Interoperability between Service Composition and Processing Pipeline: Case Study on the Language Grid and UIMA

Mai Xuan Trang, Yohei Murakami, Donghui Lin, and Toru Ishida
Department of Social Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-Ku, Kyoto, 606-8501, Japan
trangmx@ai.soc.i.kyoto-u.ac.jp, {yohei,lindh, ishida}@i.kyoto-u.ac.jp

Abstract

Integrating language resources is a critical part in building natural language processing applications. Processing pipeline and service composition are two approaches for sharing and combining language resources. However, each approach has its drawback. While the former lacks consideration about property rights of language resources, the later is not efficient to process and transfer huge amount of data through web services. In this paper we address the issue of interoperability between two approaches to mutually complement their disadvantages. We show an integration of service composition and processing pipeline, and how the integration can be used to help developers seamlessly build NLP applications. We then present a case study that adopts the integration to integrate two representative frameworks: the Language Grid and UIMA.

1 Introduction

The creation of language resources (LRs) remains a fundamental activity in the field of language technology. The number of language resources has been increasing year by year. Based on these resources, developers build advanced Natural Language Processing (NLP) applications (hereafter referred to as the applications) such as Watson and Siri by combining some of these resources. However, it is difficult for developers to collect and combine the most suitable set of language resources in order to achieve the developers' goals.

There are two types of language resource coordination frameworks supporting developers sharing and combining language resources and tools: Framework-based processing pipeline such as GATE (Cunningham et al. 2002) and UIMA (Ferrucci el al., 2004) and framework-based service composition such as the Language Grid (Ishida, 2006). Interoperability between components in one framework is dealt by defining Common Data Exchange or standard interface for components. For example, UIMA defines Common Analysis Structure as data exchange between components, the Language Grid defines standard interfaces in a ontology for their language services (Hayashi, 2007). Interoperability among formats of two processing pipeline frameworks UIMA and GATE is explored in (Ide et al., 2009a). This paper addresses the issue of how to bridge the gap between two data structures of common data exchange format. In this work we focus on interoperability of two different types of frameworks.

Therefore, this paper realizes interoperability between those two types of frameworks to mutually complement their disadvantages. To this end, we address the following issues:

- Integration between two types of frameworks: Service composition and processing pipeline. The integration provides ability to wrap components of one framework as components of another. This will lead to more language resources and tools becoming available in both frameworks, facilitating the development process of NLP applications.

- A case study of integration two representative frameworks: The Language Grid and UIMA is implemented to realize the integration concept framework.

The remainder of this paper is organized as follows: in section 2 we will briefly discuss features of the two types of language resource coordination frameworks. The integration of the service composition and processing pipeline will be presented in section 3. We show a case study on integration between the Language Grid and UIMA in section 4. Finally, section 5 concludes this paper.

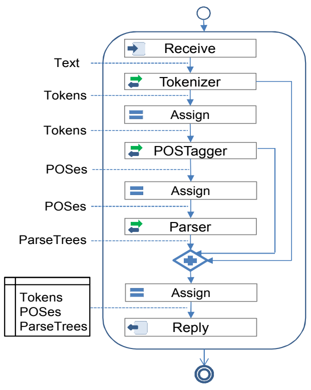


Figure 1: Service composition approach

2 Language Resource Coordination Frameworks

2.1 Service Composition Approach

In this approach, language resources are wrapped as web services that users can combine to create customized composite language services for their need. Figure 1 shows a composite service composing three language services: Tokenizer, POSTagger and Parser. Each service in the workflow is defined by an interface with input and output. Interoperability between services in a workflow is ensured by conforming interfaces of the services. Output of a previous service and input of the later service in the workflow must be compatible.

Language resources available on these frameworks are provided by variety of providers. For instance, PANACEA currently has more than 160 services provided by 11 service providers. On the Language Grid, over 170 services are provided by 140 groups from 17 countries. Providers need to protect their resources with intellectual rights, so that they can configure permission and monitor usage statistics of their resources. Service composition approach provides access control functionality to deal with this issue. This advantage encourages providers to share their language resources, increasing availability of language services.

2.2 Processing Pipeline Approach

This approach focuses on providing a setting for creating analysis pipelines, oriented towards linguistic analysis and stand-off annotation model. The purpose of these frameworks is to combine language resources to analyze huge amounts of data at the local environment.

Processing tools are combined into a pipeline to analyze documents. Each tool is defined as an annotator to annotate the document with annotations represented as stand-off annotation. The document together with annotations is formed in a Common Data Exchange Format (CDEF). The CDEF document is then exchanged between components in the pipeline. Figure 2 shows a pipeline of three annotators: Tokenizer, POSTagger, and Parser. The pipeline enriches input text with three annotation types: Token, POS, and ParseTree.

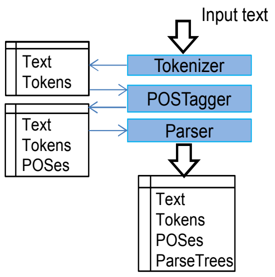


Figure 2: Processing pipeline approach

A disadvantage of this approach is the lack of access control to share language resources distributedly with intellectual rights. This limits the availability of language resources.

3 Integration of Service Composition and Processing Pipeline

3.1 Mapping Service Interface Invocation and Stand-off Annotation

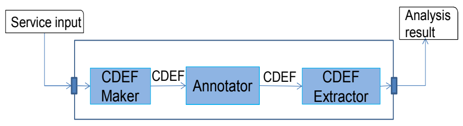
The CDEF data structure is defined based on widely used de-facto standards such as TEI (Vanhoutte, 2004), CES (Ide, 2000), and *common interface format* being developed under the context of ISO committee TC 37/SC 4 (Ide, 2009b). CDEF basically consists of two parts: one representing document text, and the other representing annotations. Figure 3 shows an example of CDEF in XML-based format:

- <*doc*>: represents the document, the *id* attribute is used to distinguish documents when a pipeline processing with multiple documents.
- <*annotations*>: represents all annotations produced by a pipeline. An annotation is described by <*annot*> tag, the *type* attribute indicates type of the annotation, two attributes *begin* and *end* define annotation's offset and the *componentID* attribute shows the annotator producing this annotation. The structure of the annotation is defined by feature structure (*fs*) tag and feature (*f*) tags.

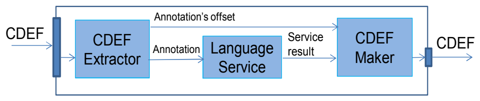
Each language service has its own interface with input and output. For an annotator in processing pipeline, we can assume that it's input

```
<?xml version="1.0" encoding="UTF-8"?>
<annotatedDoc>
  <doc id="1" mimeType="text"
        docString="Text of the document"/>
  <annotations>
     <annot type="POS" docID="1" begin="1"
             end="5" componentID="POSTager">
       <fs>
         <f name="lemma" value="Text"/>
         <f name="postag" value="noun"/>
          ...
       </fs>
     </annot>
      ...
  </annotations>
</annotatedDoc>
```

Figure 3: Structure of common data exchange



(a) Language service wrapper



(b) Language service wrapper

Figure 4: Wrappers

and output are CDEF. The mapping is defined to map input/output of language services with annotation types in CDEF. We define CDEF Maker and CDEF Extractor to conduct the mapping and create two wrappers: Language Service Wrapper and Annotator Wrapper as shown in Figure 4(a) and Figure 4(b) respectively. The former is used to wrap an annotator as a language service, the later is used to wrap a language service as an annotator:

- CDEF Extractor manipulates with CDEF to extract annotation and maps it with input/output of a language service. The Extractor uses XML parsing technique such as DOM and SAX to parse CDEF document and extract annotation. the annotation type and offset are extracted from the element *<annot>*. The annotation structure with features and values is extracted from *<fs>* node and sub-nodes *<f>s*. The Extractor then maps the annotation with a corresponding language service type which is served as input or output of a language service.
- CDEF Maker maps input/output of language services to annotation types and creates CDEF document. When wrapping an annotator as a language service with defined input and output, CDEF Maker first finds the offset of the defined input in the original text and then maps it with an annotation. Finally, it creates CDEF document from the original text and the annotation. In case of the input is text, the CDEF document is created with only *doc* part. When wrapping a language service as an annotator, CDEF Maker maps structure of the language service output with structure of a corresponding annotation type and use annotation's offset, extracted by CDEF Extractor, to create an annotation. This annotation is then added to the CDEF document.

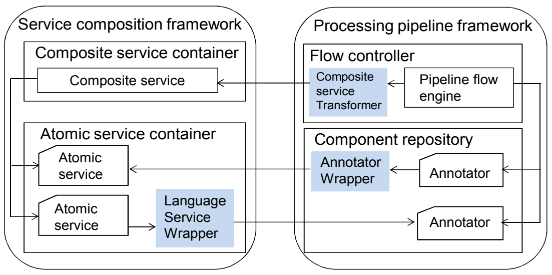


Figure 5: Integration Framework

3.2 Integration Framework

Integration framework enables users easily combine both types of components: language service and annotator. Users can use annotators to create composite services, use language services in a pipeline flow, or use both in composite services or in pipeline flows. It also provides ability to create a pipeline flow from a composite service.

Figure 5 illustrates integration of service composition and processing pipeline. A wrapper system consisting of Language Service Wrapper and Annotator Wrapper is used in this integration framework. Annotator providers use the Language Service Wrapper to wrap an annotator as a language service. This service is then shared with access control in the service composition framework. Users who have access rights to the system can invoke this service or can use this service to compose composite services. The language service providers use the Annotator Wrapper to wrap a language service into an annotator, this annotator can be executed and combined in a pipeline flow.

Language resources are shared as language services with intellectual rights, it is easy to create composite services. However, pipeline flow has better performance when processing large amounts of data compared to composite service. We define Composite Service Transformer to transfer composite services into pipeline flows. A composite service contains information about binding services. From the binding services, names, providers and sequence of language resources using in the composite service can be extracted. The transformer uses this information to build an abstract pipeline flow of these language resources. Later on, developers will negotiate with the providers to get the concrete language resources for the pipeline.

4 Case study: Integration of the Language Grid and UIMA

Using the integration framework concept, we integrate the Language Grid and UIMA. We implement two wrappers: Language Service Wrapper and Analysis Engine Wrapper. The former is used to wrap an analysis engine as a language service, while the later is used to wrap language service into an Analysis Engine. A composite service transformer is also implemented to help developers transfer composite services to UIMA flows.

CAS is common data exchanged between UIMA components. We implement CAS Maker and CAS Extractor to manipulate with CAS document and create the wrappers:

- CAS Extractor extracts annotation from CAS document and maps with input/output types of language services.
- CAS Maker maps the input/output types of language services with UIMA annotation types and creates CAS documents which are served as input/output of an analysis engine.

We use some libraries from the Language Grid and UIMA such as *jp.go.nict.langrid.client.ws_1_2.** and *org.apache.uima.** to manipulate with the Language Grid types and UIMA CAS. We also defined a new language service interface in the Language Grid to represent an analysis engine. This service interface has *analyze* operation with input is a string representing document, and output is a collection of annotations.

A mapping between UIMA annotation types and the Language Grid types is defined. We collect popular UIMA types defined for popular NLP functionalities. For each UIMA type we find a corresponding type in the Language Grid and create a mapping between these two types. For example, a *uima.annotation.Lemma* annotation can be mapped with *langrid.types.Morphem* type in the Language Grid, since these types contain similar morphological information such as partOfSpeech.

Composite Service Transformer extracts information about language resources used in a composite service and builds an UIMA flow by creating a descriptor file of the flow from the information. This process may be complex, since it is transformation between two different types of flow. To facilitate the transformer, we adapt UIMA Flow Engine into the Language Grid Composite Service Container (Murakami et al. 2011), so that users can use this engine to create composite services. This kind of composite service is much easier to be transferred into an UIMA flow.

Analysis engines are wrapped as web services and shared in the Language Grid. Developers can easily collect and combine services to build a workflow for their application. However, using web services, transformation of huge data is not efficient. After testing the workflow with small amount of data and examine the output, if it satisfies the users requirement, then this workflow is transferred to a UIMA flow. Moreover, with the integration we can create hybrid applications combining analysis engines and language services.

The integration of UIMA and the Language Grid enhances the number of language resources available in both frameworks. Especially, this increases the number of language services related NLP in the Language Grid, and increases the robustness of the Language Grid.

5 Conclusion

In this paper we proposed an integration of two types of language resource coordination frameworks: framework-based service composition and framework-based processing pipeline. The main contributions of this paper are as follows:

- The integration framework increases availability of language resources. Thus, it facilitates the process of creating applications.
- Integration of the Language Grid and UIMA is implemented to realize the framework.

In this paper, the type mapping between different frameworks is manually created. This technique is not very sufficient, due to the significant increase in number of types. Our future work will focus on using ontologies or extendable type system for a better approach of the type mapping.

Acknowledgments

This research was partly supported by a Grant-in-Aid for Scientific Research (S) (24220002, 2012-2016) from Japan Society for Promotion of Science (JSPS) and Service Science, Solutions and Foundation Integrated Research Program from JST RISTEX.

References

Bel N. 2010. Platform for Automatic, Normalized Annotation and Cos-Effective Acquisition of Language Resources for Human Language Technologies: PANACEA. *In Proceedings of the 26th Annual Congress of the Spanish Society for Natural Language Processing (SEPLN-2010).*

Cunningham H., Maynard D., Bontcheva K., and Tablan V. 2002. GATE: A framework and graphical development environment for robust NLP tools and applications. *In proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics.*

Ferrucci D., and Lally A. 2004. UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Egnineering, 10, pp. 327-348.*

Hayashi Y. 2007. Conceptual Framework of an Upper Ontology for Describing Linguistic Services. *In: Toru Ishida, Susan R. Fussel, Piek T. J. M. Vossen (Eds.): Intercultural Collaboration, LNCS 4568, Springer-Verlag, pp.31-45.*

Ide, N., Bonhomme, P., and Romary, L. 2000. An XML-based Encoding Standard for Linguistic Corpora In *Proceedings of the Second International Conference on Language Resources and Evaluation, pp. 825-830.*

Ide N., and Suderman L. 2009a. Bridging the Gaps: Interoperability for GrAF, GATE and UIMA. In *Proceeding of the Third Linguistic Annotation Workshop. Singapore, August 2009, pp. 27-34.*

Ide, N., and Romary, L. 2009b. Standards for language resources. *arXiv preprint arXiv:0909.2719.*

Ishida T. 2006. An Infrastructure for intercultural collaboration. In *IEEE/IPSJ Symposium on Applications and the Internet (SAINT-06), pp. 96100.*

Kano Y., Miwa M., Chohen K. B., Hunter L. E., Ananiadou S., and Tshujii J. 2011. U-Compare: A modular NLP workflow construction and evaluation system. *IBM Journal of Research and Development, 55(3).*

Murakami Y., Lin D., Tanaka M., Nakaguchi T., and Ishida T. 2011. Service Grid Architecture. In *The Language Grid: Service Oriented Collective Intelligence for Language Resource Interoperability, Ishida T., Ed. Springer 2011, pp. 19-34..*

Schäfer U. 2006. Middleware for Creating and Combining Multi-dimensional NLP Markup. In *Proceedings of the EACL-2006 Workshop on Multi-Dimensional Markup in Natural Language Processing. Trento, Italy, April 2006, pp. 8184.*

Vanhoutte, E. 2004. An Introduction to the TEI and the TEI Consortium. *Literary and linguistic computing*, 19(1), 9-16.