purpose is to analyze a large amount of text data by linguistic processing pipelines. These pipelines consist of language resources, most of which are provided as open source by universities and research institutes. Users can thus collect language resources and freely combine them on those frameworks without considering other stakeholders.

Different from the above frameworks, the purpose of the Language Grid is to multilingualize texts for supporting intercultural collaboration by service workflows. PANACEA (Toral et al., 2011) is also a project to overcome language barriers by automatically acquiring, producing, updating, and maintaining language resources for MT by service workflow. The difference of them is that a workflow in the Language Grid combines language resources associated with complex intellectual property issues. These resources are provided by service providers who want to protect their ownership, and used by service users who need a part of the resources. Therefore, the Language Grid must coordinate these stakeholders' motivations. However, their incentives vary with the operation model of the Language Grid. To support the various operation models, we proposes open-source platform that enables developers to implement several modules and Language Grid operators to adapt their platforms to their operation models by selecting the modules. Moreover, by connecting their platforms, we can enhance language service sharing among different platforms.

The rest of this paper is organized as follows. Section 2 explains the design concept of the platform considering stakeholders' needs. Section 3 presents system architecture to satisfy requirements of the design concept. Section 4 illustrates how to extend and customize the platform. Section 5 introduces two types of system configurations to realize a public Language Grid and a private Language Grid. To validate the customizability, we show the case study of constructing the Language Grid for Wikimedia in Section 6.

2 Design Concept

The purpose of Language Grid is to accumulate language services and compose them. To realize Language Grid, system architecture should be designed to satisfy requirements of different operation models. Therefore, this section summarizes requirements of each of the operation models, and clarifies the required functions of Language Grid.

2.1 Requirements

Language Grid operators require flexibility of system configuration so that they can adapt the configuration to their two types of operation models: public Language Grid and private Language Grid. The former model is more open than the latter one. Every stakeholder is different organization in the public one, while an operator operates Language Grid for his/her use in the private one. For example, an operator operates a private Language Grid on a single cluster of machines and deploys on the cluster services, the provision policies of which are relaxed. Meanwhile, another operator operates a public Language Grid in a distributed environment by deploying services on each provider's server because the provision policies of the services are too strict. In the former case, the operator places high priority on performance of services. In the latter case, the other operator puts priority on resource security. Further, both of them may want to expand available services by allowing their users to access services on other Language Grids.

2.2 Functions

The Language Grid platform should provide the following functions extracted from the requirements in the previous subsection.

1. Modularization of system components: Language Grid operators can change implementations of each component in Language Grid platform in order to build their own Language Grids compliant with their operation models. In particular, it is necessary to switch communication components so that they can operate the platform both in a centralized environment and a distributed environment. The platform combines implementations of each component based on a configuration file defined by operators.
2. Language Grid composition: Language Grid operators can compose several Language Grids in order to increase the number of language services. The Language Grid platform realizes information sharing among Language Grids, and service invocation across Language Grids.

In designing the Language Grid architecture that provides the above functions, there are several technical constraints. For example, the architecture should be independent of service interfaces because language service interfaces vary depending on operators. In addition, the architecture should be independent of specifications of service invocations because there are several such specifications over HTTP, such as SOAP, REST, JSON, and Protocol Buffers. Moreover, it is necessary to distribute the platform to handle physically distributed services if the services are deployed on their providers' severs. In the next section, we explain the system architecture of the Language Grid platform considering these constraints.

3 System Architecture

3.1 Overview

The Language Grid architecture consists of six parts: *Service Manager*, *Service Supervisor*, *Grid Composer*, *Service Database*, *Composite Service Container*, and *Atomic Service Container*. Figure 1 (a) focuses on the first four parts, and Figure 1 (b) focuses on the last two parts.

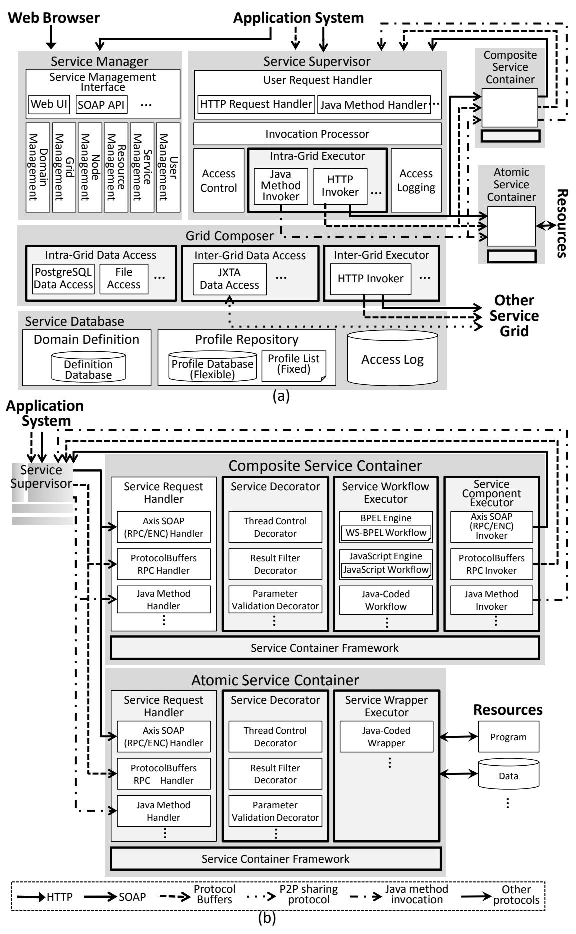

Figure 1. Language Grid Architecture

The *Service Manager* manages domain definition, grid information, node information, user information, service information and resource information registered in Language Grid. The service information includes access control settings and access logs. Since the information is registered through the *Service Manager*, it plays a front-end role for any functions other than service invocation. The *Service Supervisor* controls service invocations according to the requirements of the service providers. Before invoking the services on the *Composite Service Container* and *Atomic Service Container*, it validates whether the request satisfies providers' policies. The *Grid Composer* connects its Language Grid to other Language Grids to realize Language Grid composition for operators. The connection target is set through the *Service Manager*. The *Service Database* is a repository to store various types of information registered through the *Service Manager* and service invocation logs. The *Composite Service Container* provides composite service deployment, composite service execution, and dynamic service binding so that service users can customize services. The *Atomic Service Container* provides several utilities that service providers need in deploying atomic services.

In the remaining parts of this section, we provide the details of the *Service Manager*, *Service Supervisor*, *Grid Composer*, and *Composite/Atomic Service Container*.

3.2 Service Manager

The *Service Manager* consists of components managing various types of information necessary for Language Grid, such as domain definition, and grid, node, resource, service, and user information.

The *Domain Management* handles a domain definition that defines language service types, standard interfaces of language services, and attributes of language service profiles.

The *Grid Management* sets a target Language Grid connected by the *Grid Composer*. Based on the settings, the *Grid Composer* determines available services on other Language Grids. The *Node Management* handles information of nodes constituting its Language Grid and the connected Language Grid. Based on this information, the *Grid Composer* decides whether to save information registered on other nodes, and whether to distribute information to other nodes.

The *Resource Management* and *Service Management* handle language resource and language service information registered on Language Grid and the connected Language Grid. The information includes access control settings, service endpoints, intellectual properties associated with the language resources, and access logs. Based on this information, the *Service Supervisor* validates service invocation, locates service endpoints, and attaches intellectual property information to service responses.

Finally, the *User Management* manages user information registered on Language Grid. Based on this information, the *Service Supervisor* authenticates users' service requests.

3.3 Service Supervisor

The *Service Supervisor* controls service invocation by service users. The control covers access control, endpoint locating, load balancing, and access logging. To realize architecture independent of service specifications such as SOAP and REST, the *Service Supervisor* conducts such service invocation control based on an HTTP header.

The *User Request Handler* extracts information necessary to invoke a service from the service request over HTTP, and then authenticates the requester. The extracted information is sent to the *Invocation Processor*. Using the information, the *Invocation Processor* executes a sequence of pre-process, service invocation, post-process, and logging process. The access control is implemented as the pre-process, or the post-process.

After passing the access control, the *Intra-Grid Executor* invokes the service within its Language Grid. To invoke the service, the *Intra-Grid Executor* locates the service endpoint using the service ID. If there are multiple endpoints associated with the service ID, it chooses the endpoint with the lowest load. Finally, it invokes the service using *Java Method Invoker* implementation or *HTTP Invoker* implementation, which are selected according to the endpoint location.

3.4 Grid Composer

The *Grid Composer* not only creates a P2P grid network within its Language Grid, but also connects to other Language Grids. The former is needed to improve latency if the services are physically distributed. The latter is necessary to realize composition of Language Grids operated by different operators.

The *Intra-Grid Data Access* provides read/write interfaces for the *Service Database* within its Language Grid. In writing data, the *Intra-Grid Data Access* broadcasts the data to other nodes using a P2P network framework so that it can share the data with other nodes in the same Language Grid. As a result, service users can improve latency by sending their requests to a node located near the service. In this way, usage of the P2P network framework contributes to scalability of Language Grid.

On the other hand, the *Inter-Grid Data Access* shares various types of information with other Language Grids. The *Inter-Grid Data Access* also uses the P2P network to share information with other nodes across Language Grids. However, based on grid information registered through the *Service Manager*, the *Inter-Grid Data Access* saves only information related to the connected Language Grids.

The *Inter-Grid Executor* invokes services registered on a different Language Grid. To invoke a service across Language Grids, it replaces a requester's ID with the operator's user ID because the different Language Grid does not store user information of the requester, but rather of the operator as a Language Grid user. In addition, to control access to the services on a different Language Grid, the *Inter-Grid Executor* inserts the user ID of the requester into the request in invoking the service. By separating Language Grid that performs user authentication from the different Language Grid that performs access control, the two Language Grids do not have to share users' passwords.

3.5 Service Container

The *Service Container* executes composite services and atomic services. The *Composite Service Container* that executes composite services provides service workflow deployment and execution, and dynamic service binding. The *Atomic Service Container* that executes atomic services wraps language resources of service providers as language services with standard interfaces.

The *Service Request Handler* has multiple implementations according to service invocation protocols. If the *Service Container* is deployed on the same server as the *Service Supervisor*, the *Java Method Handler* implementation can be selected. When receiving a service request, the *Service Request Handler* receives from the *Service Container Framework* a chain of *Service Decorator*, *Service Workflow/Wrapper Executor*, and *Service Component Executor*, and executes the chain.

In invoking a component service of a composite service, the *Service Workflow Executor* can select a concrete service based on binding information included in a service request. This dynamic service binding is realized because language service interfaces are standardized.

4 Open Source Customization

The stakeholders' incentives vary depending on the operation model of Language Grid. If a Language Grid operator operates a public Language Grid, the operator promotes various users to join the Language Grid and most service providers may demand intellectual property protection. To satisfy these requirements, services are deployed on providers' servers and the Language Grid platform should provide access control functions. That is, priority is placed on security of resources. On the other hand, if a Language Grid operator operates a private Language Grid, the operator may gather language resources published under open source license to reduce the operation cost. To this end, services are aggregated and deployed on a cluster of machines, and the Language Grid platform does not have to provide user authentication and access control. That is, priority is placed on service performance.

Thus, the types of stakeholders rely on Language Grid operators. This implies that it is impossible to develop a general platform dealing with various types of operation models beforehand. Therefore, we selected open-source style customization so that each operator can adapt the platform to his/her operation model.

We have published the source codes of the Language Grid platform under an LGPL license and begun an open source project wherein each operator can freely customize the platform. In the project, the source codes are classified into a core component and optional component with different development policies because unregulated derivatives prevent interoperability of Language Grids. The specifications of core components are decided by core members in the open source community. On the other hand, the specifications of optional components can be freely changed by developers in the open source project, and derivatives can be created. This classification is done to improve the interoperability of Language Grids. As shown in Figure 1, the core components are thick-frame rectangles, and optional components thin-frame ones. In nested rectangles, outside ones are APIs and in-side ones are their implementations. These implementations can be changed.

The Intra-Grid Data Access, Inter-Grid Data Access, Intra-grid Service Executor, and Inter-Grid Service Executor are core components because they are used to communicate with other Language Grids, and they share information with other Language Grids. In addition to this, Service Decorator, Service Workflow/Wrapper Executor, Service Component Executor, and Service Container Framework in Composite/Atomic Service Container are also core components because the implementations of the components are interleaved in atomic services or composite services by the Service Container Framework. On the other hand, the Service Supervisor and Service Manager are optional components so that operators can extend them according to their operation model, because their functions are used only within the single Language Grid.

5 Configuration of the Language Grid

In this section, we introduce the system configuration of a public Language Grid and private Language Grid. In the public Language Grid, third parties are expected to join it and every stakeholder is different from the operator. In the private Language Grid, the operator uses language services for its private use. The operator often employs language resources published under open source license to reduce the operation cost and increase the performance. Moreover, the operator of the private Language Grid may connect the private Language Grid with a public Language Grid in order to use more language services on the private Language Grid.

5.1 Public Language Grid

The Department of Social Informatics in Kyoto University operates a public Language Grid. Service providers may have several provision policies to protect their language resources. Therefore, the Language Grid prefers security of language resources to performance of language services. For this reason, the Language Grid enables service providers to protect their resources on their servers, and therefore should coordinate the resources deployed on the providers' servers. To realize these functions on the Language Grid, we construct it with two different types of server nodes: the service node and core node.

The service node provides only atomic services by deploying service wrappers to standardize interfaces of language resources. The service nodes are distributed to their service providers. On the other hand, the core node controls access to services and composes services. Moreover, it communicates with other core nodes in other Language Grids to realize federated operation of the Language Grid.

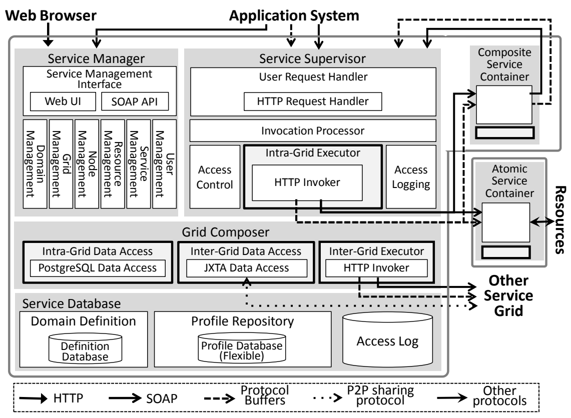

Figure 2. System Configuration of Public Language Grid

To instantiate the service node and core node, the Language Grid is configured as shown in Figure 2. The components surrounded by gray lines in the figure are deployed on the same server. The server on which the *Service Manager*, *Service Supervisor*, *Composite Service Container*, *Grid Composer*, and *Service Database* are deployed is called the core node, while that on which the *Atomic Service Container* is deployed is called the service node. This system configuration employs an HTTP invoker as the *Intra-Grid Executor* to communicate with language services on the *Atomic Service Container* physically distributed. Furthermore, the core node includes the *Inter-Grid Data Access* to share language services with other Language Grids and the *Inter-Grid Executor* to invoke language services on other Language Grids.

5.2 Private Language Grid

Unlike the system configuration of the public Language Grid, a private Language Grid prioritizing performance of language services is sometimes required.

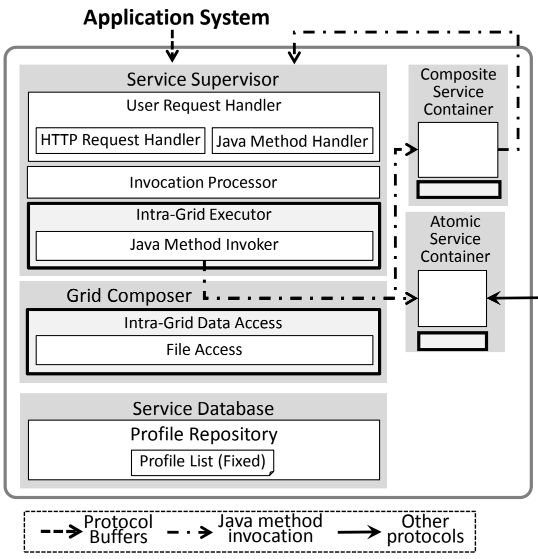

Figure 3. System Configuration of Private Language Grid

Figure 3 shows the system configuration of private Language Grid to satisfy the operator preferring performance and simplicity. The system configuration excludes the *Service Manager*, *Access Control*, and *Access Logging* components because the private Language Grid handles only language services associated with simple licenses. The *Inter-Grid Data Access* and *Inter-Grid Executor* are also removed if necessary language services can be aggregated into a single location. Moreover, the system configuration employs Java method invocation for communication between the *Service Supervisor* and *Composite/Atomic Service Container* to improve the latency of communication.

6 Case Study: Multilingual Environment for Wikimedia

In the case of employing a Language Grid to multilingualize Wikipedia, one of Wikimedia projects, by supporting multilingual discussion for Wikipedia translation community, the performance of language services should be given higher priority due to the huge amount of articles and users. Furthermore, the smaller the code size of the platform is, the more the Wikipedia operator likes it due to the low maintenance cost. We designed multilingual environment for Wikimedia considering technical requirements of the existing Wikimedia systems.

6.1 Technical Requirements

Numerous MediaWiki Extensions are available to add new features or enhance the functionality of the MediaWiki software from the users' point of view. Our goal in the development was that the actual Wikipedia community, which has a great number of users internationally, would accept the multilingual support system. From a technical point of view, as in any system development project, there are some technical requirements raised by the open-source community.

The first one is performance. Because Wikimedia projects such as Wikipedia are viewed by a great number of people every day, in particular a short response time is one of the very critical elements of the system design.

The second is usability. MediaWiki has its own look and feel, which should be consistent throughout any other MediaWiki extensions. Since Wikimedia projects are viewed by a variety of people of different age and computer skill, usability is one of the key elements to attract users.

Lastly, neutrality and independence is important for the Wikipedia community. The community does not depend too much on specific vendors, services or influence of third parties, but employs open source software and services.

6.2 System Design

Figure 4 shows the system architecture of multilingual environment using the Language Grid for Wikimedia. From the software point of view, the architecture consists of MediaWiki, the Language Grid for Wikimedia, the Language Grid Extension and Multilingual LiquidThreads Extension.

In order to develop a multilingual support system for Wikipedia discussion, we have introduced a private Language Grid, called Language Grid for Wikimedia. This employs the same system configuration as Figure 3 to prioritize performance and maintainability described in the first technical requirement. Wikimedia administrator operates the private Language Grid and aggregates several language services provided by volunteers for Wikimedia such as Microsoft and Google. Locating the Language Grid between MediaWiki and language services, we have prevented strong dependency to the language services described in the third technical requirement. Since the Language Grid is a multilingual service infrastructure, the Language Grid services should allow access via Language Grid Extension by any other MediaWiki extensions for general purposes. By unifying the access to the Language Grid, MediaWiki extensions can employ language services by invoking PHP function on the Language Grid Extension same as other MediaWiki extensions. This allows MediaWiki developers to use language services with MediaWiki's look and feel, as described in the second technical requirement.

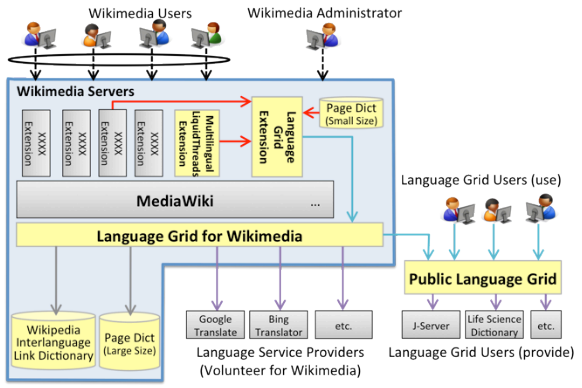

Figure 4. Multilingual Environment for Wikimedia

7 Conclusion

In this paper, we have proposed open source platform to share and compose services while satisfying various stakeholders' needs. This platform allows an operator to operate two types of Language Grid: private Language Grid and public Language Grid. The former prioritizes performance and maintainability, while the latter prioritizes intellectual property management. Moreover, combination of two types of Language Grid can complement language services on the private Language Grid with language services on the public Language Grid.

This diversity and interoperability of Language Grids are realized by classifying system architecture of Language Grid into two types of components: core components that guarantee the interoperability and optional components that provide alternative implementations. An open source project of Language Grid is expected to accelerate the diversity of Language Grid and produce other types of operation models of Language Grid.

Acknowledgments

We acknowledge the considerable support of National Institute of Information and Communications Technology, and Department of Social Informatics, Kyoto University. A part of this work was supported by Strategic Information and Communications R&D Promotion Programme from Ministry of Internal Affairs.

References

Volker Boehlke. 2009. A prototype infrastructure for D-spin-services based on a flexible multilayer architecture. Text Mining Services Conference (TMS'09)

Ulrich Callmeier, Andreas Eisele, Ulrich Schäfer, Melanie Siegel. 2004. The Deep Thought core architecture frame-work. The Fourth International Conference on Language Resources and Evaluation (LREC'04): 1205-1208.

Khalid Choukri. 2004. European Language Resources Association history and recent developments. SCALLA Working Conference KC 14/20.

Hamish Cunningham, Diana Maynard, Kalina Bontecheva, Valentin Tablan. 2002. GATE: an architecture for devel-opment of robust HLT applications. The Fortieth Annual Meeting of the Association for Computational Linguistics (ACL'02): 168-175.

David Ferrucci, Adam Lally. 2004. UIMA: an architectural approach to unstructured information processing in the corporate research environment. Journal of Natural Language Engineering 10: 327-348.

Toru Ishida. 2006. Language Grid: an infrastructure for intercultural collaboration. The IEEE/IPSJ Symposium on Applications and the Internet (SAINT'06): 96-100.

Antonio Toral, Pavel Pecina, Andy Way, Marc Poch. 2011. Towards a User-Friendly Webservice Architecture for Statistical Machine Translation in the PANACEA project. The 15th Conference of the European Association for Machine Translation (EAMT'11): 63-70.

Varadi T, Krauwer S,Wittenburg P, Wynne M, Koskenniemi K. 2008. CLARIN: common language resources and technology infrastructure. The Sixth International Conference on Lan-guage Resources and Evaluation (LREC'08): 1244-1248.