Open distributed architecture for NLP systems using services described in XML

Nicolas AUCLERC , Nick CAMPBELL & Yves LEPAGE
{nicolas.auclerc, nick.campbell, yves.lepage}@atr.co.jp
ATR 音声言語コミュニケーション研究所 / 人間情報コミュニケーション研究所 /CREST

1 Introduction

Since the new formalization of analogy, the Aleph group proposed a new open architecture based on the notion of services. Services are software components that communicate over a network. The services we developed are mainly software services, so resources used are not limited by external appliance and can be distributed and reduplicated over the network. Jini technology, which is built on Java, provides an open architecture for services. Thanks to a generic helper class included in each service, services can be defined and generated from an XML description. Blue Tongue, is an example implementation of this open architecture. Two different research groups collaborated for its realization: one in machine translation and one in expressive speech synthesis. Blue Tongue brings together their research results in one same demonstration. Thanks to the web service, we hope to demonstrate Blue Tongue with three kinds of interfaces (imode, IPaq, usual Internet browser).

2 Background

Previous implementations of applications using analogy (Auclerc 01) were designed as client/server applications because they adopted the three steps of second-generation machine translation systems: analysis, followed by transfer, followed by generation. The translation model based on analogy replaces them with several "threads" combining analogy verifications, correspondence, and resolutions of analogical equations. Translation is realized along a plurality of paths. Thus, it is not necessary for one path to wait for another path to be finished until it can starts. Consequently, the sequence analysis-transfer-generation became obsolete, and as a consequence, the client/server model was found to be no more relevant. In its place, we now use a model of software services, where requests for basic operations are sent and answered through a general blackboard.

3 Services

In client/server applications, clients and servers are different and have their own protocols. By contrast, in a service architecture, all clients and servers are services with different functionalities, and they communicate through a network by using the same set of libraries. Thank to these common libraries, a service is able to see other services through the network. A service has also the responsibility to manage its own protocol for delivery service, and has to declare its own attributes (description). As a consequence, a client-service can use other services without any foreknowledge of their implementation, nor where they reside. A service can be freely cloned on a different computer (distribution) or on the same computer (reduplication).

Services come with two different functionalities: a delivery functionality and an access functionality to other services (client). Of course, a service may be limited to one functionality. Thanks to these two basic functionalities, we can create complex service architectures like hierarchies of services as well as stand-alone services.

Services come in two kinds: software services and device services (drivers). Software services do not depend on hardware resources like external appliances. This kind of services can be cloned on the same computer or on a different computer. Unlike software services, device services manage hardware like modems, and cannot be easily cloned.

4 Our Open Service Architecture

Some current development environments offer tools to easily create services. To be able to develop an open architecture distributed over several computers, we have chosen the Jini network technology, a Java technology, which offers a development platform and supports C libraries. In our open architecture, we see a service as a generic software component, with a generic definition (like IDL: interface definition language). Thanks to this description, we clone this service by instantiating it with parameters (attributes here). We chose to write the description and the parameters of a service in XML (see Figure 1).

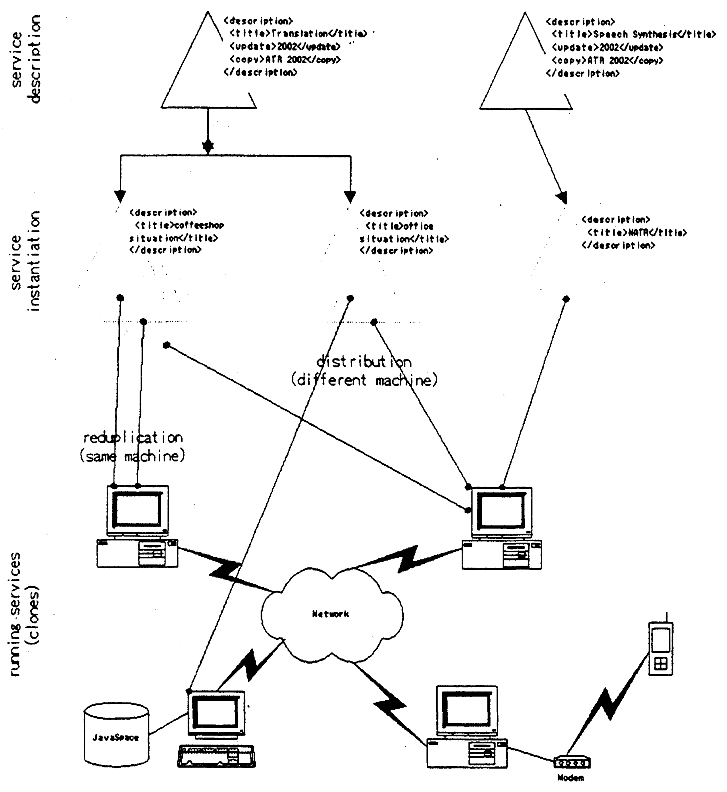

Figure 1: An open service architecture

4.1 Services as generic software components

The Jini network technology consists of an infrastructure and programming model that address the fundamental issues of how services discover and connect to each other. However, it has some limitations. To overcome these limitations, we developed a generic helper class. This add-on managed all the communications between services by using Jini common libraries and a distributed blackboard: JavaSpaces. JavaSpaces is an example of service that runs on the Jini network. By including this add-on in each service, a C or a Java component that runs as a service does not have to take care about Jini. Hence, these helper classes enable software services to be replicated or distributed by just cloning the binary. Also, as a consequence, client-services do not know which clone performs its request. Thus, the system has just to ensure that at least one clone of a service always runs on the network. For that purpose, we included a watchdog into our helper class. It ensures that a clone will be always available on the network.

4.2 Services describes in XML

In our open architecture, the XML description contains three different sections. The service description begins with the name of the service, which is unique.

The first section gives the common service attributes, plus additional specific attributes. Each of theses attributes can be redefined in the XML description of service instantiations. The common service attributes are shared by instantiations: title, update and copyright. With the attribute title, we specify the service functionality: for examples, in our machine translation mock-up, the title of the translation service changes for each domain: at home, at work, etc. (Figure 1). Additional service attributes can be defined: for example, in a modem service, a service attribute may specify the tone or impulse mode to make a phone call (see Figure 2).

The second section describes the different objects with their fields, necessary for a request and to

```
<description>
  <title>Translation</title>
  <update>2002</update>
  <copyright>ATR 2002</copyright>
</description>
<service-description>
  <Integer name="recursivity"/>
</service-description>
```

Figure 2: service attributes.

receive responses from the service (see Figure 3).

```
<declaration>
  <enum name="Lang" type="string">
   <item name="JAPANESE">japanese</item>
   <item name="ENGLISH">English</item>
  </enum>
  <message name="Totranslate">
   <class name="Source" enum="Lang">
   </class>
   <class name="target" enum="Lang">
   </class>
   <string name="sentence"></string>
  </message>
  <message name="TranslationResult">
   <string name="translation"></string>
  </message>
</declaration>
```

Figure 3: description of input/output object.

The last section enumerates C and Java components encapsulated in the service. All the input and output objects used with this method are described in the second section (see Figure 4).

```
<event>
  <action lang="C" name="translation">
  </action>
  <input message="Totranslate"></input>
  <output message="TranslationResult">
  </output>
</event>
```

Figure 4: C and Java components.

5 Blue Tongue

Blue Tongue is the result of the collaboration of two different research groups of ATR. It uses our open service architecture. It aims to show how machine translation and expressive speech synthesis can be used through portable devices like mobile phones or PDAs. Blue Tongue includes four services: a machine translation mock-up, an expressive speech processing system, a vocal server and a web service.

5.1 The machine translation mock-up

We implemented light machine translation services for specific restricted situations, like at the doctor's, at the bar, at home, etc. In these services, a unique linguistic device is used to realise the whole of translation, analogy. The idea (Lepage & 白井 01) is that new sentences, for instance *pass me the soy sauce*, can be analogically decomposed according to other sentences.

$$\textit{salt} : \begin{array}{c}\textit{pass me}\\ \textit{the salt.}\end{array} = \begin{array}{c}\textit{soy}\\ \textit{sauce}\end{array} : \begin{array}{c}\textit{pass me the}\\ \textit{soy sauce.}\end{array}$$

If the translations of the sentences in the decomposition are known, the translation of the new sentence may be obtained by solving the analogical equation (Lepage 98) formed with these translations:

塩 : 塩をとってください。 = 醤油 : x

⇒ x = 醤油をとって
ください。

The recursive application of this general principle is the core of a general translation service. This engine has several instantiations for each language pair and situation. Running a different light translation service is just a matter of changing the data files passed as arguments to the general engine. These data files contain aligned text chunks in different languages:

a bad headache	ひどい頭痛。
a bitter pain	ひどい苦痛。
bellow with pain	苦痛でうめく。
i have a stomach ache.	胃がいたい。
⋮	⋮

The use of analogy replaces the three macroscopic and specific tasks of analysis, transfer and generation, by three microscopic and generic tasks: analogy verification, taking the corresponding text chunks, and resolution of analogical equations. In this view, analogy verification and resolution of analogical equations can be executed by different clones of services, and, thus, a sentence may be translated in different ways by different clones running independently, and possibly finishing at different times. Here is the actual translation of the sentence *pass me the soy sauce*:

醤油をとってください。
醤油をとってくれませんか。
醤油をまわしてください。

5.2 The expressive speech processing system

The speech output module makes use of natural-speech corpora and re-sequencing synthesis (Springer Verlag 97) to produce a spoken rendition of the translated utterance. The input to the module is an XML sequence containing primary fields for speaker, language, mood, and text. The voice of the selected speaker is used to synthesise the speech. For certain speakers there is a choice of mood - angry, happy, sad, and normal (Speech Communication). Languages are limited to English and Japanese in the current implementation, but there are speakers for Chinese, German, and Korean available if the text field is provided with adequate markup. The text field contains the word string of the utterance to be spoken, as generated by the translation component. Currently, this is plain text in the target language, and the synthesiser makes appropriate orthography-to-pronunciation conversions as required, but if the text contains XML markup specifying, e.g., focus, loudness, speaking-rate, accentuation, pronunciation, etc., then a pre-processing module of the synthesiser is able to produce a more lively or accurate rendition of the text. Speech is produced as a waveform and returned to the original user or sent to a third party.

The present implementation is configured as a single service in the Java environment, linking to C++ subroutines, but work is in progress to allow separate services to perform individual sub-tasks of the text-to-speech synthesis process. For example, a user may prefer the prosody to be predicted by a different module from the default (to better produce dialect speech, for example) or a richer orthography-to-pronunciation, and these tasks will be performed by optional modules implemented as services in the same way as the overall system.

5.3 The vocal server

The vocal server is a modem service, which only provides service delivery. Its purpose is to make a phone call and play the requested sound file into the phone. In Blue Tongue, this device service currently only works on a Windows machine.

5.4 The web service

The web service is a software service, which is a client-service. Its purpose is to make a request to another service of Blue Tongue from HTML pages through the Java server page (JPS) and the Java common interface gateway (servlet). JSP pages are dynamic HTML pages. They are automatically adapted to the device which displays them. The device currently recognized are IPAQs and imode for mobile phones. The servlet developed will be used by the speech synthesizer services: it returns the result of the speech synthesizer as a sound stream. In Blue Tongue, this software service works on a Debian Linux system.

6 Conclusion

We have proposed an open architecture for the development, the testing and the deployment of NLP systems. Based on the Jini network technology, a Java technology; services run and cooperate on different platforms through the network. Thanks to a generic helper class included into each service, they can be defined and generated automatically from an XML description. In such a framework, different research groups with different goals can collaborate by delivering their research in the form of software services. For our machine translation mock-up, we have a hierarchy of services: verification of analogies, analogy transfer and resolution of analogical equations. For expressive speech synthesis, we have a speech synthesizer and a vocal server. This constitutes Blue Tongue a text-to-speech translation service accessible through mobile phones.

Address

`http://www.bluetongue.t2u.com/`

References

Nicolas AUCLERC & Yves LEPAGE
Tree-banking with parsing aids: an effort assessment using `Boardedit`
言語処理学会第7回年次大会, 東京大学, 2001年3月, pp. 565–568.

N. CAMPBELL, A. BLACK
Prosody and the Selection of Units for Concatenative Synthesis
Progress in Speech Synthesis, (Springer Verlag) 1997/2/1.

A. IIDA, N. CAMPBELL, M. YASUMURA
A corpus-based speech synthesis system with emotion
Speech Communication, Special Issue on Emotional Speech (in press) .

Yves LEPAGE
Solving Analogies on Words: an Algorithm
Proceedings of COLING-ACL'98, vol I, Montréal, August 1998, pp. 728–735.

Yves LEPAGE & 白井 諭
言語学的比例類推空間相似
言語処理学会第7回年次大会, 東京大学, 2001年3月, pp. 90–92.