ISBN: 978-989-8704-04-7 © 2014 IADIS

A FRAMEWORK FOR DIALOGUE-BASED WEB SERVICES

José Javier Durán and Alberto Fernández
CETINIA, University Rey Juan Carlos - Móstoles, Spain

ABSTRACT

In this paper we describe a framework for dialogue-based applications that require using Web technologies, such as REST services. Not all services are executed in one-shot (e.g. pipe process). However, many applications can only be performed using a dialogue workflow. We propose a protocol for such service interaction, and offer a framework that assists in their life-cycle: programming, deployment, search, invocation, and feedback management. A running example is used to illustrate our proposal and assess how it improves the experience of service developers.

KEYWORDS

Web services, Platform as a Service, Service directory, Middleware

1. INTRODUCTION

HTTP REST (Representational State Transfer) services are getting fast adoption in current Web applications because of their simplicity both in programming and integration (Guinard et al. 2012). Despite the fact that developing such services is easy, as well as creating composed services, there is not a standard protocol for deploying dialogue-based services. These kinds of services require an interaction between the user and the server that cannot be expressed using Web service descriptions like WADL, or exchange languages like JSON.

In this paper we present a framework for deployment and use of dialogue-based applications that require using Web technologies, such as REST services. In a dialogue, a service may ask the user for additional information, and next steps depend on the nature of the information supplied. For example, a medical diagnosis service typically requires different information (e.g. analytic measures) depending on the values of other parameters (symptoms) already analysed. We cover the main issue of such applications: services do not necessarily need to be used only in one-shot workflows. However, others can only be performed using a dialogue workflow. This is the case, for example, of a medical diagnosis service, where it is not necessary to send the whole patient health records but just the requested measure.

The proposed framework covers different needs of an ecosystem for Dialogue-Based Web Services (DBWS), such as service description, registration, invocation and reputation. The main contributions of this paper are an interaction protocol, a middleware for supporting Web services development and a Web interface for searching and invoking Web services.

The rest of the paper is organised as follows. Section 2 summarises other related works. The proposed architecture is presented in section 3. Our service directory for registration, search and reputation is briefly described in section 4. Then, section 5 details the main contribution of this paper: the development support middleware. A Web user interface is described in section 6. 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 (Guinard et al 2012, Pautasso et al 2008). Deployment environment is another important aspect in the development of Web services. Nowadays industry is moving towards PaaS (Platform as a Service) environments (Lawton 2008) 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, https://blog.heroku.com/archives/2013/4/24/europe-region).

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 (Kopel et al. 2013). Others are focused on testing services by creating requests based on service definitions, but offering an interface more appropriate to software developers (Bartolini et al. 2009). 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 directories of services. *Membrane SOA Registry* (http://www.service-repository.com/) includes a five-star rating system and a SOAP invocation user interface, but lacks of a search capability. *API-Hub* (http://www.apihub.com) and *Programmable Web* (http://www.programmableweb.com/) focus on API documentation and offer text and category-based search. *Programmable Web* allows rating, but none of them include execution mechanism. Despite the existence of all those tools, there is a lack of a solution that integrates all the important Web service mediation characteristics together.

3. ARCHITECTURE

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

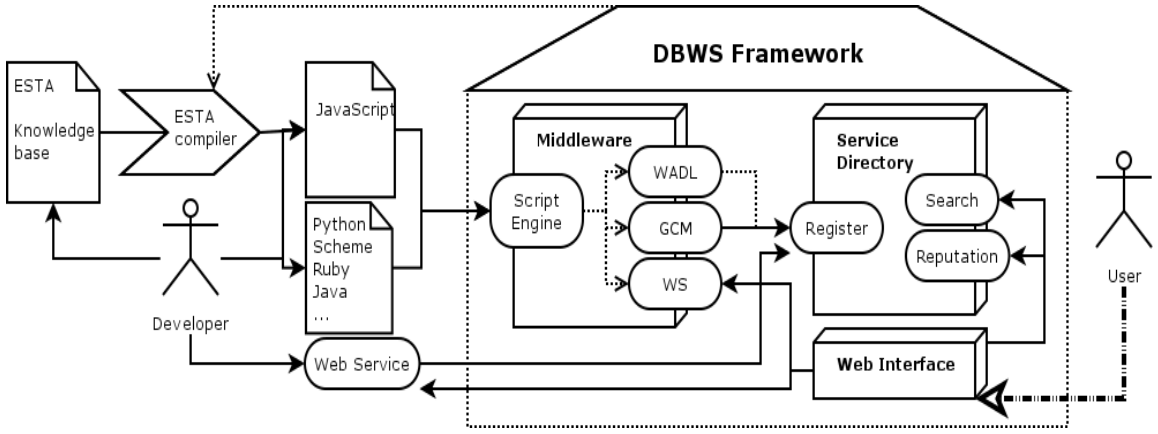

Figure 1. Framework components

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 *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.

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.

Additionally, the framework includes a compiler to translate ESTA knowledge bases into JavaScript.

4. SERVICE DIRECTORY

The service directory is a key element to coordinate providers and clients. It keeps a database of service descriptions and provides the following functionalities:

1. *Registration*: provider agents can advertise their services by providing a description of their functionality and access point. We use a heterogeneous service directory called Nuwa (Fernández et al. 2012). Heterogeneous means that, differently to most approaches, it allows different description languages both for service description and service queries. In particular, Nuwa allows several well-known existing service description languages, ranging from semantic descriptions (OWL-S, SAWSDL) to syntactic ones (WSDL, hREST), including general syntactic descriptions as text or keywords.

Nuwa manages service descriptions in a unified way allowing using a query specified in one of those languages (e.g. keywords) and obtaining as relevant results services described in a different language (e.g. OWL-S). Internally, services are represented in a unified language, named *General Common Model (GCM)*, which can be also used for registration and querying. *GCM* descriptions include typical elements found in existing service description models such as inputs, outputs, preconditions, effects, category, keywords, tag cloud (weighted keywords) and text. Service descriptions are mapped into *GCM* at registration time. Service registration also requires specifying the *grounding* of the service, i.e. the endpoint where the service can be invoked. In addition, the provider agent (e.g. developer organisation) is stored to be used by the reputation mechanism.

2. *Search*: client agents can query the directory to get a list of services matching the required functionality. As mentioned previously, different query languages can be used in our framework. Service search (or matchmaking) is provided by Nuwa. Service matchmaking in Nuwa (Cong et al. 2013) is based on the similarity of each pair of corresponding elements in their *CGM* representation (note that service advertisements are stored in *GCM* format, queries are transformed at search time). Service description components are classified into three categories: semantic elements (inputs, outputs, keywords), syntactic elements (inputs, outputs, keywords, tag clouds) and category of the service. Semantic element matching is based on the subsumption relation between the involved concepts and their location in the ontology tree. We combine the four degrees of match proposed by (Paolucci et al. 2002), with the numerical similarity function proposed by (Li et al. 2003), so as to obtain a number in [0,1]. For syntactic matching, WordNet (Miller 1995) is used as global ontology for calculating the similarity between words. Finally, degrees of match of each component are aggregated using a weighted sum.

3. *Feedback*: clients can provide feedback to the directory by evaluating their experience with the services they have interacted. Our framework includes a reputation module. When a service search is launched the relevant set obtained by the *service search* module is attached with a reputation value ([0..1]) of each service based on past experiences. We use a simplification of the reputation mechanism proposed by (Hermoso et al. 2006), for task oriented multi-agent systems. They propose a trust model for Virtual Organisations where agents play some roles in different interactions. In our case we have only two components, namely agents (organisations) and services. The reputation of a given service (*rep(s)*) is updated whenever a new evaluation (*eval(s)*) is obtained by the following equation:

$$rep(s) = \alpha \cdot rep'(s) + (1 - \alpha) \cdot eval(s)$$

where *rep'* is the reputation value previous to the update and $\alpha \in [0..1]$ is a parameter (empirically adjusted) specifying the importance of the past reputation value.

It may happen that a matching service has not been sufficiently evaluated (zero or very few evaluations). In those cases we take into account the reputation of other services run by the same organisation following Hermoso's approach, which is basically a weighted sum of the reputation of similar services provided by the same organisation.

5. 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

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:

- *Isolate the communication layer from the application*. The middleware is divided in two parts: Transport, and Script engine. The transport layer captures Web requests and translate them into a standard *model* object. Then, the script engine is invoked with that object, and the new state of the object is sent back to the requester as a response.
- *Transform Web requests into software objects used by the application.* Whenever a Web request is received the middleware transforms it into an object that contains the value of the parameters used in the dialogue. Also, the application can register parameters and their constraints (used in the dialogue). An advantage of this isolation is the possibility to create unitary tests without requiring access to the middleware.
- *Do not impose a programming language, or paradigm*. The script engine used is the standardized Java script engine (JSR-223). This script engine lets the developer include new parsers, while keeping a common API. We only require access to method invocation with a unique argument, the *object model*.
- *Avoid the use of special structures, or patterns, for dialogue management*. Thanks to the model object, applications do not require to apply a special pattern for dialogue management. Whenever an application requires access to a parameter, the script engine marks the parameter as missing and stops the execution of the script until the requester provides a value. This way of dealing with dialogue is similar to lazy programming, in the way that parameters are asked only when they are really needed by the program.

In next subsections we use a simple example application that assists users in establishing a friendship, by advising the user with some actions that can be performed in order to meet somebody.

5.1 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.

5.1.1 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 in next section.

Fig. 2 shows the interactions involved during the friendship service execution. Solid arrows represent user to service messages, while dashed arrows represent service to user ones.

After invoking the service (first interaction), the service asks the user for three variable values (next three pairs of interactions). First, the server suggests making a phone call and asks whether the friend is at home (*true/false*). The user answers *false*. For clarity, we omitted the questions in the figure. Then, the service asks whether the user wants to share a meal (*true/false*) and the user answered *false*. Finally, the service recommends asking the user's friend for a beverage and asks the user which beverage he would like to have. In this case the possible answers are enumerated (*tea, coffee, cocoa, no*). The user answered *cocoa* and the service closes the dialogue with an "enjoy drink" message with no further values to be provided.

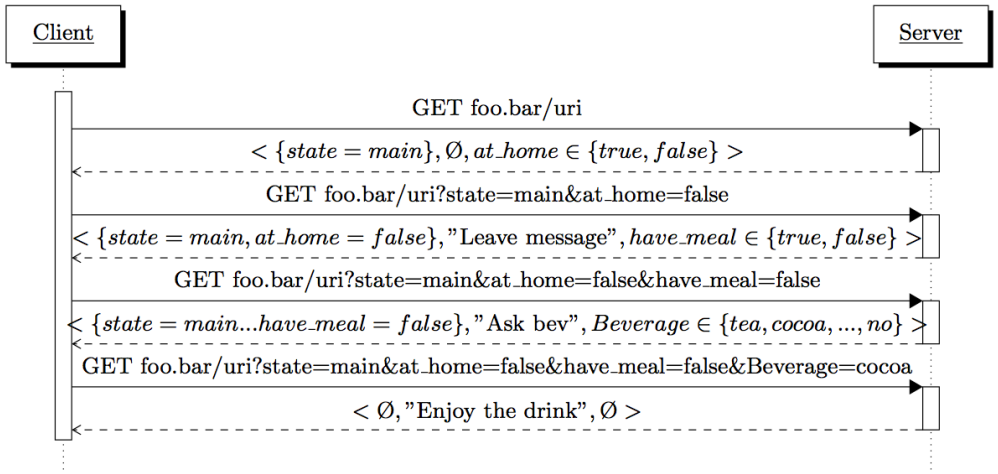

Figure 2. HTTP message exchange between a web client and the service. Format: <state information, response, question>

5.1.2 Message Format

We divide the dialogue message in three parts (Fig. 3):

```
{
  state_info: {
    at_home: "false",
    have_meal: "false",
    state : "main"
  },
  response: [
    "Leave message. Wait for a callback.",
    "Ask a for beverage"
  ],
  question: {
    id: "beverage",
    question: "Would like one of these beverages?",
    motivation: " http://dbpedia.org/ontology/Beverage",
    type: "enum",
    values : [ "tea", "coffee", "cocoa ", "no" ]
  }
}
```

Figure 3. Example of a message sent by the Web service.

- *State information*. This part includes a set of variables, with their current values, 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. Examples of information include parameters asked, internal variables, session id, or a combination of them. They might not be important for the client, except that they must be included in subsequent requests.
- *Responses*. This is 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. In general, those responses are expected to change in reply to next requests, although it is not mandatory (e.g. the response is always a unique RDF document, but with additional triples).
- *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*) (e.g. an integer a range, an enumeration of possible values, or a class/type like a date). If that field is missing the dialogue is considered finished as the system does not need any further interaction with the client.

5.2 Script Engine Middleware

The script engine relies on the implementation of the JSR-223 API (http://www.jcp.org/en/jsr/detail?id=223) present in the Java runtime. This API is capable of loading applications created in different script languages, 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. In our case, we require the presence of a *setup* method, and specifying the initial method. Both methods must accept as unique argument a *model* object.

The *model* object offers a proxy between the middleware and the script. There are different methods used to connect the script with the dialogue process:

- *name*: a descriptive name for the application.
- *language*: the default language used in the messages.
- *initialState*: establishes which method must be invoked to process the information received.
- *registerInput*: parameters are registered in the setup method with id, question, motivation, and type (e.g.: boolean, string, enumeration).
- *get*: used to access to the dialogue parameters. If a parameter is missing the execution is stopped and a response is sent to the requester asking for a value.
- *info*: used to send information to the user (*response* message field).
- *setState*: it changes the method that will be invoked in next requests, for example to continue the dialogue process from a different method instead from the initial one.
- *set*: it assigns values to dialogue parameters. It may be used to keep a trace of a dialogue status when changing state. For example, the service asks (in its initial state) the requester's birthday and, if he is minor, that age is sent to another state focused on children.

An example illustrates the use of the *model* object in Fig. 4.

```
function setup(domain) {
      domain.name = "Friendship Algorithm";
      domain.language = "en-us";
      domain.initialState = "main";
      domain.registerInput("at_home", "Make phone call. At home?", null, "boolean", null);
      domain.registerInput("have_meal", "Would like to share a meal?",
         "http://dbpedia.org/resource/Meal", "boolean", null);
      domain.registerInput("beverage", "Would like one of these beverages?",
         "http://dbpedia.org/resource/Beverage", "enum", ["tea","coffee","cocoa","no"]);
      domain.registerInput("common_interest", "Do you share an interest?",
         "http://dbpedia.org/resource/hobbies", "boolean", null);
}
function main(env) {
      if (!env.get("at_home"))  env.info("Leave a message. Wait for callback.");

      if(env.get("have_meal")){
               env.info("Dine together. Enjoy friendship!");
               return;
      }
      env.info("Ask for a beverage");
      if ( env.get("beverage") != "no" ){
               env.info("Offer "+ env.get("beverage") +". Enjoy friendship!");
               return;
      }
      if(env.get("common_interest")){
               env.info("Do that together. Enjoy friendship!");
      }else{
               env.info("Ask for a new one. Repeat the process.");
      }
      return;
}
```

Figure 4. The friendship algorithm script

The Script Engine Middleware accepts different programming languages. In particular all JSR-223 compliant languages can be used, such as Java, JavaScript, Python, Scheme, Ruby, Lua, PROLOG, etc. In addition, we have built a compiler that transforms ESTA (Expert System Shell for Text Animation) knowledge bases into JavaScript compliant with our middleware. ESTA is a rule-based language used to build Decision Support Systems (DSS).

6. 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, e.g. for a boolean or small enumeration a button for each option is shown, for long enumerations a drop list is used, etc.

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).

Fig. 5 shows several snippets of the user interface, in particular obtained from a mobile phone version.

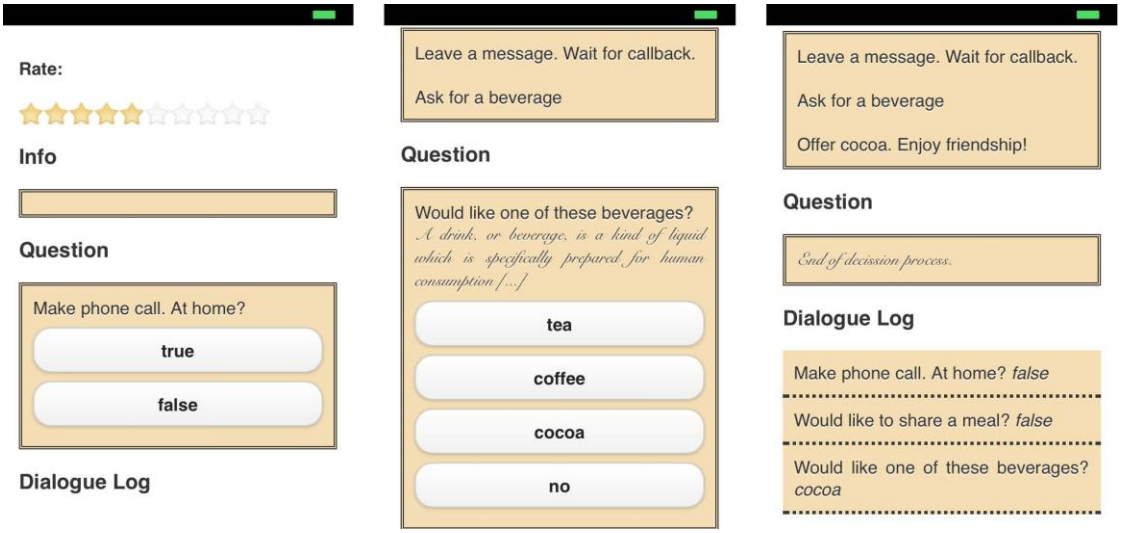

Figure 5. Mobile Web User Interface for the application

Due to mobile screen size, we show only part of the interface in each figure. The interface shows four elements: the current service rating, some information, the question and the dialogue log. Left picture corresponds to the first message sent from the service to the user. In this case there is neither information nor log to show. Middle screenshot shows the information and question fields of the last question posed by the service. The final window is shown in the right figure, including the recommendation, and the dialogue log.

7. CONCLUSION

In this paper we have described a framework for developing and interacting with Dialog-Based Web Services. We have reused a multi-language service directory that manages service advertisement and search, extended with a reputation mechanism to take into account user's feedbacks. We have proposed a protocol for interacting with this kind of services. In order to support service construction we have developed a middleware that generates web services from scripting languages. We also have developed a generic Web interface to invoke such services using our framework.

Web service developers and users can benefit from the proposed framework in different ways, using all or part of its functionality. (i) Developers can implement web services using the techniques they prefer and use only the directory functionality by registering their services providing descriptions in one of the allowed languages (section 4). Alternatively, (ii) developers can implement the functionality of their services using a scripting language (section 5) and the tool generates the Web services and directory registrations. In addition, (iii) developers (maybe not experienced programmers) can write an ESTA knowledge base and our compiler generates the script that can be used in (ii). Finally, the Web Interface is a tool that allows users to (iv) search services using different languages (e.g. keywords) and/or (v) invoke them if wanted.

We are currently working on the implementation of a system to assist clinicians in their diagnosis. The system integrates knowledge-based medical decision support systems. Those systems are programmed in ESTA expert system, and its integration in our framework has been straightforward. 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 can pause their execution and resume it later (e.g. a diagnosis service requires blood analysis tests). Web service composition is another open issue we plan to tackle.

ACKNOWLEDGEMENT

Work partially supported by the Spanish Ministry of Science and Innovation through the projects OVAMAH (grant TIN2009-13839-C03-02; co-funded by Plan E) and ”AT” (grant CSD2007-0022; CONSOLIDER-INGENIO 2010) and by the Spanish Ministry of Economy and Competitiveness through the project iHAS (grant TIN2012-36586-C03-02).

REFERENCES

Bartolini, C. et al, 2009 Ws-taxi: A wsdl-based testing tool for web services. *In Software Testing Verification and Validation, 2009. ICST ’09. International Conference on*, pp 326–335.

Cong, Z., Fernandez, A. 2013. Enabling web service discovery in heterogeneous environments. *Int. Journal of Metadata, Semantics and Ontologies*, Vol. 8, No. 2.

Fernandez, A. et al, 2012. Bridging the gap between service description models in service matchmaking. *Multiagent and Grid Systems*, Vol. 8, No. 1, pp 83–103.

Guinard, D. et al, 2012. In search of an internet of things service architecture: REST or WS-*? a developers’ perspective. *In Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pp 326–337.

Hermoso, R. et al, 2006. Effective use of organisational abstractions for confidence models. *Proceedings of the 7th international conference on Engineering societies in the agents world*, pp 368–383.

Kopel, M. et al, 2013 Automatic web-based user interface delivery for soa-based systems. In *Computational Collective Intelligence. Technologies and Applications, volume 8083 of Lecture Notes in Computer Science*, pp 110–119.

Lawton, G. 2008. Developing software online with platform-as-a-service technology. *Computer*, Vol. 41,No. 6,pp 13–15.

Li, Y. et al, 2003. An approach for measuring semantic similarity between words using multiple information sources. *IEEE Transactions on knowledge and data engineering*. pp 871–882.

Miller, G. 1995 WordNet: a lexical database for English. *Communications of the ACM* Vol. 38, No. 11, pp 39–41

Paolucci, M. et al, 2002. Semantic matching of web services capabilities. *The Semantic Web-ISWC 2002*. pp 333–347.

Pautasso, C. et al, 2008 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 ’08*, New York, USA, pp 805–814.