Chapter 2
Service Grid Architecture

Yohei Murakami[1], Donghui Lin[1], Masahiro Tanaka[1], Takao Nakaguchi[2], and Toru Ishida[3]

1 National Institute of Information and Communications Technology (NICT), Language Grid Project, 3-5 Hikaridai, Seika-Cho, Soraku-Gun, Kyoto, 619-0289, Japan, e-mail: {yohei, lindh, mtnk}@nict.go.jp

2 NTT Advanced Technology Corporation, 12-1 Ekimae-Honmachi, Kawasaki-Ku, Kanagawa, 210-0007, Japan, e-mail: takao.nakaguchi@ntt-at.co.jp

3 Department of Social Informatics, Kyoto University, Yoshida Honmachi, Sakyoku, Kyoto 606-8501 Japan, e-mail: 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 language services from a range of providers. Since the Language Grid takes the service-oriented collective intelligence approach, the platform requires the services management to satisfy stakeholders' needs: access control for service providers, dynamic service composition for service users, and service grid composition and system configurability for service grid operators. To realize the Language Grid, this chapter describes the design concept and the system architecture of the platform based on the service grid.

2.1 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 their intercultural/multilingual activities by combining language resources.

In previous works, 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

T. Ishida (ed.), *The Language Grid: Service-Oriented Collective Intelligence for Language Resource Interoperability*, Cognitive Technologies, DOI 10.1007/978-3-642-21178-2_2,

of which are provided as open sources 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. A workflow combines language resources associated with complex intellectual property issues, such as machine translators, parallel corpora, and bilingual dictionaries. 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. That is, it requires language service management to satisfy the following stakeholders' needs as well as language service composition for service users.

- Protecting intellectual properties of resources: Some service providers can agree on providing their services if they can retain ownership of their resources and specify the extent that service users utilize the services. For detecting fraudulent usage, they also want to know what their service is used for.
- Utilizing necessary services when needed: Service users want to utilize necessary services when needed, but not own the resources. Moreover, they may want to customize composite services for their goals by freely combining services.
- Configuring platform according to operation models: Operators create various operation models to meet stakeholders' needs. To fit their platforms to their operation models, they need to optimize system configuration. In addition, by connecting their platforms, they want to allow service users to share and invoke services on other platforms.

The above requirements are not only inherent in the Language Grid and language resources, but in any system composing services provided by others. Here we call the platform to share and compose services provided by different providers as the *service grid*, and design the service grid architecture. The service grid is a general platform independent of specific service domains so that it can be applied to a specific domain by defining services specific to the domain. For example, the Language Grid is a service grid specific to the language resource domain. Firstly, based on these requirements, this chapter clarifies functions that the service grid should provide, and explains its design concept and system architecture. Furthermore, we validate the service grid architecture by using it as basis for constructing the Language Grid.

2.2 Design Concept

The purpose of the service grid is to accumulate services and compose them. To realize the service grid, system architecture should be designed to satisfy different requirements from the stakeholders. Therefore, this section summarizes require-

ments of each of the stakeholders, and clarifies the required functions of the service grid.

2.2.1 Requirements

Service providers demand prevention of data leaks and fraudulent usage of resources because the resources represent intellectual properties. Specifically, the service providers want to deploy their services on their servers and provide their services following their provision policies, but not publish their resources under a common license, like Wikipedia. Furthermore, to check whether service users employ their services properly, they may want to know when their services are accessed and who accesses them.

On the other hand, service users prefer flexibility in customizing services and convenience in invoking the services to acquiring ownership of the resources. This is because they want to concentrate on developing application systems by reducing the resource maintenance cost. Specifically, they need to access the services through standard Web service technologies over HTTP. Moreover, they also need to create composite services freely and change service combinations.

Finally, service grid operators require flexibility of system configuration so that they can adapt the configuration to stakeholders' incentives. For example, the operators operate the service grid on a single cluster of machines by collecting services if the provision policies of the services are relaxed. Meanwhile, they operate the service grid in a distributed environment by deploying services on each provider's server if the provision policies of the services are too strict. In the former case, the operators place high priority on performance of services. In the latter case, they put priority on resource security. Further, they may want to expand available services by allowing their users to access services on other service grids.

2.2.2 Functions

The service grid platform should provide the following functions extracted from the stakeholders' requirements in the previous subsection.

(1) Service access control and monitoring: Service providers can set out their provision policies defining the terms of service use, and the platform controls access to the services according to these policies. For instance, restrictions on users who may be licensed to use the service, on the purpose for which the service may be used, and on the number of times the service may be accessed, the amount of data that may be transferred from the service, and so on. Furthermore, the platform accumulates service request messages and service response messages as access logs, and enables service providers to monitor service invocation histories and the status of

their services. Service monitoring involves monitoring events or information produced by the services; viewing services' statistics, including the number of accesses; viewing the status, or a summary, of selected services; and suspending, resuming, or terminating selected services.

(2) Service workflow execution and service dynamic binding: Service users can invoke a composite service consisting of several services on the service grid according to a service workflow. By employing standard workflow technologies such as WS-BPEL, the service grid platform allows service users to independently create and register service workflows. Besides, by standardizing service interfaces, the service grid platform enables service users to freely change the alternative services with a dynamic service binding function. This leads to various composite service invocations.

(3) Service grid composition: Service grid operators can compose several service grids in order to increase the number of services in the same domain and different domains. The service grid platform realizes information sharing among service grids, and service invocation across service grids.

(4) Modularization of system components: Service grid operators can change implementations of each component in the service grid platform in order to build their own service grids compliant with their operation models. In particular, switching communication components is necessary to 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 the operator.

In designing the service grid architecture that provides the above functions, there are several technical constraints. For example, the architecture should be independent of domains because service profiles and service interfaces vary depending on domains. 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 service grid platform considering these constraints.

2.3 System Architecture

2.3.1 Overview

The service grid architecture consists of six parts: *Service Manager*, *Service Supervisor*, *Grid Composer*, *Service Database*, *Composite Service Container*, and *Atomic Service Container*. Fig. 2.1 (a) focuses on the first four parts, and Fig. 2.1 (b) focuses on the last two parts.

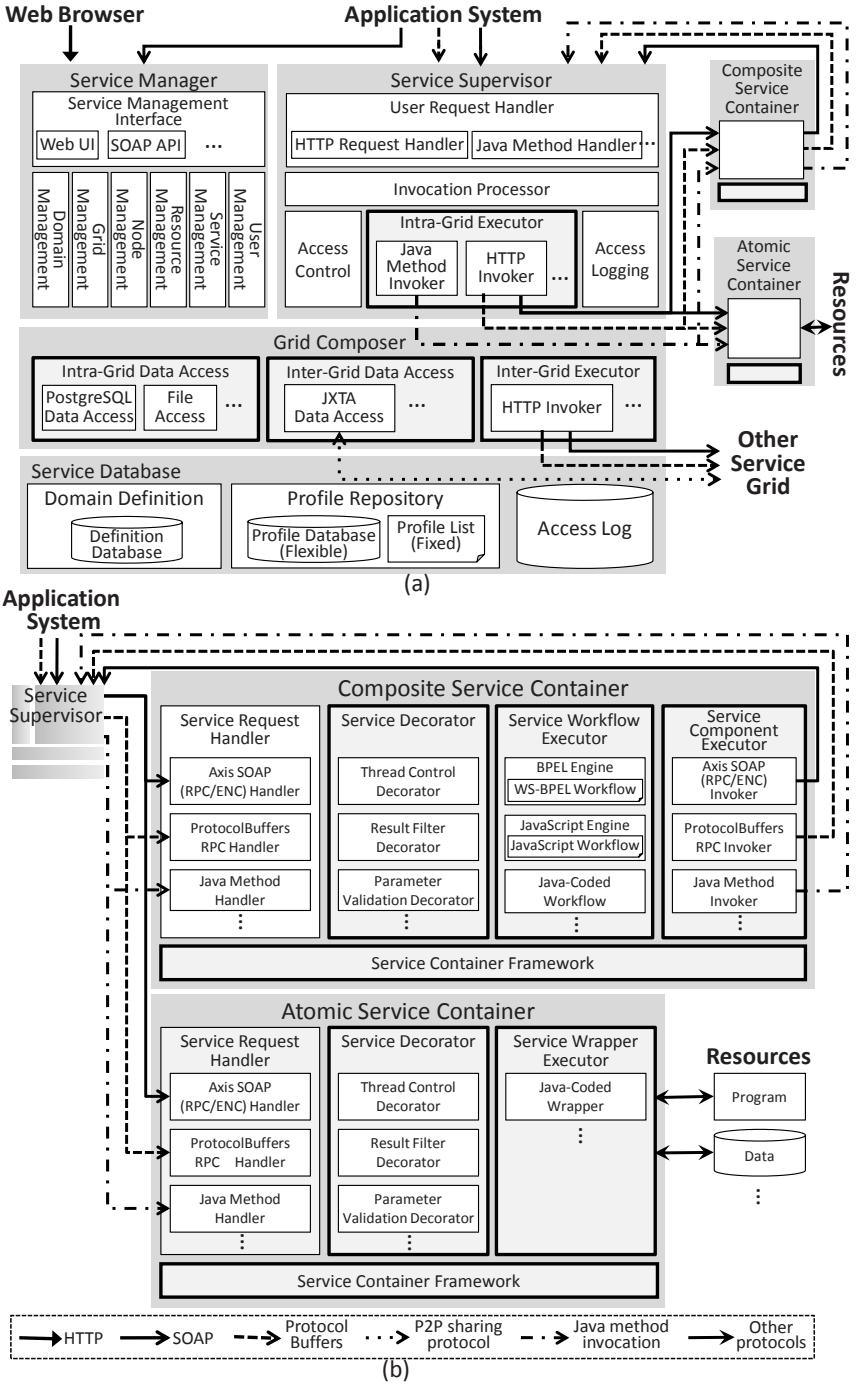

Fig. 2.1 Service grid architecture

The *Service Manager* manages domain definition, grid information, node information, user information, service information and resource information registered in the service 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 service grid to other service grids to realize service 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. By using a SOAP message handler, providers can deploy their services on their servers.

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

2.3.2 Service Manager

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

The *Domain Management* handles a domain definition that applies a general service grid to a specific domain. This component sets service types, standard interfaces of services, and attributes of service profiles according to domain definitions.

The *Grid Management* sets a target service grid connected by the *Grid Composer*. Based on the settings, the *Grid Composer* determines available services on other service grids. The *Node Management* handles node information of its service grid and the connected service 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 resource and service information registered on the service grid and the connected service grid. The information includes access control settings, service endpoints, intellectual properties associated with the 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 the service grid. Based on this information, the *Service Supervisor* authenticates users' service requests.

2.3.3 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 user who sends the request. 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 defined as the system requirements is implemented as the pre-process, or the post-process.

After passing the access control, the *Intra-Grid Executor* invokes the service within its service 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.

2.3.4 Grid Composer

The *Grid Composer* not only creates a P2P grid network within its service grid, but also connects to other service grids. The former is needed to improve latency if the services are physically distributed. The latter is necessary to realize service grid composition defined in the system requirements.

The *Intra-Grid Data Access* provides read/write interfaces for the *Service Database* in its service 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 service 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 the service grid while keeping data consistency.

On the other hand, the *Inter-Grid Data Access* shares various types of information with other service grids. The *Inter-Grid Data Access* also uses the P2P network to share information with other nodes across service grids. However, based

on grid information registered through the *Service Manager*, the *Inter-Grid Data Access* saves only information related to the connected service grids.

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

2.3.5 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 defined as system requirements. The *Atomic Service Container* that executes atomic services wraps resources of service providers as services with standard interfaces.

The *Service Request Handler* has multiple implementations according to the types of 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 resources are wrapped as services with standard interfaces in the *Atomic Service Container*.

2.4 Open Source Customization

The operation model of the service grid varies depending on the target stakeholders. If service providers demand intellectual property protection, services are deployed on their servers and the service grid platform has to provide access control. That is, priority is placed on security of resources. On the other hand, if service providers publish their resources under an open source license, services are aggregated and deployed on a cluster of machines, and the service grid platform does not provide user authentication and access control. That is, priority is placed on service performance.

The types of stakeholders assumed rely on service 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 service 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 service 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 service grids. As shown in Fig. 2.1, the core components are thick-frame rectangles, and optional components thin-frame ones. In nested rectangles, outside ones are APIs and inside ones are their implementations. These implementations can be changed. Implementations with gray labels have yet to be implemented.

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 service grids, and they share information with other service 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 service grid.

2.5 Realization of the Language Grid

In this section, we validate the service grid architecture by using it as basis for constructing the Language Grid. In particular, we focus on language service composition for language service users, language service management for language service providers, and system configurability for the Language Grid operators.

2.5.1 Language Service Composition

Among the existing research, EuroWordNet (Vossen 2004) and Global WordNet Grid (Fellbaum and Vossen 2007) are pioneering works on connecting dictionaries in different languages based on word semantics. The Language Grid, however,

aims to build an infrastructure where users can share and combine various language services. The following two types of language services are available in the Language Grid:

- Atomic service: A Web service with a standard interface that corresponds to an individual language resource. Examples include bilingual dictionaries, parallel texts, morphological analyzers, machine translators, and so on.
- Composite service: An advanced service described by a workflow that combines several atomic services. Examples include domain-specialized translation and multilingual back translation.

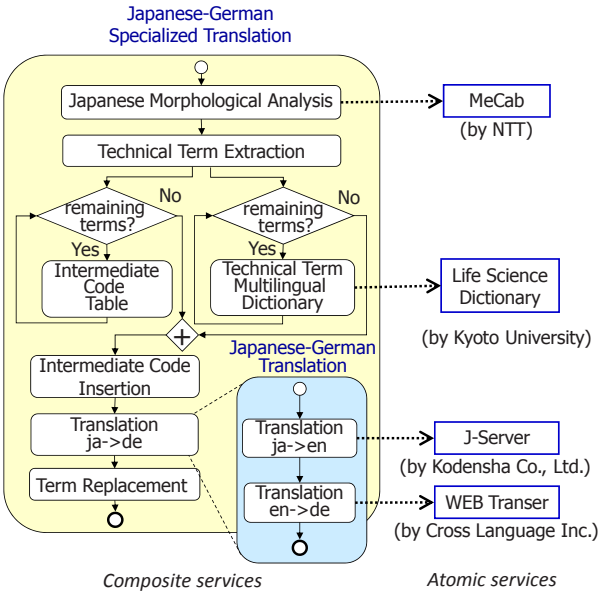

Fig. 2.2 Example of composite services

Fig. 2.2 shows a domain specialized translation workflow for improving the translation quality of technical sentences. The Language Grid uses Java-coded workflows, JavaScript, BPEL4WS (Khalaf et al. 2003), and WS-BPEL to describe the workflow. Domain specialized translation workflow consists of several component services: morphological analysis, multilingual dictionary, and translation. To invoke the composite service, service users have to bind a concrete atomic service to each component service, such as MeCab to the morphological analysis service, Life Science Dictionary to the multilingual dictionary service, and a two-hop translation service consisting of J-Server (machine translator) and WEB-Transer (machine translator) to the translation service. Service users can also invoke other combinations of concrete atomic services by changing the service bindings.

In the case of invoking composite services, the request will be sent to the Service Workflow Executor in the service grid architecture. After receiving the request, the Service Workflow Executor invokes the atomic services defined in service binding information through the Service Component Executor. If SOAP communication is used between the Service Supervisor and Composite Service Container in the service grid architecture, we can employ not only a BPEL engine but also Web service-based language resource coordination frameworks as the Service Workflow Executor, such as Heart of Gold, UIMA, and D-Spin. We have bridged Heart of Gold and the Language Grid (Bramantoro et al. 2008) and will apply the results to combine UIMA and the Language Grid.

2.5.2 Language Service Management

Language Grid Service Manager[1] is a Web application-based implementation of the service management interface in the service grid architecture. This enables Language Grid users to access various types of management functions provided by the Language Grid. In the current Language Grid, three types of access control component are implemented: access right control, access count control, and data transfer size control.

The access control function allows service providers to set access rights for each language service. Service providers who find that service users are accessing the service excessively can prohibit them from accessing the service. Moreover, service providers have two choices in publishing their services: "public mode," which permits every user by default, and "members only mode," which prohibits every user by default. Using the "members only" mode, a service provider who sells a language resource can permit a service user who purchased the language resource or its license to access the resource.

The access control function provides access constraint settings as well as access right settings. Access constraints include total access count per month, week, and day, and data transfer size (KB) per month, week, day, and request. This function enables a service provider who sells a language resource to provide limited service as a trial to service users who have not purchased it, and provide various types of service according to the fees.

In the Language Grid operated by Department of Social Informatics in Kyoto University, service providers who sell their resources realize various ways of providing their services by effectively employing the language service management (Murakami et al. 2010).

National Institute of Information and Communications Technology (called NICT hereafter) for a fee offers a concept dictionary and a bilingual dictionary, called EDR, as a whole. That is why NICT has difficulty in allowing language service users to freely employ EDR. Therefore, NICT provides a trial service of

[1] http://langrid.org/operation/service_manager/

EDR to every user by setting maximum access counts per month at 1000 counts and maximum data transfer size per request at 15 KB for the concept dictionary, and maximum access counts per month at 1000 counts and maximum data transfer size per request at 5KB for the bilingual dictionary, respectively. These constraints are configured so as to take about one year to extract all the data of the EDR. Moreover, NICT has registered the concept dictionary and the bilingual dictionary of EDR with no restrictions in the "members only mode". In this way, NICT provides unlimited EDR services only to users who purchase the EDR license.

KODENSHA Co., Ltd. (called KODENSHA hereafter) allows us to provide translation service based on J-Server, a machine translator, to third parties, if the application area of the Language Grid does not conflict with an already existing business market. Kyoto University and NICT have now purchased J-Server software to provide a translation service to other language service users. If the application area of the Language Grid conflicts with the existing market, NICT and Kyoto University prohibit the conflicting users from accessing J-Server. Furthermore, KODENSHA has registered the J-Server ASP service operated by KODENSHA in the "members only mode". This lets KODENSHA provide the latest J-Server service to only those users who purchase a J-Server ASP license. KODENSHA has registered Japanese and Simplified Chinese, Japanese and Traditional Chinese, Japanese and Korean, and Japanese and English translation separately in order to increase service variations. This enables users to purchase the language pairs that they need to use.

Kyoto University provides language service users with various language services based on machine translation software, translation ASP service, and a text-to-speech engine that Kyoto University purchased from several companies; KODENSHA Co., Ltd., Cross Language Inc., Translution, and HOYA Corporation. Since Kyoto University concluded an agreement that establishes the provision of a language resource for only non-profit use with each language resource developer, Kyoto University has to monitor for any potential abuse of the language resource. In fact, by monitoring access to the language resources, Kyoto University detected that a user accessed J-Server excessively from a specific IP address. It then obtained a contact address from the user profile and contacted the user to confirm whether it was being employed for non-profit use.

2.5.3 System Configuration of the Language Grid

In the Language Grid operated by Department of Social Informatics in Kyoto University, service providers have several provision policies to protect their language resources as described in the previous subsection. 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 must coordinate the resources deployed on the providers' servers. To equip the Language Grid with such a function, we construct it

with two different types of server nodes: the service node and core node (Murakami et al. 2006).

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.

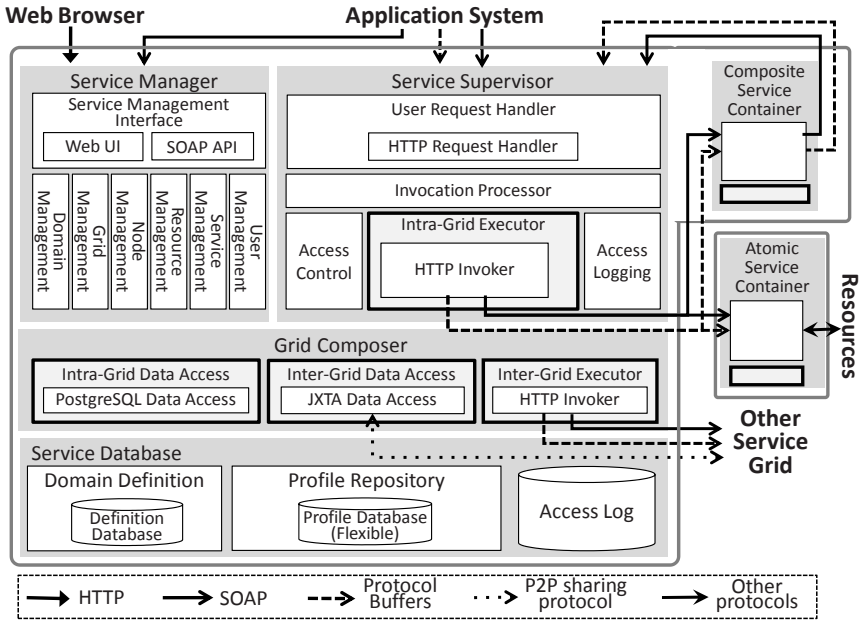

Fig. 2.3 System configuration of the Language Grid

To instantiate the service node and core node, the Language Grid is configured as shown in Fig. 2.3. 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 distribute the *Atomic Service Container* as service nodes. 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.

Unlike the above system configuration, a Language Grid prioritizing performance of language services is sometimes required. For example, in the case of employing the Language Grid to multilingualize Wikipedia articles, the performance of language services should be given higher priority due to the huge amount of ar-

ticles and users. Furthermore, the smaller the code size of the platform is, the more the Wikipedia operator likes it.

Fig. 2.4 shows the other system configuration to satisfy the operator preferring performance and simplicity. The system configuration does not include the *Service Manager*, *Access Control*, and *Access Logging* components because the Language Grid handles only language services associated with simple licenses. The *Inter-Grid Data Access* and *Inter-Grid Executor* are also removed because necessary language services will 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.

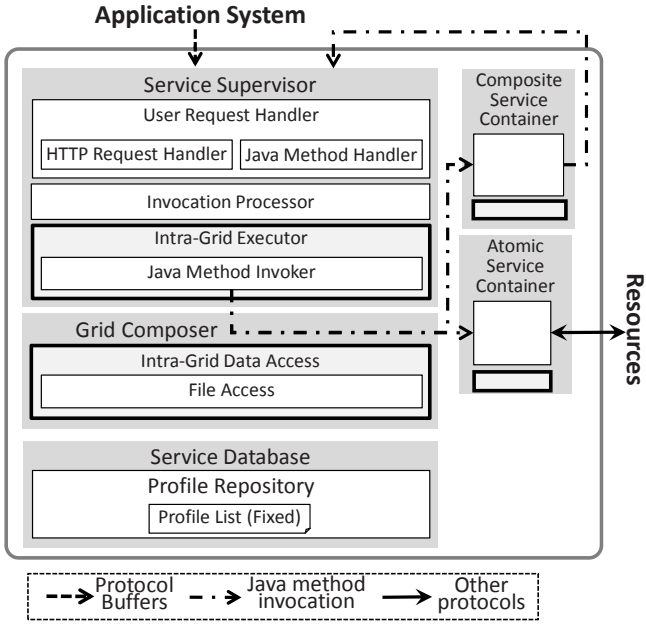

Fig. 2.4 System configuration of the Language Grid for Wikimedia

2.6 Conclusion

In this chapter, we have proposed a service grid architecture to share and compose services while satisfying stakeholders' needs. The main contributions of the proposed architecture include the following aspects.

- Protecting intellectual properties of resources: We have developed the Service Supervisor, which controls service invocations from service users. It extracts the user ID, the purpose, and so on from a service request before invoking the

service, and then checks whether the user can satisfy the provision policy of the service. In this way, service providers can provide their services within their provision policies.

- Utilizing necessary services when needed: We have developed the Composite Service Container, which binds services with service workflows at run-time. Based on service binding information in service requests, the Composite Service Container can select a concrete atomic service to invoke as a component service.
- Configuring platform according to operation models: We have developed the Grid Composer, which communicates with other service grids. It enables service providers to share their services with service users of other service grids, and service users to invoke services across service grids. Moreover, the service grid architecture allows the service grid operator to select any implementation of each main component for his/her system configuration because the main components of the service grid architecture are modularized.

We have already applied the proposed architecture to the Language Grid operated by Kyoto University. To encourage service users and providers to share and compose language services on the Internet, we need not only the proposed service grid architecture but also institutional design considering incentives among stakeholders (Ishida et al. 2008). This institutional design will be introduced in the Chapter 18.

Acknowledgments We are grateful to Nicoletta Calzolari Zamorani, Riccardo Del Gratta, Luca Dini, Alessio Bosca, Nancy Ide, and Erik Moeller for giving advice and encouragement to us. 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

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

Bramantoro A, Tanaka M, Murakami Y, Schäfer U, Ishida T (2008) A hybrid integrated architecture for language service composition. The Sixth International Conference on Web Services (ICWS'08): 345-352

Callmeier U, Eisele A, Schäfer U, Siegel M (2004) The Deep Thought core architecture framework. The Fourth International Conference on Language Resources and Evaluation (LREC'04): 1205-1208

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

Cunningham H, Maynard D, Bontecheva K, Tablan V (2002) GATE: an architecture for development of robust HLT applications. The Fortieth Annual Meeting of the Association for Computational Linguistics (ACL'02): 168-175

Fellbaum C, Vossen P (2007) Connecting the universal to the specific: towards the global grid. Intercultural Collaboration, LNCS 4568, Springer-Verlag

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

Ishida T (2006) Language Grid: an infrastructure for intercultural collaboration. The IEEE/IPSJ Symposium on Applications and the Internet (SAINT'06): 96-100

Ishida T, Nadamoto A, Murakami Y, Inaba R, Shigenobu T, Matsubara S, Hattori H, Kubota Y, Nakaguchi T, Tsunokawa E (2008) A non-profit operation model for the language grid. The First International Conference on Global Interoperability for Language Resources (ICGL'08): 114-121

Khalaf R, Mukhi N, Weerawarana S (2003) Service-oriented composition in BPEL4WS. The World Wide Web Conference (WWW'03)

Murakami Y, Ishida T, Nakaguchi T (2006) Infrastructure for language service composition. The Second International Conference on Semantics, Knowledge, and Grid (SKG'06)

Murakami Y, Lin D, Tanaka M, Nakaguchi T, Ishida T (2010) Language service management with the language grid. The International Conference on Language Resources and Evaluation (LREC'10): 3526-3531

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

Vossen P (2004) EuroWordNet: a multilingual database of autonomous and language-specific wordnets connected via an inter-lingual index. International Journal of Lexicography 17(2): 161-173