A Framework for Multi-Language Service Design with the Language Grid

Donghui Lin[†], **Yohei Murakami**[*], **and Toru Ishida**[†]
[†]Department of Social Informatics, Kyoto University,
Yoshida-Honmachi, Sakyo-ku, Kyoto, 606-8501, Japan
{lindh,ishida}@i.kyoto-u.ac.jp
[*]Unit of Design, Kyoto University,
91 Chudoji Awata-cho, Shimogyo, Kyoto 600-8815, Japan
yohei@i.kyoto-u.ac.jp

Abstract

To collect and share language resources like machine translators and dictionaries, we developed the Language Grid in 2006, a service-oriented language infrastructure on the Internet. Although we have put a lot of effort into improving the service grid technologies and collecting language services, international NPO/NGOs are struggling with the design and development of tools and systems for supporting multi-language communication in the real world by utilizing available language services. This paper proposes a framework for service design with the Language Grid by bridging the gap between language service infrastructures and multi-language systems. The proposed framework is implemented as a toolkit, Multilingual Studio, which is open to allow the users to design and develop multilingual communication services and tools in the real world.

Keywords: language service infrastructure, service design, multi-language systems

1. Introduction

The Language Grid, a service-oriented language service infrastructure on the Internet, enables the sharing of various language resources as language services around the world (Ishida, 2011). We started the research and development of the Language Grid in 2006, focusing on the technologies of service grid server software and mechanisms permitting its federated operation among multiple organizations (Murakami et al., 2012) (Murakami et al., 2014). The Language Grid has been widely used as a platform for the research of services computing (Lin et al., 2012), language resources (Lin et al., 2010), and human-computer interaction (Murakami et al., 2018).

During the past several years, the importance of language service infrastructures has been widely recognized by the research community of language resources. Typical examples of other language service initiatives include U-Compare (Kano et al., 2009), META-SHARE (Piperidis, 2012), PANACEA (Toral et al., 2011), DKPro (Eckart de Castilho and Gurevych, 2014), and the LAPPS Grid (Ide et al., 2014). Most of the work focus on the interoperability of language resources. Unfortunately, international NPO/NGOs still find it difficult to design and develop systems for supporting multi-language communication due to the complicated situations in the real world and the difficulties of customizing language resources to meet users' requirements.

Different types of users and developers have different requirements regarding language service infrastructures. To support language service developers, we have provided a series of standardized interfaces of atomic language services and composite language services with various Web service specifications in the Language Grid (Murakami et al., 2011). For end users of the Language Grid, we have developed several tools for trying out various language resources through a web browser and customizing multilingual communication components as well (Sakai et al., 2009) (Tanaka et al., 2010). To support application developers, this paper aims at providing a framework that deals with the gap between language service infrastructures and multi-language systems.

To achieve the above goal, we address the following two issues in this paper. First, since multi-language service design and development in the real world is always an iterative process (Lin and Ishida, 2013) (Lin and Ishida, 2014), language services need to be deployed as flexible components for easy invocation and composition. Second, it is necessary to enable application developers of multi-language systems to use their preferred programming languages without mastering Web service specifications and technologies.

The contributions of this paper are as follows. First, we propose a framework for multi-language service design to bridge the gap between the service-oriented language infrastructures and multi-language applications by introducing a layer of service invocation components that enables application developers to easily invoke language services in the Language Grid and flexibly manage customized local language services. Second, we implement the proposed framework as Multilingual Studio, which is a toolkit open to the users and application developers for designing and developing multi-language services in the real world.

The rest of this paper is organized as follows: In Sect. 2, we introduce development and operation of the Language Grid, and then explain the requirements for multi-language service design in Sect. 3. Section 4 describes our design concept and the proposed framework for multi-language service design with the Language Grid. Section 5 introduces the implementation of our proposed framework and the applications. Finally, the conclusion is presented in the last section.

2. The Language Grid

Developed as a service-oriented language infrastructure that enables users to share and create language resources on the Internet, the Language Grid (Ishida, 2011) is built on service grid server software, and consists of five parts: Service Manager, Service Supervisor, Grid Composer, Service Database, and Composite Service Container (Murakami et al., 2011). We have operated the Language Grid since 2007 for non-profit use and research use. To enhance the sharing of language resources, we established the federated operation of the Language Grid with three organizations in Thailand (Bangkok), Indonesia (Jakarta), and China (Urumqi) after 2010. As of February 2018, 225 language services are shared among the federated Language Grid[1]. Moreover, we started the Open Language Grid (Ishida et al., 2014) to allow users to access language services for profit use and personal use.

We have been aiming to support three types of users and developers in the Language Grid: language service developers who are familiar with Web service technologies, multi-language application developers who use the language services, and end users who use multi-language applications for various purposes.

To support language service developers, we have put effort into standardizing language services by constructing a Language Grid Ontology (Murakami et al., 2014). All the language services in the Language Grid are wrapped from language resources by standardized Web service interfaces defined by the Language Grid Ontology. Using the atomic Web services, we have also developed a series of composite services. For example, a composite machine translation service is composed of a morphological analysis service, a dictionary service, and a machine translation service. Some of the available service interface classes and examples of corresponding service types defined by the Language Grid Ontology are shown below.

- `translate`: BackTranslation, MultihopTranslation, Translation, TranslationWithTemporalDictionary
- `search`: BilingualDictionary, BilingualDictionaryWithLongestMatchSearch, ConceptDictionary, DialogCorpus, ParallelText, PictogramDictionary
- `parse`: DependencyParse
- `identify`: LanguageIdentification
- `analyze`: MorphologicalAnalysis
- `tag`: NamedEntityTagging
- `recognize`: SpeechRecognition
- `speak`: TextToSpeech
- `paraphrase`: Paraphrase
- `calculate`: SimilarityCalculation

To support end users, we developed two general tools: Language Grid Playground[2] and Language Grid Toolbox[3]. Language Grid Playground was developed as a Web application to allow users to access and try out language services in the Language Grid via a Web browser (Sakai et al., 2009). However, Language Grid Playground was designed as a showcase for displaying the Language Grid services rather than a tool for supporting intercultural collaboration (Tanaka et al., 2011). On the other hand, Language Grid Toolbox was developed for customizing multilingual communication tools by providing flexible modules of multilingual BBS, text translation, language resource management and language service creation (Tanaka et al., 2010). Language Grid Toolbox has been contributing to the support of intercultural collaboration and multi-language communication in various fields including education, medical care, and agriculture (Ishida, 2011) (Ishida, 2016).

In this paper, we focus on supporting application developers for multi-language service design with the Language Grid. To this end, we need to propose a framework that enables application developers to easily invoke and compose language services.

3. Multi-Language Service Design

So far the development and operation of the Language Grid have been focused on supporting language service developers and end users. However, application developers in international NPO/NGOs continue to struggle with available language services in creating multi-language systems. The reason lies in the fact that the complicated situations in the real fields make it difficult to customize language resources to meet users' requirements (Ishida et al., 2016). To design multi-language systems for the real world, the iterative service design approach is always applied; it consists of the following four phases (Lin and Ishida, 2014).

- **Observation:** This phase is to understand and abstract requirements for multi-language service design. Through observation in the real world, user requirements, multi-language service flows, available language services and evaluation criteria are clarified.
- **Modeling:** This phase is to model the multi-language service that can best satisfy the user requirements. Available language services are composed in this phase based on evaluation of quality of services.
- **Implementation:** There are two aspects of implementation of multi-language services: implementation of service composition, and embedding of composite services into application systems. Therefore, it is necessary to enable the application developers to easily design and test service composition.
- **Analysis:** This phase is to evaluate the multi-language service by analyzing log data and interview results based on defined evaluation criteria. Service usage information and problems with service composition are

[1]The list of available language services can be found at `http://langrid.org/service_manager/`

[2]`http://langrid.org/playground/`

[3]`http://langrid.org/tools/toolbox/`

explored for discovering more specific user requirements and improving the service process model in an iterative manner.

In many cases, human services are involved in the multi-language service design process to compensate the limitations of available language services (Lin et al., 2014). Moreover, the service design process might have many iterations when the situations in the real fields are complicated and change frequently (Lin and Ishida, 2014).

4. Design Concept

Our goal is to propose a framework for supporting application developers of multi-language service by using the Language Grid. Due to the gap between language service infrastructure and multi-language applications, we need a framework that enables language services to be deployed as flexible components for easy invocation and composition. Moreover, the proposed framework must allow developers to use their preferred programming languages without mastering knowledge of Web services and the different interfaces of the language resources.

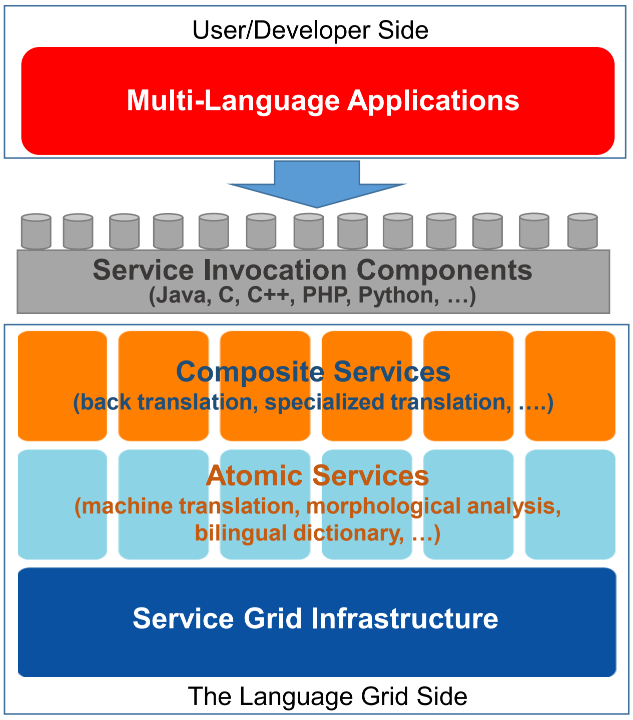
User/Developer Side
Multi-Language Applications
Service Invocation Components
(Java, C, C++, PHP, Python, ...)
Composite Services
(back translation, specialized translation, ...)
Atomic Services
(machine translation, morphological analysis, bilingual dictionary, ...)
Service Grid Infrastructure
The Language Grid Side

Figure 1: Framework for multi-language service design with the Language Grid by introducing a layer of service invocation components

4.1. The Proposed Framework

The Language Grid was originally designed with four layers (Ishida et al., 2011). The bottom layer is the service grid infrastructure that manages all requests to the Language Grid and invokes language services. The layers of atomic service and composite service enable language service developers to deploy services that are wrapped from language resources. The layer of multi-language applications include a series of collaboration tools.

To achieve our design goal, we propose a framework with a layer of service invocation components between the language services and multi-language applications as shown in Fig. 1. The purpose of introducing service invocation components is to transform Web service interfaces into libraries that support various programming languages for easy service invocation and management using APIs.

4.2. Service Invocation Components

Table 1 shows important functions provided in the service invocation components, including service authentication, atomic service invocation, composite service invocation, and local service management.

Table 1: Overview of functions in the service invocation components

Function	Description
Service Authentication	This function provides an API for specifying authentication information to access language services. Developers can either specify the default authentication information for the Language Grid or for each language service.
Atomic Service Invocation	This function provides an API for invoking each type of atomic language services that are defined by the Language Grid Ontology. Developers need to specify the service endpoint URL and values for invocation parameters when using an API for invoking an atomic service.
Composite Service Invocation	This function provides an API for invoking each type of composite language services that are defined by the Language Grid Ontology. Developers need to specify the service endpoint URL, values for invocation parameters, binding information with a service ID list of concrete atomic services that act as components of the composite service.
Local Service Management	This function provides a series of APIs for creating, updating, deleting a local language resource. It also provides the APIs for deploying the local resource as a local language service for invocation.

Table 2 shows an example of API for invoking a service with the service interface of **translate**. As described in Sect.2, the service interface class **translate** has a list of corresponding atomic/composite service types (**BackTranslation, MultihopTranslation, Translation, TranslationWithTemporalDictionary**) defined by the Language Grid Ontology. Therefore, the API example in Table 2 can be used to invoke either an atomic translation service or a composite translation service, depending on the specified service endpoint URL.

Table 2: An example of API: service interface of **translate**

API	**Service Interface: translate** String translate(String $sourceLang, String $targetLang, String $source)
Description	This API invokes a translation service that belongs to the service interface of **translate**, following the translation setting identified by three parameters: source language, target language, and the string to be translated. This API is used to invoke an atomic translation service when specified with the service endpoint URL for atomic translation (e.g. **Translation**), or a composite translation service when specified with the service endpoint URL for composite translation (e.g. **TranslationWithTemporalDictionary**). Detailed examples of invoking atomic services and composite services using this API will be introduced in Sect.5.
Parameters	**$sourceLang**: source language **$targetLang**: target language **$source**: the string to be translated
Return Value	The translation result will be returned.

Provided the APIs for the service invocation components, application developers can easily invoke various language services registered in the Language Grid and manage customized local services for flexible multi-language service design and development.

5. Implementation of Multilingual Studio

We implemented the service invocation components in the Language Grid as a toolkit called Multilingual Studio[4]; it is open and allow users to design and develop multi-language systems in the real world. As of 2018, Multilingual Studio provides Java library and PHP library for language service invocation in the Language Grid. Libraries of other programming languages can be implemented using our proposed framework as well.
With Multilingual Studio, developers can easily invoke all 225 language services registered in the Language Grid by using their preferred programming languages. Moreover, developers can also manage, deploy and invoke local services like dictionaries and parallel texts.

5.1. Service Invocation using Multilingual Studio

Figure 2 shows an example of using PHP to invoke an atomic translation service using Multilingual Studio. The following information is required for invoking an atomic service: API client of the service type, Web service description URL of the atomic service (WSDL) as the service endpoint, authentication information, and specified values for service invocation parameters used in the API. To invoke an atomic service, developers only need to write several lines of source code by using Multilingual Studio.
Moreover, developers can reuse the source code for invoking the same type of atomic service by replacing the Web service description URL of the atomic service because the service interface type is standardized in the Language Grid. Figure 3 shows an example of reusing the source code to invoke a different translation service[5].

```
;; Create the atomic language service client
$client = ClientFactory::createTranslationClient
  ('http://langrid.org/service_manager/wsdl/
    kyoto1.langrid:KyotoUJServer');

;; Specify the service authentication information
$client->setUserId('someUserId');
$client->setPassword('somePassword');

;; Set invocation parameters and get the result
$result = $client->translate(
  Language::get('en'), Language::get('ja'),
  'Have a nice day!');
```

Figure 2: Example of invoking the atomic translation service `KyotoUJServer`

```
;; Create the atomic language service client
;; Only URL is different with that of previous example
$client = ClientFactory::createTranslationClient
  ('http://langrid.org/service_manager/wsdl/
    kyoto1.langrid:GoogleTranslate');

;; Specify the service authentication information
$client->setUserId('someUserId');
$client->setPassword('somePassword');

;; Set invocation parameters and get the result
$result = $client->translate(
  Language::get('en'), Language::get('ja'),
  'Have a nice day!');
```

Figure 3: Example of invoking the atomic translation service `GoogleTranslate`

Figure 4 shows an example using Multilingual Studio to invoke a composite translation service. We use the approach of hierarchical service composition in the Language Grid by introducing the *bind* function which can assign atomic services or composite services to the service invocation in a composite service, so that we can create service composition variant virtually at runtime (Nakaguchi et al., 2016). Therefore, the only additional information that is necessary for invoking a composite service is the service binding information, which specifies concrete atomic services forming the composite service as we have described in Sect.4. In the example of the composite translation service shown in Fig. 4, the developer specifies three atomic service bindings: **BilingualDictionaryPL** for a dictionary service, **MorphologicalAnalysisPL** for a morphological service, and **TranslationPL** for a machine translation service.

[4] `http://langrid.org/developer/`

[5] Both `KyotoUJServer` and `GoogleTranslate` belong to the same service type `Translation` in the Language Grid.

```
;; Create the composite language service client
$client = ClientFactory::createTranslationClient
  ('http://langrid.org/service_manager/wsdl/kyoto1.
    langrid:TranslationCombinedWithBilingualDictionary');

;; Specify the service authentication information
$client->setUserId('someUserId');
$client->setPassword('somePassword');

;; Specify the composite service binding information
$client->addBindings(new BindingNode
  ("BilingualDictionaryPL", "KyotoTourismDictionaryDb"));
$client->addBindings(new BindingNode
  ("MorphologicalAnalysisPL", "TreeTagger"));
$client->addBindings(new BindingNode(
  "TranslationPL", "KyotoUJServer"));

;; Set invocation parameters and get the result
$result = $client->translate(
  Language::get('en'), Language::get('ja'),
  'Mount Hiei lies between Kyoto and Shiga.');
```

Figure 4: Example of invoking a composite translation service that consists of a machine translation service, a morphological analysis service and a dictionary service

Similar with invoking an atomic service, using Multilingual Studio to invoke a composite service also requires just a few lines of source code. Moreover, multi-language application developers can reuse source code to invoke the same composite service by replacing bound atomic services. By this means, source code rewriting to invoke different composite services can be dramatically reduced, which is extremely important in the design process of multi-language systems. A detailed evaluation of the effects of hierarchical service composition was reported in our previous work (Nakaguchi et al., 2016).

5.2. Applications

Multilingual Studio has been used by application developers for various purposes including scientific analysis of multi-language activities, service development in the real multi-language field, and integration with other simulation tools for participatory service design.

Terui and Hishiyama conducted the research of cross-cultural analysis by developing a multilingual case method system for global classroom environments to benefit students of different cultures and native languages (Terui and Hishiyama, 2013). To analyze the effects of self-tagging during multilingual conversational chat, Nose and Hishiyama developed a multilingual gaming simulation environment (Nose and Hishiyama, 2013). Multilingual Studio was also used to develop multi-language tools for analyzing expert knowledge transmission (Suzuki and Hishiyama, 2016) and multi-language simultaneous display in translation systems (Sato and Hishiyama, 2017).

In the multi-language field activities, a typical example is multi-language service design for the YMC-Viet project during 2011 to 2014, which is an agricultural support project for Vietnamese farmers by Japanese experts through children of the farmers (Takasaki et al., 2015) (Lin et al., 2016). In the YMC-Viet project, Multilingual Studio was effectively used for designing and simulating the composite translation service by applying the iterative service design process we have described in Sect.2 (Yamaguchi et al., 2013) (Lin and Ishida, 2014).

Moreover, Multilingual Studio has been used together with a multi-agent gaming simulation tool called MAGCruise for participatory service design of multi-language systems (Lin and Ishida, 2013) (Nakajima et al., 2015).

6. Conclusion

The development and operation of the Language Grid have been focused on supporting language service developers and end users of multi-language systems. In this paper, we aimed at bridging the gap between language service infrastructures and multi-language systems to support application developers for service design. To achieve this goal, we proposed a framework for service design for the Language Grid by introducing a layer of service invocation components that enable developers to easily invoke language services in the design of multi-language systems. The proposed framework was implemented as Multilingual Studio, which transforms Web service interfaces of the Language Grid into libraries of different programming languages. Multilingual Studio has been used for scientific analysis of multi-language activities and multi-language service design in the real world.

7. Acknowledgements

We are grateful to Dr. Yuu Nakajima, Lecture of Toho University, Japan for helping design part of the specifications for Multilingual Studio. This research was supported by a Grant-in-Aid for Scientific Research (A) (17H00759, 2017-2020) and the Kyoto University Foundation.

8. Bibliographical References

Eckart de Castilho, R. and Gurevych, I. (2014). A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, pages 1–11.

Ide, N., Pustejovsky, J., Cieri, C., Nyberg, E., Wang, D., Suderman, K., Verhagen, M., and Wright, J. (2014). The Language Application Grid. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 22–30.

Ishida, T., Murakami, Y., and Lin, D. (2011). The Language Grid: Service-oriented approach to sharing language resources. In *The Language Grid: Service-oriented collective intelligence for language resource interoperability*, pages 3–17. Springer.

Ishida, T., Murakami, Y., Lin, D., Nakaguchi, T., and Otani, M. (2014). Open Language Grid - towards a global language service infrastructure. In *Proc. of the Third ASE International Conference on Social Informatics*.

Ishida, T., Lin, D., Otani, M., Matsubara, S., Murakami, Y., Hishiyama, R., Nakajima, Y., Takasaki, T., and Mori, Y. (2016). Field-oriented service design: A multiagent approach. In *Serviceology for Designing the Future*, pages 451–463. Springer.

Ishida, T. (2011). *The Language Grid: Service-oriented collective intelligence for language resource interoperability*. Springer Science & Business Media.

Ishida, T. (2016). Intercultural collaboration and support systems: A brief history. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 3–19. Springer.

Kano, Y., Baumgartner, W., McCrohon, L., Ananiadou, S., Cohen, K., Hunter, L., and Tsujii, J. (2009). U-Compare: Share and compare text mining tools with UIMA. *Bioinformatics*, 25(15):1997–1998.

Lin, D. and Ishida, T. (2013). Participatory service design based on user-centered QoS. In *Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 01*, pages 465–472. IEEE Computer Society.

Lin, D. and Ishida, T. (2014). User-centered service design for multi-language knowledge communication. In *Serviceology for Services*, pages 309–317. Springer.

Lin, D., Murakami, Y., Ishida, T., Murakami, Y., and Tanaka, M. (2010). Composing human and machine translation services: Language Grid for improving localization processes. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation*.

Lin, D., Shi, C., and Ishida, T. (2012). Dynamic service selection based on context-aware QoS. In *Services Computing (SCC), 2012 IEEE Ninth International Conference on*, pages 641–648. IEEE.

Lin, D., Ishida, T., Murakami, Y., and Tanaka, M. (2014). QoS analysis for service composition by human and web services. *IEICE TRANSACTIONS on Information and Systems*, 97(4):762–769.

Lin, D., Ishida, T., and Otani, M. (2016). A value co-creation model for multi-language knowledge communication. In *Serviceology for Designing the Future*, pages 435–447. Springer.

Murakami, Y., Lin, D., Tanaka, M., Nakaguchi, T., and Ishida, T. (2011). Service grid architecture. *The language grid: Service-oriented collective intelligence for language resource interoperability*, pages 19–34.

Murakami, Y., Tanaka, M., Lin, D., and Ishida, T. (2012). Service grid federation architecture for heterogeneous domains. In *Services Computing (SCC), 2012 IEEE Ninth International Conference on*, pages 539–546. IEEE.

Murakami, Y., Lin, D., and Ishida, T. (2014). Service-oriented architecture for interoperability of multilanguage services. In *Towards the Multilingual Semantic Web*, pages 313–328. Springer.

Murakami, Y., Lin, D., and Ishida, T. (2018). *Services Computing for Language Resources*. Springer.

Nakaguchi, T., Murakami, Y., Lin, D., and Ishida, T. (2016). Higher-order functions for modeling hierarchical service bindings. In *Services Computing (SCC), 2016 IEEE International Conference on*, pages 798–803. IEEE.

Nakajima, Y., Hishiyama, R., and Nakaguchi, T. (2015). Multiagent gaming system for multilingual communication. In *Culture and Computing (Culture Computing), 2015 International Conference on*, pages 215–216. IEEE.

Nose, T. and Hishiyama, R. (2013). Analysis of self-tagging during conversational chat in multilingual gaming simulation. In *Future Generation Communication Technology (FGCT), 2013 Second International Conference on*, pages 81–86. IEEE.

Piperidis, S. (2012). The META-SHARE language resources sharing infrastructure: Principles, challenges, solutions. In *Proc. of the 8th International Conference on Language Resources and Evaluation Conference (LREC'12)*, pages 36–42.

Sakai, S., Gotou, M., Murakami, Y., Morimoto, S., Morita, D., Tanaka, M., and Ishida, T. (2009). Language Grid Playground: Light weight building blocks for intercultural collaboration. In *Proceedings of the 2009 international workshop on Intercultural collaboration*, pages 297–300. ACM.

Sato, M. and Hishiyama, R. (2017). An analysis of multi-language simultaneous display in the translation system. In *Computer Software and Applications Conference (COMPSAC), 2017 IEEE 41st Annual*, volume 2, pages 666–671. IEEE.

Suzuki, H. and Hishiyama, R. (2016). An analysis of expert knowledge transmission using machine translation services. In *Proceedings of the Seventh Symposium on Information and Communication Technology*, pages 352–359. ACM.

Takasaki, T., Murakami, Y., Mori, Y., and Ishida, T. (2015). Intercultural communication environment for youth and experts in agriculture support. In *Culture and Computing (Culture Computing), 2015 International Conference on*, pages 131–136. IEEE.

Tanaka, M., Murakami, Y., Lin, D., and Ishida, T. (2010). Language Grid Toolbox: Open source multi-language community site. In *Universal Communication Symposium (IUCS), 2010 4th International*, pages 105–111. IEEE.

Tanaka, M., Inaba, R., Nadamoto, A., and Shigenobu, T. (2011). Intercultural collaboration tools based on the Language Grid. In *The Language Grid: Service-oriented collective intelligence for language resource interoperability*, pages 35–49. Springer.

Terui, K. and Hishiyama, R. (2013). Multilingual case method system for cross-cultural analysis. In *Culture and Computing (Culture Computing), 2013 International Conference on*, pages 117–122. IEEE.

Toral, A., Pecina, P., Way, A., and Poch, M. (2011). Towards a user-friendly webservice architecture for statistical machine translation in the PANACEA project. In *Proc. of the 15th Conference of the European Association for Machine Translation (EAMT'11)*, pages 63–70.

Yamaguchi, T., Hishiyama, R., Kitagawa, D., Nakajima, Y., Inaba, R., and Lin, D. (2013). Evaluation of rewriting service in language translation web services workflow. In *Culture and Computing (Culture Computing), 2013 International Conference on*, pages 21–26. IEEE.