Easy Development and Use of Dialogue Services

José Javier Durán[1], Alberto Fernández[1], Sara Rodríguez[2], Vicente Julián[3], and Holger Billhardt[1]

[1] CETINIA, University Rey Juan Carlos, Móstoles, Spain
{josejavier.duran,alberto.fernandez,holger.billhardt}@urjc.es
[2] Universidad de Salamanca, Salamanca, Spain
srg@usal.es
[3] Universidad Politécnica de Valencia, Spain
vinglada@dsic.upv.es

Abstract. We present a framework for Dialogue-Based Web Services (DBWS), i.e. services that require several message exchanges during their execution. Service development is simplified with the use of script languages and abstracting the communication layer. Service advertisements are carried out with a semantic Web Service directory with search and reputation capabilities. Execution can be performed from a mobile user interface that includes capabilities for user assistance. Our framework aims at filling the gap between services and non-IT users/experts. An example illustrates our proposal.

Keywords: Web services, Service directory, Middleware, User assistance.

1 Introduction

When humans request support from experts in some field, they do not usually exchange a single message with the problem description and an answer/solution from the expert. However, they typically engage in several interactions where the expert asks for context information, desires, etc., where questions may depend on previous answers and expert knowledge. The same approach should apply when one (or several) of the previous roles (usually the expert) are played by software agents.

Building such software systems is not an easy task. Even though many experts are able to program software pieces (knowledge bases) like rule-based, logic, scripts, etc. they usually lack skills to create software accessible by humans or agents (Web applications, Web services, software agents, …).

An additional problem is how a user can access those services. Firstly, the user needs to find a service that might be of interest. Then, the service has to be used, i.e. invoked passing the necessary parameters, possibly requiring several interactions as mentioned above.

In this paper we propose a framework focused on filling the gap between Web Services (WS) and humans, at different levels. First, the framework supports the development of WS using different scripting languages, and isolates the communication layer associated to WS from the dialogue process. Next, services are indexed in a

C. Ramos et al. (eds.), *Ambient Intelligence - Software and Applications*,
Advances in Intelligent Systems and Computing 291,
DOI: 10.1007/978-3-319-07596-9_9,

directory capable of searching services using different techniques (including free text). Also, that directory enriches search results with reputation information, in order to assist users to choose the most reliable/best service, based on other user's experiences. Finally, a generic interface is provided for service invocation, which covers mobile devices and offers an assistant that helps users with context information.

The rest of the paper is organized as follows. In section 2 we analyze other related works. Section 3 describes the architecture of our framework. Section 4 explains the development support middleware. An example of using our framework is described in section 5. We finish with conclusions and future works.

2 Related Work

Description languages and transport protocols are important parts of Web services development. There are two main technologies: REST services with JSON payload (mainly described using WADL), and SOAP (as WSDL services). The former is lightweight, easier for developers to understand, and more adaptable. The latter is more widely adopted in industry due to existing standards (WS-*) and tools [1-2]. Deployment environment is another important aspect in the development of Web services. Nowadays industry is moving towards PaaS (Platform as a Service) environments [3] in which different applications are deployed together sharing resources and its highly useful when different applications share a common structure and/or they are used in the same way (e.g. Heroku platform is running more than 3 million applications[1]).

There are different solutions focused on the creation of dynamic interfaces for Web services. Usually, the user interface is created depending on the type of service to use, or the parameters required for its execution. Some of these solutions translate a WSDL description into a Web interface that represents the different kinds of restrictions and input types using HTML widgets [4]. Others are focused on testing services by creating requests based on service definitions, but offering an interface more appropriate to software developers [5]. There are other options that integrate both a directory of services with a test user interface for such services, even including options for user feedback. In particular, there are several existing public service directories. In Table 1 we enumerate the different characteristics that we think should be present in a Web Service directory, and how they are implemented in different solutions. The first characteristic is whether the directory provides *search* capabilities. *Registry* defines whether users can register their own services or the directory is closed. A useful information for selecting services is *reputation*. There are different mechanisms for reputation, such as: rating, users' feedback as comments, or wiki-like in which users can update the description of a service in order to correct any wrong information. By *execution* we mean if it is possible to invoke the service directly from the directory web interface, without needing to develop an ad-hoc application, or if there is specific documentation of that process (e.g. example script, or unitary tests of the service). Finally, *format* represents the kind of services that can be registered (SOAP/WSDL, REST, …).

[1] `https://blog.heroku.com/archives/2013/4/24/europe-region`

Table 1. Comparison of different web service directories

	Search	Registry	Reputation	Execution	Format
Membrane SOA registry	No (list)	Yes	Rating	Yes Low-level	SOAP
WS-index.org	Text	No	Rating	No	*Unknown*
API-Hub	Text + Filters	Yes	No	No	Any
Programmable web	Text + Filters	Yes	Rating	No	Any
X Methods	No (list)	Yes	No	No	SOAP
BioCatalogue	Text + Filters + In/Out	Yes	No (wiki)	Examples	SOAP, REST
Embrace	Text	No	Comments	Unitary tests	SOAP, REST, DAS, BioMOBY

Membrane SOA Registry[2] includes a five-star rating system and a (low-level) SOAP invocation user interface, but lacks of a search capability. *WS-index.org* is a directory of web-pages related to web services, but a standard format is not applied to the entries, and most of the entries are out-dated. *API-Hub*[3] and *Programmable Web*[4] focus on API documentation and both offer text and filter-based search. *X Methods*[5] offers a WSDL-only directory, but it lacks of search capabilities and reputation mechanisms. *BioCatalogue*[6] offers a complex search mechanism able to filter by text, tags, and kind of input and/or output, but instead of offering an execution mechanism, it serves as a repository of execution examples. *Embrace*[7] is a specialized directory for medical services (support for domain description formats like DAS and BioMOBY), which offers access to unitary tests that are run in background in order to measure the reliability of the services. Despite the existence of all those tools, there is a lack of a solution that integrates all the important Web service mediation characteristics together. *Programmable Web* is the most complete regarding those characteristics, but it does not allow execution, which is only supported by *Membrane*. Moreover, they do not provide support for service development.

[2] `http://www.service-repository.com/`
[3] `http://www.apihub.com`
[4] `http://www.programmableweb.com/`
[5] `http://www.xmethods.com/`
[6] `https://www.biocatalogue.org/`
[7] `http://www.embraceregistry.net/`

3 Architecture

Fig. 1 shows our framework architecture. There are three main components: a service directory, a middleware and a Web interface.

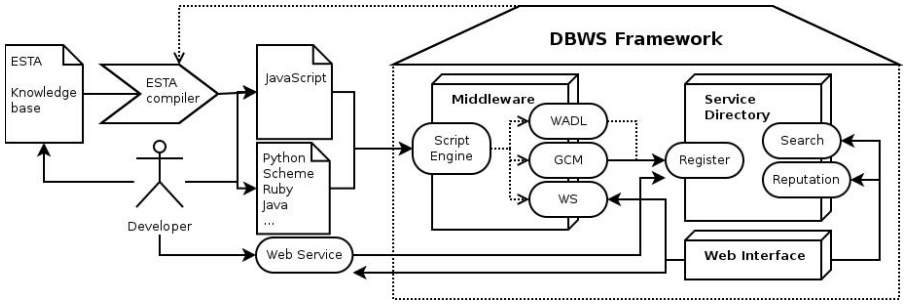

Fig. 1. Framework components

The *Service Directory* acts as a mediator (yellow pages) among services and users. Agents advertise the services they provide by registering with the directory. A service registration includes (i) a *description* of its functionality, (ii) a *grounding* specifying the endpoint where the service can be invoked, and (iii) the agent/organisation that created or owns the service (for reputation management). The service directory coordination is carried out by means of a heterogeneous service directory called Nuwa [6], and reputation management is based on a simplification of the reputation mechanism proposed by Hermoso et al. [7] for task oriented multi-agent systems. In this paper we do not focus on the description of our service directory, which can be found in the references above.

The *Development support middleware* is a set of tools that facilitate the development of dialogue-based services. A *Script Engine* takes script code and generates a Web service implementation (*WS*) and its *GCM* and *WADL* descriptions, as is detailed in next sections. Additionally, the framework includes a compiler to translate *ESTA*[8] knowledge bases into JavaScript code.

The *Web Interface* is a generic Web application that provides a human interface to search and invoke services registered with the directory, as well as providing feedback about service use.

4 Service Development Support Middleware

In order to ease the implementation and integration of Web Services using our framework, we have developed a middleware that deals with process workflow and message exchange. The advantage of this middleware is that it is possible to create a DBWS without implementing any Web functionality, since the communication part is isolated from the application itself. Also, this middleware offers a sandbox

[8] `Expert System Shell for Text Animation`

environment in which multiple applications can be run together isolated among them, and where errors are properly managed by the middleware.

The main characteristics of the proposed middleware are: (i) isolate the communication layer from the application, (ii) transform Web requests into software objects used by the application, (iii) do not impose a programming language, or paradigm, and (iv) avoid the use of special structures, or patterns, for dialogue management.

4.1 Interaction Protocol

In this section we describe the most important aspects of our framework: a workflow process for dialogue-based services, and a format for message exchange.

Workflow. In order to use dialogue-based services, a record of the interaction has to be kept. Services could be invoked in two states: initialisation and resume. During initialisation a service communicates to the client which parameters must be provided. During resume, the service takes the parameters received and returns a message that may include additional information (parameters) required to continue the execution or the result. The message content is explained next.

Message Format. We divide the dialogue message in three parts:

- *State information*: includes a set of variables representing the service state. This information is used when interacting with stateless services and must be sent to the service again in order to keep a track of the dialogue.
- *Response*: a set of messages that are sent to the client for its use. Each message can be, for example, a text, an HTML document, a picture, or an RDF document. Those messages are considered the output of the service.
- *Question*: When a service requires more information, or asks the user to wait for a time condition to be reached, a question is sent to the client. That question has a textual condition (the question), a motivation (why it is needed, and/or some semantic information about the question), a parameter name (id) (used to send back a client response), and a rule of accepted values (combination of type and values).

4.2 Script Engine Middleware

The script engine relies on the implementation of the JSR-223[9] API present in the Java runtime. This API is capable of loading applications created in different script languages (such as Java, JavaScript, Python, Scheme, Ruby, etc.), offering an abstraction of the communication between Java classes and script applications. The advantage of this approach is that it is possible to access applications independently of the programming language as long as a parser for that language is available.

[9] `http://www.jcp.org/en/jsr/detail?id=223`

4.3 Web Interface for Web Service Invocation

Since our framework defines a common interface for multiple services (the message protocol) it is possible to reuse a user interface to access different services. In our case, we have developed a user interface that covers the main aspects of our proposal: search, invocation, and feedback.

Search. The user interface accesses to the service directory, and offers two kinds of search methods: by keywords or free text. The service directory returns the matching services with their degree of match and reputation. The results are shown to the user ordered by these two parameters. The user can switch between both.

Invocation. The proposed protocol includes information needed for a dialogue stage, i.e. parameter required (question field) and response messages. The user interface shows the response messages followed by the parameter question and by a log of previous responses in the dialogue. The parameter question contains two elements: the parameter question (enriched with motivation information) and the input field. The latter is created with the most appropriate HTML input.

Feedback. During the invocation process, the current reputation score is shown, and the user can submit a feedback about the service. The feedback can include a score, a text about the user's experience and the dialogue log (e.g. for debugging).

In addition to those main functionalities, our Web interface includes a *question assistant* module that provides information related to the current question, e.g. main concept or language translation. The current implementation uses WordReference (synonyms of main terms), and a natural language question answering system (*START*[10]) to clarify the meaning of a concept or even suggest an answer for a question (e.g. if the question is asking about the value of a biochemical parameter, it will offer the textual description of that parameter from Wikipedia.org). Access to Google Translate has been implemented but it is disabled because of its commercial license.

5 Case Study

In this section we use an example to illustrate the process of adapting a specific dialogue-based application to our architecture. We chose a simple application that assists users in deciding what cocktail to make, by asking the user questions about desired ingredients or restrictions (e.g. % alcohol). Fig. 2 shows the interactions involved during a cocktail drinks' assistance. Solid arrows represent user to service messages, while dashed arrows represent service to user ones.

First, the server asks the user which is the limit of alcohol that he wants in the drink (*an enumeration*). The user answers *'< 50%'*. For clarity, we simplified the question field, omitting their description. Then, the service asks whether the user wants it with some juice (*true/false*) and the user answered *true*. Next, Vodka is offered as a possible ingredient (*true/false*), and the user agrees (*true*). Finally, the

[10] `http://start.csail.mit.edu/`

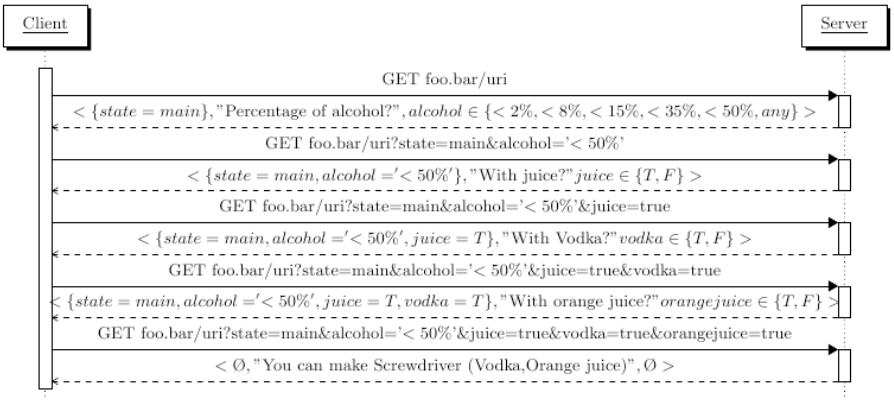

Fig. 2. HTTP message exchange between a web client and the service. Message format: <state information, response, question>

Fig. 3. Mobile Web User Interface for the application

service asks the user if he also wants something with orange juice (*true/false*), which he agrees (*true*). As a result, a cocktail is found in the knowledge base, and the service closes the dialogue with the recipe as a message with no further values to be provided.

Fig. 3 shows several snippets of the user interface, in particular obtained from a mobile phone access to the service. The first one shows the selection screen, in which a DBWS can be chosen. Next screens show questions 1 and 3 from the previous sequence diagram, including information from the question assistant (Vodka definition).

6 Conclusion

In this paper we have described a framework for developing and interacting with Dialog-Based Web Services. The main contributions of this paper are (i) a framework that supports service development by providing an integration component

for different scripting languages, which definitely facilitates Web service implementation; and (ii) a generic Web interface that supports the user to invoke such services. The framework includes a multi-language service directory with registration, search and reputation mechanisms, which we adopted from previous work.

Although our framework includes several components (Directory, Script Engine, Web Interface), developers can use their desired functionalities. Then, they might want to use only the directory functionality by registering their services. Or they might want to provide a script (or ESTA) implementation to the Script Engine so as to generate the Web service. Independently of the previous options, the Web Interface tool can be used to search and/or (v) invoke services if wanted.

The proposed framework has been implemented and we are currently working on its use for the development of a system to assist clinicians in their diagnosis. The system integrates 16 different knowledge-based medical decision support systems. Those systems are programmed in *ESTA* expert system, and have been integrated in our framework in straightforward way. We use the user interface presented in this paper to test that system. We will use that application to evaluate our framework in a real case, including the reputation mechanism with feedback provided by domain experts (clinicians).

In the future, we also plan to extend our approach to deal with asynchronous services, i.e. services that must pause their execution and resume it later (e.g. an expert is required to emit a response, or validate a conclusion).

Acknowledgement. Work partially supported by the Spanish Ministry of Science and Innovation through the project "AT" (grant CSD2007-0022; CONSOLIDER-INGENIO 2010) and by the Spanish Ministry of Economy and Competitiveness through the project iHAS (grant TIN2012-36586-C03-01/02/03).

References

1. Guinard, D., Ion, I., Mayer, S.: In search of an internet of things service architecture: REST or WS-*? A developers' perspective. In: Puiatti, A., Gu, T. (eds.) MobiQuitous 2011. LNICST, vol. 104, pp. 326–337. Springer, Heidelberg (2012)
2. Pautasso, C., Zimmermann, O., Leymann, F.: Rest- ful web services vs. "big'" web services: Making the right architectural decision. In: Proceedings of the 17th International Conference on World Wide Web, WWW 2008, pp. 805–814. ACM (2008)
3. Lawton, G.: Developing software online with platform-as-a- service technology. Computer 41(6), 13–15 (2008)
4. Kopel, M., Sobecki, J., Wasilewski, A.: Automatic web-based user interface delivery for soa-based systems. In: Bădică, C., Nguyen, N.T., Brezovan, M. (eds.) ICCCI 2013. LNCS, vol. 8083, pp. 110–119. Springer, Heidelberg (2013)
5. Bartolini, C., Bertolino, A., Marchetti, E., Polini, A.: Ws-taxi: A wsdl-based testing tool for web services. In: International Conference on Software Testing Verification and Validation, ICST 2009, pp. 326–335 (2009)
6. Fernandez, A., Cong, Z., Balta, A.: Bridging the gap between service description models in service matchmaking. Multiagent and Grid Systems 8(1), 83–103 (2012)
7. Hermoso, R., Billhardt, H., Centeno, R., Ossowski, S.: Effective use of organisational abstractions for confidence models. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS (LNAI), vol. 4457, pp. 368–383. Springer, Heidelberg (2007)