Using Personal Assistant Dialogs for Automatic Web Service Discovery and Execution

Márcio Fuckner[1], Jean-Paul Barthès[1] and Edson Emilio Scalabrin[2]
[1]*UMR CNRS 7253 Heudiasyc, Université de Technologie de Compiègne, Centre de Recherche de Royallieu, 60200 Compiègne, France*
[2]*Programa de Pós-Graduação em Informática - PPGIa, Pontifícia Universidade Católica do Paraná, Rua Imaculada Conceição, 1155 - Prado Velho, Curitiba - PR, Brazil*

Keywords: Web Services, Discovery, Execution, Natural Language, Personal Assistants.

Abstract: Web services and their standards have reduced the complexity of integration between heterogeneous systems, having a widespread adoption by industry. Several semantic Web services techniques concerning automatic discovery and execution are promising but still too complex to allow a large-scale adoption. Consequently, such services require end users to use traditional software engineering practices, with complex and non-intuitive interfaces in some cases. In this paper, we present an automatic approach for service discovery and execution, using the Web service descriptor as the only source. The process could be summarized as a two step process: (i) identifying candidates based on linguistic cues extracted from the Web service descriptor; (ii) extracting from the user's natural language sentences the necessary parameters to complete the action. The generated proof-of-concept shows its viability for publishing independent Web services to end-users using natural language sentences, giving only their descriptors as a source.

1 INTRODUCTION

Web services and their standards, such as the SOAP message format (XML Protocol Working Group, 2007) and the WSDL interface definition language (Web Services Description Working Group, 2007) have gained widespread adoption and reduced the complexity of integration between heterogeneous systems. Lightweight Web services, such as REST (Fielding, 2000) also have encouraged the adoption of basic and ad hoc integration scenarios. As a result, many industrial development tools adopted such standards and now can reduce the complexity of developing such Web service applications.

The semantic Web vision proposed by (Berners-Lee et al., 2001) has brought new opportunities to leverage the Web services, moving them from a syntactic to a semantic level. Standards such as OWL-S (Web-Ontology Working Group, 2004) and WSMO (WSMO Working Group, 2005) have created a common entry point to different proposals for discovery, composition and execution of services. However implementing them using a bottom-up approach is still complex, where thousands of services are already available within and outside enterprises (Vitvar et al., 2008). Allowing the leverage of even simple Web services from the syntactic level to the semantic level at a large scale still remains a major challenge. Service wrapping requires traditional software engineering steps, in some cases leading to the creation of complex and non-intuitive interfaces with end-users.

The paper of (Chai et al., 2001) presents a case study aiming at validating the usage of natural language by end users, in comparison with traditional point-and-click systems. The study reveals a reduction of 33 percent in time to execute the same task, as well as a reduction of 63 percent regarding the quantity of mouse clicks. Companies are also working on better end-users interfaces in order to improve their efficiency. An example of this behavior was presented in a recent post of MIT Technical Review Magazine (Simonite, 2012): an international bank reveals that around 65 percent of the time is used by their branch staffers with customer information desk. As a response, the bank is installing a new personal assistant using natural language through a chat window added as a mashup in their customer's Internet banking software. Thus, the customer could send sentences like "How about my investments today?" to the assistant.

Having those issues in mind, we propose an ap-

Fuckner M., Barthès J. and Emilio Scalabrin E..
Using Personal Assistant Dialogs for Automatic Web Service Discovery and Execution.
DOI: 10.5220/0004355001890198
In *Proceedings of the 9th International Conference on Web Information Systems and Technologies* (WEBIST-2013), pages 189-198
ISBN: 978-989-8565-54-9

proach for wrapping independent Web services to end-users using a natural language dialog approach. We call independent services, those services executed directly, without semantic dependencies on other services. We built a proof-of-concept system using NLP techniques, running in a multi-agent environment with dialog-based human-interaction facilities. The system allows end-users to make requests in natural language sentences representing what they are looking for. The system identifies and understands key concepts from the user's input and conducts the user towards an appropriate dialog, which is finished when a service is executed. The proof-of-concept shows the viability of usage, giving only the web service descriptors as a source.

The rest of the paper is organized as follows: In Section 2 we present some background information regarding the techniques used in our work. In Section 3 we describe in detail our approach. The outcomes of a usage example are shown in Section 4. A discussion of related work is presented in Section 5. Some conclusions presenting what we have learned from the study and proposed future work are presented in Section 6.

2 BACKGROUND

In this section we briefly describe two key areas related to this work: (i) Personal assistants, which use multidisciplinary approaches to improve the interface between a human and a software component, such as natural language processing. (ii) Web service description languages, which present key information to build the basic vocabulary and information model.

Personal Assistants. In Multi-Agent Systems, the Personal Assistant Agent (PA) is an agent built to be an assistant of one user or its master. The term "digital butler", coined by (Negroponte, 1996) is also commonly used to describe a PA. It aims to simplify the interface between a human and an agent in a multi-agent system. This approach is promoted by projects like the PAL Program (DARPA PAL Program, 2012) (Personalized Assistant that Learns) proposed by DARPA, with contributions from SRI and several other laboratories with the CALO project (Tur et al., 2010). The CALO framework provides assistant components, such as the CALO Express (CE), a lightweight personal desktop assistant that uses learning techniques to identify relevant information on the desktop, such as documents, presentations, emails and agenda. Meeting Express (ME) is another component example, designed to help a user in a meeting.

One special interest to the personal assistant approach in this work is the frequent usage of natural language (NL) written or spoken as a common interface. Natural language processing (NLP) is a field of computer science and linguistics concerned with the interactions between computers and humans. In this work we use the available tools to deal with NL and also improve the personal assistant vocabulary. Techniques such as stemming, part of speech tagging and word sense disambiguation are explored in our proof-of-concept application.

Web Service Description Languages. Several specifications were proposed during the Web service history. For the sake of clarity, we listed some important standards using (Vitvar et al., 2008) taxonomy, which proposes a two-level stack, namely a *semantic* and a *non-semantic level*.

- At a *non-semantic level*, WSDL is the de-facto standard for Web services specification. It specifies the interface, operations, the data types using XML Schema, and non-functional descriptions, such as communication protocol and physical endpoint information. Behavioral aspects can also be specified, for example, using WS-BPEL (WS-BPEL Technical Committee, 2007).
- At a *semantic level*, OWL-S and WSMO provide a framework for describing semantics for services, adopting a domain ontology, a formalism to describe the service capabilities, the effects and goals of Web services.

This clear separation between semantic and syntactical information on Web services is only conceptual, revealing in practice a thin line between them in industry. As a response to the difficulty to implement them in a large-scale, the work from W3C called Semantic Annotations for WSDL and XML Schema (SAWSDL Working Group, 2007) was initiated. It provides a model where the WSDL could be annotated with semantic information. It could be interpreted as a bridge between the non-semantic layer and the semantic one, using the WSDL as a non-marginal source of information. WSMO-Lite (Vitvar et al., 2008) creates an extension of SAWSDL, addressing the need of a concrete service ontology as its next evolutionary step.

3 OVERALL APPROACH

In this section we present a detailed description of our approach to Web service discovery and execution using a multi-agent platform. We could summarize it

in two high-level processes:

i service parsing, in order to create a knowledge representation of the service and its dialog;

ii discovery and execution using a personal assistant with natural language understanding skills.

First, we present the OMAS multi-agent platform and its components, used to implement the proposed solution. The subsequent sections describe each high-level process separately.

3.1 Multi-agent Platform

To implement our proof-of-concept, we selected OMAS[1], as a multi-agent platform (Barthès, 2011b). This platform was designed for building cognitive agents and provides several types of agents: service agents (SA), transfer agents (XA) and personal assistant agents (PA). They are organized around a single net local loop and share messages using broadcast mode (UDP). A physical loop is called a *local coterie* and transfer agents are responsible to connect different physical loops or different platforms. A set of loops in a particular application is called a *coterie*. All the communications are P2P meaning that there is no central directory or a single point of failure (SPOF). These are mandatory features when talking about service oriented and loosely coupled architectures. To illustrate the architecture, a typical configuration of an OMAS multi-agent system is shown in Figure 1 containing service agents, transfer agents and personal assistants.

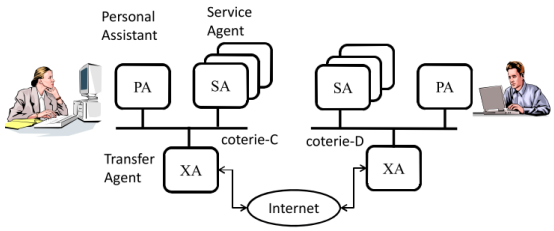

Figure 1: A typical configuration of an OMAS multi-agent system.

Personal Assistant Agents. An agent targeted to this work is the personal assistant (PA). This agent has a very specialized task: it makes an interface with a person or its master and delegates more specific tasks to other types of agents present in the environment. As a design principle, a PA has very superficial technical skills and for technical problems relies on other agents called service agents. This design approach leads to modularization and easier maintenance, since the technical expertise is distributed in separate agents. As a good design practice, service agents answer their PA and no other agents. However, a service agent can access any other agent.

A personal assistant agent in OMAS has a set of functions to deal with the user through a natural language dialog (NL). It has a standard top-level dialog that determines what the user wants to do: a request, an assertion or a command for example. The agent reacts executing the underlying task or invokes an ELIZA-like dialog to analyze the input and either produces an adequate answer or tells the master that it did not have enough information to process the input. The ELIZA-like dialog is a special dialog inspired by the work of (Weizenbaum, 1966).

Certainly, the top-level dialog is not enough to model more complex applications. Therefore, the framework allows the creation of nested sub-dialogs. Such sub-dialogs are modeled by a conversation graph, having a set of nodes representing states of the conversation. Each node contains a set of rules that apply to a fact base containing information obtained from the master's input or resulting from the analysis steps of the previous states. The fact base is similar to the fact base of a rule-based system. At a given node, applying the rules triggers either a transition to a new state or an action (e.g. a message sent to some agent) (Barthès, 2011a). These features will be explored with more details in the next sections, when an automatic dialog is created, based on the Web service structure.

3.2 Service Parsing

This process aims at extracting relevant keywords and structural information of Web services using a descriptor as an input. Each step contributes to construct a representation for service discovery and execution: for the discovery point of view, a keyword construction and the mechanism to add synonyms; for the execution point of view, a structural representation of the service and a dialog to extract the input from natural language sentences. The diagram shown in Figure 2 provides an overview of the process and also of the relevant information used and generated during each step.

3.2.1 Extract Service Information

This is the starting step, which uses a Web service descriptor as an input parameter, as shown in Figure 2. The goals of this step are: (i) extract keywords from a Web service descriptor; and (ii) build a tree representing the service. As presented in the Background Sec-

[1]The platform and the documentation are available at http://www.utc.fr/∼barthes/OMAS/.

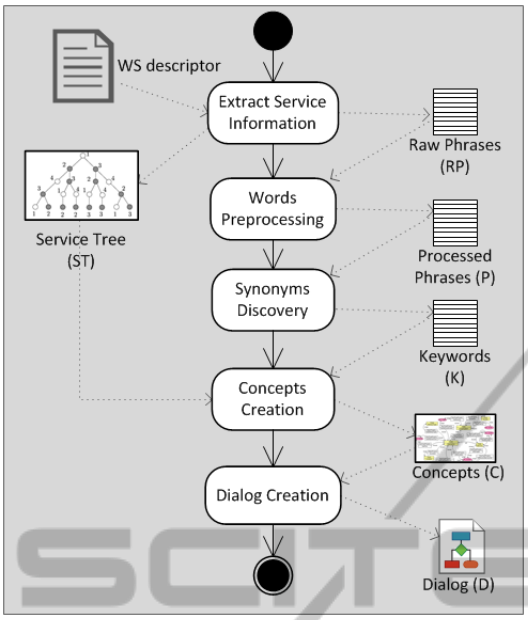

Figure 2: Service parsing diagram.

tion, there are many different standards for the Web service specification. As a result, one must build specific parsers to provide input for the goals (i) and (ii). By all means, we believe that a minimum documentation and a description of the input and output parameters are a kind of least common denominator for Web services. For example, standards such as OWL-S and WSMO allow rich semantic specifications such as capabilities, conditions and effects.

For our first prototype we created a parser for the WSDL specification, focusing on the operations and their parameters. For the operations, we extract the values from the nested documentation tags. For the parameters, we extract the input and output representation, described as XML Schema types.

The *documentation* tag value related to the operation is used to build a set of raw phrases $RP = \{rp_1, rp_2, ...rp_n\}$. A raw phrase rp in this context is interpreted as the original phrase with spaces and stop words. Figure 3 shows a fragment of a WSDL document with a documentation tag inside the operation. In this case, the resulting set RP would be: {"Get a share price quote on any listed NY stock", "Requires one valid stock identifier"}.

During the breadth-first search, a service tree ST is generated for each operation found containing a representation of the complex types in the input and output parameters. The tree will be used to build a common representation and an automatic goal-oriented dialog, using natural language sentences.

The root of the tree is the operation, with two branches representing the input and the output structure. Finding the input and output structure is not a complex task, once the complex types are described with their respective types and documentation in the WSDL structure, as can be seen in the fragment example shown in Figure 4. The WSDL specification allows the tag documentation in all element definitions. We took the advantage of this feature to improve the experience with the user and give him a user-friendly description of such pieces of information in the dialog. If the documentation is not present, the name defined in the type will be used. A resulting graphical representation of the generated tree is shown in Figure 5, with the operation *GetSharePrice* as the root of the tree, followed by the input and output branches and their respective attributes.

To sum up, this step generates for each operation a set of raw phrases RP that will be used to create the keywords and a tree ST for creating a common representation and an automatic dialog.

```
<wsdl:interface name="StockInterface">
  <wsdl:operation name="getSharePrice" ... >
    <wsdl:documentation>
       Get a share price quote on
       any listed NY stock. Requires
       one valid stock identifier.
    </wsdl:documentation>
    ...
  </wsdl:operation>
</wsdl:interface>
```

Figure 3: A documentation fragment for an operation.

```
<element name="QuoteRequest">
  <complexType>
    <all>
      <element name="id" ... >
              <documentation>Identifier<...
      </element>
      <element name="dtStock" ...>
              <documentation>Date<...
      </element>
    </all>
  </complexType>
</element>

<element name="QuoteResponse">
   <complexType>
     <all>
       <element name="stockPrice" ...
              <documentation>Price<...
       </element>
     </all>
   </complexType>
</element>
```

Figure 4: Types fragment.

3.2.2 Words Preprocessing

After extracting the service information, a set of raw

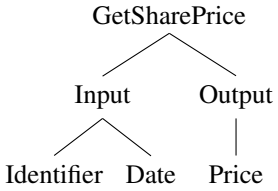

Figure 5: Generated tree.

phrases $RP = \{rp_1, rp_2, ...rp_n\}$ is used in this step as an input. This process aims at preparing the set of words using some NLP techniques, described in the Background Section.

First, a word segmentation of each raw phrase (rp) is processed, using white space, tabs and new line characters as delimiters. Second, a filter removes all the stop words. Third, in order to improve the accuracy of the word synonym lookup process, a Part of Speech (POS) is executed for each word, identifying nouns, verbs, adverbs and adjectives.

A phrase could be used to give an example about the usage of the POS algorithm. For the phrase $\{Financial, Reserve\}$, the POS recognizes the word "Reserve" as a noun instead of a verb. This approach has a positive impact on the disambiguation quality and performance, thanks to the reduction on the search space. The word "reserve" in this example has 7 senses for nouns and 4 senses for verbs in the WordNet 3.1 database. Without the POS algorithm, the verb senses would represent noisy information and, of course more information to process.

Finally, a stemming algorithm is invoked, transforming the original words into root words. As a result a new set of phrases called $P = \{p_1, p_2, p_n\}$ is generated, where each p is a set of instances $\{(w_1,t_1),(w_2,t_2),...,(w_j,t_j)\}$, w is the word, t is their respective type (noun, verb, adverb or adjective) and j is the quantity of words in the given phrase.

3.2.3 Synonyms Discovery

The goal of this step is to collect as many synonyms as possible for each word extracted from the service descriptor. Discovering a word synonym is a complex action, due to the multiple meanings of the same word in different contexts (aka polysemy). Therefore, two minimum components are necessary to work on this problem of word sense disambiguation: (i) a dictionary or a thesaurus; and (ii) a method of disambiguation.

For (i), we chose WordNet (Miller, 1995) (Stark and Riesenfeld, 1998): an electronic lexical database for the English language, developed at Princeton University. WordNet groups noun, verbs, adjectives and adverbs by means of conceptual semantic and lexical relations. Each grouping is called *synset*, and a wide range of tools is available to deal with these structures.

For (ii) we chose one of the first works related to the theme, proposed by (Lesk, 1986). Lesk uses an unsupervised method that disambiguates two words by finding the pair of senses with the greatest overlap in their dictionary definitions. This algorithm presents a low computing overhead because it explores only the sense of the words presented in the phrase, avoiding a deep navigation in the graph.

The algorithm receives the set of phrases P generated in the previous section as an input. Then, for each word processed, it looks for the set of senses in WordNet. To clarify the concept of sense, each sense has a dictionary definition that will be used by the disambiguation process. Giving an example, we try to disambiguate the word "risk" in the set of words $p = \{\{project, noun\}, \{risk, noun\}\}$. The word risk has 6 senses for noun and each one points to at least the original word plus one or more synonyms. Using the algorithm, the chosen sense of risk was "a venture undertaken without regard to possible loss or injury," instead of "the probability of being exposed to an infectious agent." This is an effect of the presence of the word "undertaken" in both definitions (risk and project). As a result two new words, related to the same sense were added: "peril" and "danger," expanding the vocabulary.

Lesk's approach is sensitive to the exact wording of definitions. In certain cases, words in the definition do not link in fact the words. The absence of a certain word or a presence of a frequent word can radically change the results. Several proposed methods achieve good results for disambiguation, but require a preprocessed dependency knowledge database as presented in (Chen et al., 2009) or make deep explorations in the glosses graph as presented in (Navigli and Velardi, 2005) and (Adala et al., 2011). They should be considered in future experiments.

In brief, at the end of this step, a set of keywords are generated for the service called $K = \{k_1, k_2, k_n\}$, where each k_n is a word (original or synonym) and n is the number of keywords for the underlying service.

3.2.4 Concepts Creation

A representation language is necessary to model the services, the dialogs and the input and output parameters to deal with the service. We use a representation language called MOSS[2] (Barthès, 2011b), summarized in the next paragraphs.

[2]available at http://www.utc.fr/∼barthes/MOSS/

MOSS. MOSS is a complex frame-based representation language, allowing the description of concepts, individuals, properties, classless objects, default values, virtual concepts or properties. It includes an object-oriented language, a query system, multilingual facilities and many other features described in the online documentation. MOSS is centered on the concept of property and adopts a descriptive (typicality) rather than prescriptive approach, meaning that defaults are privileged and individuals may have properties that are not recorded in the corresponding concept. Reasoning is done via a query mechanism. We only present here the features used for the creation of a general concept representing the service and its structure.

Concepts, Attributes and Relations. The service tree generated in Section 3.2.1, called *ST*, will be used as a source to create a representation of the service using the MOSS language. Given the tree, if a visited node is a leaf, then it will be transformed into a final attribute. On the other hand, if the node is intermediary, then it will be transformed into a new concept, and also a relationship with this new concept will be created in the current concept. This approach could result in possible redundancies when a complex type is used more than once. However it simplifies the transformation and also prevents the occurrence of circular references. To give an example of the creation of nested concepts, let's use the tree of Section 3.2.1, changing the simple attribute date to one more complex; for example a date type with a day, month and year inside its structure. Figure 6 shows the refactored tree.

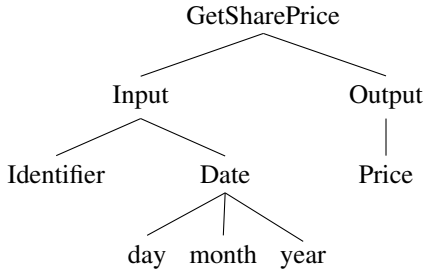

Figure 6: Modified tree.

In this example we have four nodes as candidates to concepts: The toplevel node *GetSharePrice*, the standard *Input* and *Output* nodes and the *Date* node. Figure 7 shows the generated concepts using the MOSS syntax. The *Date* node was transformed into a concept called *DateConcept*, once the node is intermediary. It also influenced the creation of the *InputConcept*, which had a relationship with the concept *DateConcept* instead of a simple attribute.

```
(defconcept "GetSharePriceConcept"
   (:rel "Input"
         (:to "InputConcept"))
   (:rel "Output"
         (:to "OutputConcept")))

(defconcept "InputConcept"
   (:att "Identifier")
   (:rel "Date"
        (:to "DateConcept")))

(defconcept "OutputConcept"
   (:att "Price"))

(defconcept "DateConcept"
   (:att "day")
   (:att "month")
   (:att "year"))
```

Figure 7: Generated concepts (:att specifies an attribute, :rel a relation).

After the creation of the concepts (*C*), they are incorporated into the OMAS environment, allowing the creation of individuals during any dialog session.

3.2.5 Dialog Creation

The goal of this step is to build an automatic sub-dialog based on the concept set (*C*), created during the previous step. To allow a conversation between the user and the PA, this goal-driven dialog is modeled as a finite state machine as shown in Figure 8. The goal of this type of dialog is the fulfillment of all of the attributes and relations of an individual representing the input.

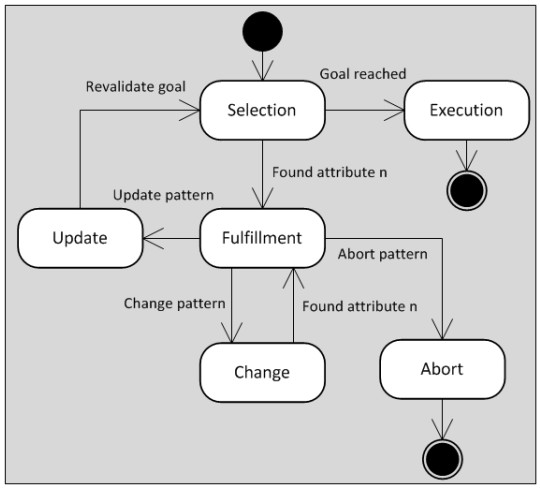

Figure 8: Service execution state machine.

***Selection* State.** The sub-dialog starts in the *Selection* state. This state aims at finding empty input attributes. Our technique proceeds along the following

assumptions:

1. One individual or instance related to the service context is created for each dialog;
2. For each relationship, at least one individual corresponding to the type of the relationship is created and associated to the underlying individual.

If all of the attributes of the input are filled, the next state will be *Execution*, responsible for the execution of the service, as well as displaying the execution outcomes represented by the output individual. The *Selection* state does not have interaction with the user and only uses the input individual for making the inference.

***Fulfillment* State.** If an empty attribute *n* is found, the control is transferred to this state, which aims at asking the master to fulfill the information. A question/answer dialog is started, asking the user to fill the attribute *n*.

***Update* State.** The control is transferred to this state if the algorithm detects an update pattern. Knowledge engineering processes using unstructured natural language texts are still an expert task and are sensitive to the context. We choose a discovery process that extracts the attribute value from the sentence based on simple grammatical constructions. We used a small subset of the proposal made by (Hahn and Schnattinger, 1998) called *Linguistic Quality Labels*. Linguistic quality labels reflect structural properties of phrasal patterns in which unknown lexical items occur. We assume here that the type of grammatical construction exercises a particular interpretative force on the unknown item. Based on this idea, we used the traditional triple noun-verb-noun phrase to detect sentences like "My age is 23" or "The window has a size of 3x5m." Apposition is also a source of information. The apposition almost unequivocally will determine the attribute in our case. An example of apposition phrase could be "The user Mary," where *user* is the attribute name and *Mary* is the correspondent value.

***Abort* State.** When the PA detects abort patterns in the master's phrase, it interrupts the process confirming their action. If the user confirms, then the service execution is aborted. If not, the control returns to the *Selection* state. An abort pattern could be interpreted as a set of words such as $\{cancel, quit, ...\}$. The sets were made empirically during this preliminary phase and must be evaluated in the future in order to improve their accuracy.

***Change* State.** This state enables the user to change the sequence of the fulfillment and for example, ask to change the value of any other attribute. Let's use the phrase "Sorry, I made a mistake when giving my age" during the *Fulfillment* state. The sentence does not fit in the requirements of the *Abort* state nor the *Update* state. It also presents linguistic cues indicating the desire to change something such as $\{mistake, sorry, ...\}$. If the algorithm matches one attribute in the phrase, then the control returns to the *Fulfillment* state. If there is no correspondence, the control returns to the *Selection* state.

***Execution* State.** The control is transferred to this state when the master has filled all the attributes. Here, the PA sends a message to the service agent (SA) responsible for the SOAP envelope creation, transportation and response decoding. The PA waits for the response and then presents the output to the end user, terminating the sub-dialog.

3.3 Discovery and Execution

The discovery and execution process is tailored to identify the user needs based on linguistic cues and conduct a dialog conversation. This process was inspired by the work on multi-agents task selection and execution using personal assistants dialogs described in (Barthès, 2011a).

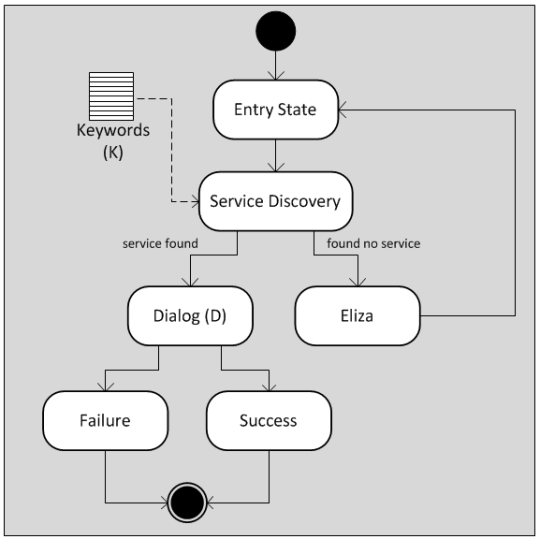

Figure 9: Discovery and execution activity diagram.

During this stage, we assume that the PA is currently associated with an end-user. That is to say, the user now has a channel opened with the personal assistant to make requests and also see the PA responses. It is important to point-out that the conversa-

tion between a PA and the end-user starts in a generic top-level dialog, described in the diagram shown in Figure 9. Any other dialog in the PA is considered a nested or sub-dialog, even our special sub-dialog used here. The starting point of the top-level dialog, namely *Entry State* allows users to input their requests. The next state is called *Service Discovery*.

Service Discovery. Using the service keywords (K), the service discovery is done as follows:

1. The user asks something to the PA (e.g., "Give me the project risks and their financial reserves")
2. The phrase is segmented into a set of words, removing spaces, words separators and empty words. For example, the resulting list of words would be $\{give, project, risk, financial, reserves\}$;
3. A stemming algorithm is executed for each word, transforming them into root words.
4. For each known service, the PA matches the user set of words with the set of keywords (K), using a MYCIN-like formula to compute a score. This is a simple but efficient approach, since each match of a subexpression reinforces the overall match score;
5. Services are then ordered by decreasing scores;
6. The service with the highest score is selected.

If a task cannot be determined, the PA enters in the state *Eliza*, in order to analyze the input and either produces an adequate answer or tells the user that it did not have enough information to process the input.

Dialog Execution. Once the top-level dialog has identified the target service to execute, a nested sub-dialog associated with the service is instantiated and started. The creation of this goal-oriented dialog was detailed in Section 3.2.5. After the execution of the sub-dialog the control is then returned to the main dialog.

4 A DIALOG EXAMPLE

In this section we present an utilization dialog example extracted from one ready-to-use OMAS environment, since the details to build one dialog were presented in previous sections. Figure 10 shows the beginning of the dialog, started with a welcome message from the PA, followed with some interactions.

When the PA sends the greeting message to the end-user, the conversation graph is currently in the

```
P: Good morning and welcome to the ACME Bank.
   I am Ariberto, your personal assistant.
   How can I help you?
U: I'd like to contract a health insurance.
P: Thank you for the interest in our
   Health Insurance Quote service.
P: Could you please tell me your name?
```

Figure 10: A conversation fragment that triggered the service identification.

Entry state. This state allows the interaction with the user and is changed as soon as the user types the request. The next state, *Service Discovery*, will process the user input, transforming the data into a set of root words, without delimiters and stop words. A matching process is then executed for each known service in the environment, using a MYCIN-like formula ($a+b-ab$). Given the set of words provided by the user, the selected service was the *Health Insurance Quote Service*. For demonstration purposes the environment had a few services, related to health, travel and vehicle insurance quotes.

As a result, a sub-dialog related to the health service was created. As a preparation process, one individual representing the service concept were created, as well as individuals to fill its relationships. To be more specific in this case, one individual was created for both input and output concepts, once the types related to this service are simple attributes. All service sub-dialogs start in the *Selection* state. It means that the sub-dialog behavior is to look for empty attributes in the input individual. In this example, the PA found the *name* and transferred the control to the *Fulfillment* state, which always waits for the user input.

As shown in Figure 11, the user sent the sentence "My name is John Doe" to the PA. The pattern found by the PA was compatible with an update, which in turn, extracted only the fragment "John Doe". This context information was then available in the *Update* state. As a result, the PA changed the value to "John Doe", returning the control back to the *Selection* state.

The user can change the order of fulfillment at any time. The same dialog of Figure 11 shows the PA asking for the user's age. In the dialog example, the user refused the fulfillment answering: "No, sorry, the name is not correct." The pattern found in the phrase matched with the *Change* state. This state found the attribute *name*. Hence, the control was transferred back to the *Fulfillment* state, which allowed the user to change his name and transfer the control to the *Update* state. The control was then returned back to the *Selection* state. Finally, the Web service was executed after the fulfillment process, as show in Figure 12.

```
U: My name is John Doe.
P: Thank you for the information.
   Confirming your name as John Doe.
P: Could you please tell me your age?
U: No, sorry, the name is not correct.
P: Your current name is John Doe. Could you
   please tell me your name again?
U: The name is John Malbec Doe
P: Thank you for the information.
   Confirming your name as
   John Malbec Doe.
   Could you please tell me your age?
U: I'm 23.
P: Your age is 23.
```

Figure 11: A conversation fragment that identifies a user intervention.

```
P: Now, I have all of the information to
   process your request.
   I would like your attention to check
   the following information before
   confirming your request.

   Name: John Malbec Doe
   Age: 23
   ...

P: Could you please confirm the request?
U: yes
P: Service requested successfully.
   Process number is 9992323231.
   Insurance price: $ 150.00.
```

Figure 12: A conversation fragment that shows the service execution.

5 RELATED WORK

Many research efforts have been made to propose approaches for discovery, composition and execution of Web services. In general, the proposals use semantic-based approaches on semantic descriptors and domain-specific ontologies. To give some examples, (Rivera et al., 2011) presents a framework for building applications targeted to end users who have no previous technology experience. One of the features is the service discovery, which is based on pre and post-conditions expressed in RDF graph patterns. It also presents a solution to service execution using a wrapper figure. This approach uses a dummy service execution in order to match the messages with domain specific data and wraps it into an application widget, which could be used as a common component for application modeling. (Lim and Lee, 2010) presents a sophisticated mechanism to discover and execute web services using semantic Web service information described in OWL-S. Different workflow templates are extracted, and then the algorithm selects the most suitable workflow by calculating similarities between sub-workflows. (Adala et al., 2011) uses a similar Web service discovery mechanism, using NLP techniques such as word disambiguation. However, the approach used to match the user natural language request and their knowledge representation is slightly different. It uses SUMO (Suggested Upper Merged Ontology) as a source to calculate the distance between the user request and the representation.

As could be seen, most of these solutions use semantic descriptors as an input. We have a great amount of services lacking semantic descriptors, normally found by third-party providers or even traditional Web services published in UDDI directories. It opens an opportunity for other techniques based on non-semantic information.

6 CONCLUSIONS

The work proposed in this paper provides an approach for discovery and execution of Web services. The approach presented here is both simple and efficient, particularly in environments with independent Web services without a proper domain-ontology and semantic annotations describing the service capability. It presents a discovery mechanism based on expanded keywords presented in a Web service descriptor and a fulfillment process using natural language. The sentences are provided by and end-user without previous knowledge about the physical location and structure of the service. The approach leverages the usage of several Web services constructed in a bottom-up fashion where only operational descriptors are available.

We are working on improvements of the approach in order to: (i) allow more sophisticated dialogs, especially when dealing with complex representation of input data and (ii) allow more complex scenarios using composite Web services.

The OMAS platform containing all the machinery for implementing multi-agents and dialogs, and the corresponding documentation can be downloaded from http://www.utc.fr/∼barthes/OMAS/

REFERENCES

Adala, A., Tabbane, N., and Tabbane, S. (2011). A framework for automatic Web service discovery based on semantics and NLP techniques. *Advances in Multimedia - Special issue on Web Services in Multimedia* , 2011.

Barthès, J.-P. A. (2011a). Flexible communication based on linguistic and ontological cues. In *E-Technologies: Transformation in a Connected World*, volume 78 of

Lecture Notes in Business Information Processing, pages 131–145. Springer.

Barthès, J.-P. A. (2011b). OMAS - a flexible multi-agent environment for CSCWD. *Future Generation Computer Systems*, 27(1):78–87.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. *Scientific American*.

Chai, J., Lin, J., Zadrozny, W., Ye, Y., Stys-budzikowska, M., and Horvath, V. (2001). The role of a natural language conversational interface in online sales: A case study. *International Journal of Speech Technology*, 4:285–295.

Chen, P., Ding, W., Bowes, C., and Brown, D. (2009). A fully unsupervised word sense disambiguation method using dependency knowledge. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, pages 28–36. Association for Computational Linguistics.

DARPA PAL Program, D. (2012). *Personalized Assistant that Learns*. SRI International. Available at https://pal.sri.com/Plone/framework.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine. Available at http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm.

Hahn, U. and Schnattinger, K. (1998). Towards text knowledge engineering. In *AAAI '98/IAAI '98 Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 524–531. American Association for Artificial Intelligence.

Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems documentation*, SIGDOC '86, pages 24–26. ACM.

Lim, J. and Lee, K.-H. (2010). Constructing composite Web services from natural language requests. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(1):1–13.

Miller, G. A. (1995). WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41.

Navigli, R. and Velardi, P. (2005). Structural semantic interconnections: a knowledge-based approach to word sense disambiguation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1075–1086.

Negroponte, N. (1996). *Being Digital*. Knopf Doubleday Publishing Group.

Rivera, I., Moller, K., Handschuh, S., and Zundorf, A. (2011). Web service wrapping, discovery and consumption - more power to the end-user. *7th International Conference on Web Information Systems and Technologies (WEBIST 2011)*.

SAWSDL Working Group, W. (2007). *Semantic Annotations for WSDL and XML Schema*. W3C Recommendation. Available at http://www.w3.org/TR/sawsdl/.

Simonite, T. (2012). Siri's new cousin works as a bank teller. MIT Technology Review - Business Report - The Future of Work. Available at http://www.technologyreview.com/news/428430/siris-new-cousin-works-as-a-bank-teller/.

Stark, M. M. and Riesenfeld, R. F. (1998). WordNet: An electronic lexical database. In *Proceedings of 11th Eurographics Workshop on Rendering*. MIT Press.

Tur, G., Stolcke, A., Voss, L., Peters, S., Hakkani-Tur, D., Dowding, J., Favre, B., Fernandez, R., Frampton, M., Frandsen, M., Frederickson, C., Graciarena, M., Kintzing, D., Leveque, K., Mason, S., Niekrasz, J., Purver, M., Riedhammer, K., Shriberg, E., Tien, J., Vergyri, D., and Yang, F. (2010). The CALO Meeting Assistant System. *Audio, Speech, and Language Processing, IEEE Transactions on*, 18(6):1601–1611.

Vitvar, T., Kopecky, J., Viskova, J., and Fensel, D. (2008). WSMO-Lite annotations for Web services. In *ESWC'08 Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, pages 674–689. Springer-Verlag.

Web-Ontology Working Group, W. (2004). *OWL-S: Semantic Markup for Web Services*. W3C Member Submission. Available at http://www.w3.org/Submission/OWL-S/.

Web Services Description Working Group, W. (2007). *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. W3C Recommendation. Available at http://www.w3.org/TR/wsdl20/.

Weizenbaum, J. (1966). ELIZA - a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.

WS-BPEL Technical Committee, O. (2007). *Web Services Business Process Execution Language Version 2.0*. OASIS Standard. Available at http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

WSMO Working Group, W. (2005). *Web Service Modeling Ontology (WSMO)*. W3C Member Submission. Available at http://www.w3.org/Submission/WSMO/.

XML Protocol Working Group, W. (2007). *SOAP Version 1.2 Part 0: Primer (Second Edition)*. W3C Recommendation. Available at http://www.w3.org/TR/soap12-part0/.