Enabling Intelligent Service Discovery with GGODO

MARIANO RICO, FRANCISCO GARCIA-SANCHEZ*, JUAN M. GOMEZ**, RAFAEL VALENCIA-GARCIA+ AND JESUALDO T. FERNADEZ-BREIS+

Computer Science Departament
Universidad Autónoma de Madrid
28049 Madrid, Spain

*Department of Informatics and Systems
University of Murcia
30100 Murcia, Spain

++Department of Computer Science
Universidad Carlos III
28911 Madrid, Spain

The Web has changed from a mere repository of information to a new platform for business transactions where organizations deploy, share and expose business processes via Web services. New promising application fields such as the Semantic Web and Semantic Web Services are leveraging the potential of deploying those services, but face the problem of discovering and invoking them in a simple way for common users. GGODO is an experimental solution that combines natural language analysis and semantically-empowered techniques to let users express their goals in a guided way, which produces better results than previous non-guided tools.

Keywords: semantic web services, human-computer interaction, software agents, semantic web, natural language processing

1. INTRODUCTION

The need to incorporate semantics to the Web becomes critical as more and more information is available. For example, Google have indexed billions of Web pages, but if it is asked for “books in which any one of Shakespeare’s books are mentioned” it cannot give us a correct answer. A simpler query such as “current number of web servers” would not return the desired results either. It is clear that keyword searching and indexing are not sufficient to solve previous queries. In May 2001, Tim Berners-Lee, James Hendler and Ora Lassila depicted their vision of the Semantic Web in their famous Scientific American article [1]. They envisioned a Web where software agents can deal with our wishes, exchange information with other agents, offer us the best options and execute our final decisions. However, it is necessary to provide semantics not only to Web pages, but also to Web services so that Web services discovering, composition and invocation can be done automatically. The problem so far has been that the simplicity of UDDI registries results in a lack of effectiveness when searching using a UDDI browser (such as the UDDI Browser from Sourceforge.net1).

The main public initiatives involved in adding semantics to Web Services with the

Received December 15, 2008; revised April 7, 2009; accepted August 20, 2009.
Communicated by Jonathan Lee.
* This work was supported in part by the Spanish Ministry for Industry, Tourism and Commerce through projects TSI-020100-2008-564, TSI-020100-2008-665, TSI-0204000-2009-148 and TSI-020100-2009-263, and Ministry for Science and Innovation under projects TIN2008-02081 and TIC-4425. We would like to acknowledge to Dr. David Camacho for his revision and suggestion in this work.
1 https://sourceforge.net/projects/uddibrowser/.
The purpose of creating the so-called Semantic Web Services (SWS) are OWL-S [2] and WSMO [3]. In the case of OWL-S there are no functional implementations although the first specification is already five years old. WSMX (Web Services Execution Environment) is a reference implementation of the WSMO ontology [4]. Nonetheless, WSMX still has not solved the issues related to the interaction with the human user. Currently, the inputs of systems like WSMX are based on the concept of ‘Goal’. However, describing a goal requires such complex formal statements that few users can do it (not even those who are more familiarized with computers). Users demand simplicity. They would like to express their wishes with the only language they know: natural language. Hence, recognizing user wishes is the first step in the chain. The ideal situation is when users can introduce their wishes by means of simple strings in natural language through a browser. In a second step, the goals should be extracted from the sentence in natural language, executed and achieved.

The system we have developed goes in this direction. Based on our previous prototype GODO [5], GGODO is a “Guided” GODO, adding to our previous work an assisted guidance that enables end-users to express their wishes easily. Inherited from GODO, GGODO is a software agent located between WSMX middleware and common users. It is able to transform the user requests expressed in natural language to the specific format required by WSMX goals. With this purpose it incorporates a language analyzer, which infers the concepts, attributes, attribute values, and relationships within a sentence thus producing a lightweight ontology. This ontology is then matched with a set of goal templates retrieved from a (local) repository. The goals produced as result of the matching process are sorted and sent to WSMX. In layman’s terms, the user has to type his/her wishes and GGODO will find the way of extracting the goals, composing them and make sure they reach WSMX, where they will match Web services capabilities and will execute certain Web services.

The experimental evaluation compares the effort to express goals. Two groups of users expressed different sets of goals to GODO and GGODO respectively. Results confirm that GGODO provides more successful goals, especially for short sentences and less experienced users.

The rest of the paper is structured as follows. Related work is surveyed and put into context in section 2. Section 3 presents the core of our work, whereby a complete description of GGODO and its main components is shown. A practical usage example is shown in section 4. The experimental evaluation is shown in section 5 and relevant conclusions are presented in section 6.

2. RELATED WORK

The main contributions of this work are related to Semantic Web Services and WSMX. Subsection 2.1 provides a brief description of the main characteristics of both, and subsection 2.2 describes the tools more related to GGODO.

2.1 Semantic Web Services and WSMX

The joint application of the Semantic Web and Web Services technologies with the
ENABLING INTELLIGENT SERVICE DISCOVERY WITH GGODO

Purpose of creating intelligent WS is referred as to Semantic Web Services. Semantic Web Services add a semantic description to Web services, so that discovery, composition and invocation processes for these services can be done in an automatic way. The W3C is currently examining various approaches with the purpose of reaching a standard for the Semantic Web Services technology. At this point in time, there are four submissions under consideration (OWL-S, WSMO, SWSF, and WSDL-S) and a complementary approach which is a W3C recommendation (SAWSDL).

The first approach to be submitted was OWL-S (OWL Web Ontology Language for Services). OWL-S is an ontology for services that makes possible for agents to discover, compose, invoke, and monitor services with a high degree of automation. It is composed of three main parts: the service profile for advertising and discovering services; the process model, which gives a detailed description of a service’s operation; and the grounding, which provides details on how to interoperate with a service, via messages [2].

WSMO Submission comprises three different elements: Web Service Modeling Ontology (WSMO), Web Service Modeling Language (WSML), and Web Service Execution Environment (WSMX). WSMO provides a conceptual framework for semantically describing all relevant aspects of Web Services in order to facilitate the automation of discovering, combining and invoking electronic services over the Web [3]. It is based on the Web Service Modeling Framework (WSMF) [6], consists of four main elements: ontologies, which provide the terminology used by other elements; goals that define the problems that should be solved by Web services; Web services descriptions that define various aspects for a Web service (functional and behavioral); and mediators, which bypass interoperability problems. WSML provides a formal syntax and semantics for WSMO and is based on different logical formalism (Description Logics, First-Order Logic and Logic Programming). Finally, WSMX [4] defines a Semantic Web Services architecture and provides an initial implementation based on the conceptual model of WSMO. It enables discovery, selection, mediation, invocation and interoperation of Semantic Web Services.

WSMX is the reference implementation for WSMO. The WSMX goal is twofold: to provide a test bed for WSMO and to demonstrate the viability of using WSMO as a means to achieve dynamic inter-operation of Web services. It provides dynamic discovery, mediation, selection and invocation of semantically described Web services. For this, it receives as input a user goal specified in WSML [7], the underlying formal language of WSMO. Basically, the process can be briefly summarized as follows: the user goal is matched with the description of a Web service and then the Web service is invoked and the results are returned.

2.2 Tools Related to GGODO

Several investigations are being carried out with the purpose of integrating both Multiagent Systems and (Semantic) Web services. By extending (Semantic) Web services with Intelligent Agents features, new more autonomous and flexible applications can be developed. For example, in [8] an application where intelligent agents, aided by context information provided by Semantic Web Services, assist their users with different sets of tasks is described. Another practical approach in this direction is Semantic Web FRED [9]. It combines agent technology, ontologies, and Semantic Web Services in or-
order to develop a system for automated cooperation. In this system, software agents (‘Freds’) perform tasks on behalf of their owner and interact among them if they have to. In order to resolve a task, agents make use of Services, computational resources that allow for automated resolution of tasks. The authors define three types of Services: plans (Java programs), processes (complex and nested services), and external Web services (through WSDL). A further possibility for Web services and Intelligent Agents interoperation is presented in [10]. In this work, the authors highlight the passive behaviour of Web services and propose to wrap them into proactive agents. The idea is to use an intelligent agent acting on behalf of a Web service within a workflow. When this Web service is intended to be executed, its representative agent gets the flow control and attempts to improve the workflow process by using its ‘intelligent’ capabilities. Therefore, static workflows defined through Web services become highly dynamic flows of control thanks to the proactive and autonomous characteristics of agents. They envision workflow-based multiagent systems where service providers are agents themselves, thus acquiring the full proactive, autonomous, and selfish characteristics that are associated with agency.

As we will describe later, for the interaction with the user we use an ontology-guided input approach based on a system called Ontopath [11]. Ontopath assist editing in such a way that it recognizes the resource type of a description and offers users context-sensitive actions to perform on that description. For it, Ontopath makes use of a domain ontology which represents domain specific nouns and verbs by means of RDF triples. Then, an editor helps users with guidance on choosing next words, using the approved grammars and semantic relations of entities from the ontology. Later, some improvements were introduced into this tool, and the CLN Editor was presented [12]. The CLN Editor allows enlarging the expressivity by expanding the grammar, specifying patternized sentences, and adapting informal expressions. In order to achieve this increase in expressivity, the CLN Editor defines the grammar through CFG-LD (Context-Free Grammar with Lexical Dependency). A further approach for the generation of natural language sentences with the guidance of an ontology is proposed in [13]. In this work, the authors describe GINO (Guided Input Natural Language Ontology Editor for the Semantic Web), an editor that allows users to edit and query ontologies in a language akin to English. Although the users guidance is similar, the main differences are: (1) application scopes are different (edition and querying in GINO, and goals analysis in GGODO), and (2) platform (GINO is a desktop application, GGODO is a web application).

3. GGODO: AN AGENT FOR SEMANTIC WEB SERVICES

As previously mentioned, WSMX enables automatic discovery, selection, mediation, invocation and interoperation of Semantic Web Services. It receives goals in a specified format as input and returns the Web services invocation results. In [14], Wooldridge defines the term “agent” as a “computer system situated in some environment and capable of autonomous action in this environment in order to meet its design objectives”. Thus, GGODO can be seen as a software agent that acts on behalf of a user with the objective of facilitating the interaction with the WSMX execution environment. It accepts a text in natural language expressing the user wishes. Once GGODO has inferred what the user wants, it creates the goals WSMX have to execute in order to achieve it. In other words,
the GGODO approach is concerned with enabling orchestration of Web services by means of natural language communication. Our goal is to extract the intention from the natural language based generation and synthesis. For this we have conceptualized the domain of the communication and designed a software architecture to extract the intention of the user.

GGODO extends GODO in several ways. It includes an alternative method for the input of user requirements, which further facilitates the interaction of users with the system. Besides, it incorporates a sophisticated mechanism to carry out the matching between the requirements inserted by users and the goals available in the repository. The aim of this section is to describe the architecture of this evolution and the functionality of its constituting components. Loose coupling and reusability of components have been the major intentions behind the architectural decisions and implementation.

3.1 User Input Module

GGODO disposes of two alternative methods to obtain the users wishes. One is based on a natural language processing tool being able to transform an input in the form of a natural language sentence into a lightweight ontology containing the concepts that are part of the sentence and their relationships. Since current natural language recognition techniques are far from being 100% accurate, an alternative method was incorporated into the first stage of the process. This second approach consists in helping the us-

Fig. 1 depicts the main components of GGODO. The main component is the Control Module, which is supposed to supervise all the process and act as intermediary among the other components. The GUI is placed between the user and the Control Module. It receives the user text and passes it directly to the Control Module that asks the User Input Manager the meaning of the sentence. After that, the Goal Loader gets all the possible goals from a repository and the Goal Matcher infers what goals should be sent to WSMX taking into account the results of the Language analyzer. Finally, the Control Module sequentially sends these goals to the Goal Sender, which takes over the mission of dispatching them in WSML (Web Services Modeling Language) format to WSMX.
ers to introduce their wishes with the assistance of a domain ontology. Users are offered with a set of possible words to insert next in the sentences they are writing. Next, these two mechanisms are described in detail.

3.1.1 Language analyzer

The task of the Language analyzer is to filter and process the input introduced by the user in natural language and determine the concepts (attributes and values) and relations included in it.

The Language analyzer used in this work is based on the methodology presented in [15] to get knowledge from text. This methodology, which uses ontologies and one incremental knowledge acquisition technique termed MCRDR [16], is based on the idea that relationships between concepts are usually associated to verbs in natural language. This methodology uses the mentioned technique and the grammar category of the words in the current sentence to infer other knowledge entities (e.g., concept, attributes and values) in order to create an ontology from a text fragment.

It is necessary a previous phase of training in which the experts establish all the patterns that will use the analyzer to get the concepts and relations within the sentence introduced by the user. This training process is based on the study of a set of texts in natural language related to a specific domain. In this phase, the experts indicate the concepts, concepts’ attributes, attributes’ values and relations among concepts, and the system stores all this information in the database. After this, when a real user introduces a sentence the system must be able to get the concepts, attributes, values and relations within this sentence taking into account all what it has learned in the training process.

This knowledge acquisition process is divided into three sequential phases, namely the “POS-Tagging”, “Concept search” and “inference” phases.

The main objective of the POS-Tagging phase is to obtain the grammatical category of each word in the current sentence; for this purpose, the POS-tagger described in [17] is used. In the Concept search phase, linguistic expressions representing concepts are identified. The associations between linguistic expressions and concepts have been stored in a conceptual knowledge base obtained in the (previous) training phase of the system. As a result of this phase we obtain all the expressions of the fragment, which are already in the conceptual knowledge base. The last phase, inference, is based on the idea that, in natural language, relationships between concepts are usually associated to verbs. The MCRDR component used to obtain the relationships between concepts is formed by a knowledge base containing linguistics expressions representing generic conceptual relationships, and by a subsystem that infers the participants in these relationships.

This process comprises the next steps: (1) the verb in the current sentence is identified, (2) the system searches for the type of semantic relationship associated to that verb. Once the type of relation associated to the main verb in the current sentence has been found, the MCRDR sub-system is applied to extract knowledge by means of the grammatical category of the words, their position in the current sentence, and the type of relation associated to the verb, if any. We will include the following example for further clarification.

Suppose that the Language analyzer component receives a sentence in natural language as a parameter and returns the concepts (attributes and values) within this sentence.
and the relations among these concepts (a lightweight ontology). As a result, the ontology resulting from the sentence “I want to buy a plane ticket from Galway to Madrid, then book a hotel in the centre and rent a C class car” would be like the one shown in Fig. 2.

![Fig. 2. Example of resulting lightweight ontology.](image)

In this figure we can distinguish four concepts (“Subject”, “Planet ticket”, “Hotel” and “Car”) and three relations (“buy”, “book” and “rent”). The concept “Subject” is a more general concept than “I” and is directly inferred by the system. The other three concepts have attributes such as “from”, “to” and values for these attributes (“Galway”, “Madrid”).

This lightweight ontology stemming from unstructured natural language will be used to be matched with a goal template or a goal description.

### 3.1.2 Ontology-guided input

Since the accuracy of the language analyzer cannot cover all the richness of the natural language, we provide users with an alternative mechanism to indicate their wishes to the system. The aim of the Ontology-guided Input component is to assist users in composing their sentences. For this purpose, the Ontopath System and other approaches for natural language interfaces (NLIs) (e.g. [11, 12, 13, 18, 19]) has been used as source of inspiration. In this context, NLIs help users avoid the burden of learning any logic-based language offering end-users a familiar and intuitive way of query formulation [20].

For the development of this component we have also taken into account the problems of linguistic variability and ambiguity of NLI systems. In order to overcome In recent years, Controlled Natural Language (CNL) [21, 22] has received much attention due to its ability to reduce ambiguity of natural language. CNLs are mainly characterized by two essential properties [23]: (1) the grammar of the controlled language is more restrictive than that of the general language, and (2) the vocabulary of the controlled language contains only a fraction of the words that are permissible in the general language. These restrictions in the language aim at reducing or even eliminating both ambiguity and complexity.

The Ontology-Guided Input component is a CNL-based NLI system called OWL-Path that assists users in indicating their queries to the system. The global architecture of
this component is depicted in Fig. 3. OWLPPath is composed of five main components: the “Ajax interface”, the “Suggester”, the “Grammar checker”, the “SPARQL generator”, and the “Ontology manager” (which manages the access to an external ontology repository). In a nutshell, the system works as follows. When a user selects the Ontology-Guided Input, a set of ontologies are loaded as the underlying knowledge base. Thereafter, the user interacts with the system through the “Ajax interface”. In order to input a query, users must select the desired terms they want to put next in the sentence from the list of terms provided by the interface in a pop-up. Entries that are not in the pop-up list are ungrammatical and not accepted by the system. This list of terms are generated by the “Suggester” module, which makes use of the “Grammar checker” to determine the terms that can come next in the query by combining the knowledge in both a question ontology and a domain ontology, and taking into account the previously inputted terms in the sentence. The question ontology comprises the grammar that all formulated queries must comply with and in the domain ontology is represented the relevant knowledge about the application domain. Once the query is completed, the system includes the “SPARQL generator” to optionally translate the entry to SPARQL statements to be able to issue it to an RDF-based knowledge base. Otherwise, the user query remains internally represented as an ontology. Next, we provide a more detailed description of the components that comprise OWLPPath.

**Ajax interface** This component constitutes the input text interface with the user. Through this interface, the system shows users the most appropriate terms that can follow in the elaboration of a sentence. The entries list is generated by the “Suggester” in accordance with a number of variables: the grammar defined in the question ontology, the imported ontologies, and the words that precede the new term in the query. The possible completions of the user’s entry are offered by presenting the user with choice pop-up boxes. These pop-up menus offer suggestions on what the next term might be. Users can navigate the pop-up with the arrow keys or with the mouse and choose a highlighted proposed option with the space key. As stated before, entries that are not in the pop-up
Suggester  The main objective of this component is to determine what words can be inserted next in a sentence. Each user input generates navigation in the ontology, so all the possible directions for such navigation are obtained by this component. From the linguistic perspective this means that it will provide the list of linguistic expressions that might be used given a particular context. This context can be summarized by the current node in the navigated ontology. Then, the “Suggester” will process the user input by considering the semantic relations defined in the domain ontologies and the grammar implicitly contained in the question ontology. Hence, it generates a tree containing all the possible grammatical options given the current input and the list of candidate expressions are sent to the “Ajax interface” component to be shown to the user.

Grammar checker  Most guided input systems (e.g. GINO [13] and Ontopath [11]) are based on BNF grammars to define the sentences that are grammatically correct. These grammars are mainly based on rules containing references to classes or instances in a domain ontology. In our work, the grammar of the system is represented by means of an OWL ontology, namely the question ontology, that imports ontological elements from the domain ontologies that the system has to take into account for building the query. Given that domain ontologies can contain a lot of different ontological elements (classes, datatype properties, object properties, instances, etc.), it is desirable that we can restrict the elements that will be part of the users queries. The question ontology is the core of this module. It represents the system grammar and models the queries that are grammatically correct by importing relevant elements from the domain ontologies that constitute the knowledge base. This ontology is implemented in the DL-variant of the OWL ontology language.

SPARQL generator  Once the user has decided that the sentence containing the query is completed, OWLPath has to translate it into SPARQL statements. The “SPARQL generator” is the component responsible for transforming natural language queries into SPARQL. The input parameter of this component is the final sentence submitted by the user. In fact, this sentence is not only composed of the words that comprise the phrase, but also contains the underlying triples the “Suggester” used to provide the optional entries.

Ontology manager  This component allows accessing the ontology repository by using the Jena Semantic Web Framework [25]. Strictly speaking, it provides an interface that can be implemented using Jena, the OWLAPI [26], web services and so on. It will load the ontologies needed for the correct operation of the system, including the domain ontology and the application one. It would also allow to handle distributed, heterogeneous ontology repositories, providing an OWL interface for them. This component will be responsible for providing all the ontological content needed to the “Grammar checker”
and to the “SPARQL generator”. The “Suggester” sends the options to the “Grammar checker”, and then the “Ontology manager” provides the information for checking the correctness of the question ontology. Furthermore, when the user completes the query, it can be translated into SPARQL and issued through the “Ontology manager” to an ontology repository.

An example of an ontology-guided input is shown in Fig. 4. As a result of both the Language analyzer and the Ontology-Guided Input components, the system holds an ontology that represents the user intention. This ontology will be used in later stages of the execution process.

3.2 Goal Managing Module

This module comprises three basic submodules: the Goal Loader, the Goal Matcher and the Goal Sender. These components are responsible for processing the user input, now represented by means of an ontology, and sending the associated goals to the corresponding semantic web services execution environment.

3.2.1 Goal loader

This component looks for the goals in the WSMO goal repository or in an internal file where the goals are stored. Currently, GGODO loads the goals from the file system. The local folder containing the goal files is set in the configuration file and loaded during
the application startup. However, given that this component is outside the architecture, anybody may plug in his/her own goal repository.

The syntax of these goals is expressed in the Web Services Modeling Language (WSML) [7] used by WSMO as the underlying logical formalism. From our point of view, these goals are just syntactically important given that, at this stage of the implementation, we will be using the inline concepts for our matching.

### 3.2.2 Goal matcher

Matching is a widely-used term, which, in our case, encompasses a syntactic and semantic perspective. The Goal Matcher compares the ontology elements obtained from the analysis of the user’s wishes to some parts of the description of the WSMO goal templates extracted from the repository. Depending on the user input, one or several goals are selected and sent to the Control Module, which composes them in order to build up the sequence of execution.

![WSMO Goal definition](image)

![WSMO Capability definition](image)

In order to better understand the matching process taking place in this component, a detailed description of the contents of WSML goals as defined by the WSMO ontology becomes necessary. In Fig. 5, the definition of Goal given by WSMO is shown. From our matching process viewpoint, the requested capability is the most important element in the definition of the goal. It describes the capability of the Web Services the user would like to have. This capability element is also present in the WSMO Web Service definition and is composed of four main components (see Fig. 6), namely, preconditions, assumptions, postconditions and effects. Postconditions and effects are particularly important for the definition of goals. Postconditions describe the state of the information space that is guaranteed to be reached after the successful execution of a Web Service, and effects refer to the state of the world that is guaranteed to be reached after the successful execution of a Web Service. The matching in GGODO focuses on these two elements of the
goals definition. The concepts and relationships in the lightweight ontology representing the user’s wishes are compared with the contents of the goals postconditions and effects. For simplicity reasons, in its current state of development, GGODO assumes that the same ontology has been used for both representing the user’s wishes and defining the goals. Thus, a mere syntactic comparison between the labels in both elements is enough to find the goals that satisfactorily represent the user requirements.

For example, taking a closer look at the resulting ontology in Fig. 2 it is possible to recognize three major relationships, namely, buy, book and rent. The Goal Matcher matches each of these relationships with a different goal, so three subsequent goals are involved in the resolution of this user request. Once the Goal Matcher has got all the necessary goal templates, it is responsible for fulfilling them. For this purpose, the remainder elements of the ontology come into scene. In particular, attributes and attribute values are applied to set the concrete values of the free variables in the goal template. The task of the Goal Matcher is to determine the elements in the ontology (Fig. 2) that correspond to each gap. For this purpose, it uses the attribute names (e.g. from, to), which in this example exactly matches the expected attributes. Then, the values of this attributes in the ontology are included in the goal template leading to the complete specification of a goal to be sent to the WSMX execution environment.

![Ggodo](image)

Fig. 7. GGODO identified goals proposal.

Fig. 7 shows the goals identified by GGODO. If the user agrees with the proposal, these goals are sent to WSMX; otherwise, the user returns to the main page and can refine the original sentences.

3.2.3 Goal sender

This component sends the different goals to WSMX. Its functionality is quite simple since the orchestration is predefined in the GGODO Control Module. The goals are sent sequentially, without taking into account any other workflow constructs.
WSMX provides several ways for third-party applications to access the functionality exposed by the system [27]. The main operations available through these entry points are ‘achieveGoal’, ‘discoverWebServices’, ‘invokeWebService’ and ‘store’. These operations have been made available through HTTP and can be also accessed via Web Services. The WSDL of the Web Service exposing the WSMX entry points and additional operations is available at http://host:8001/jaxws/WSMXEntryPoints?wsdl (JAX-WS) or http://host:8050/axis/services/WSMXEntryPoints?wsdl (Axis).

From the GGODO viewpoint, we are particularly interested in ‘achieveGoal’, which receives the goal to achieve as input and (supposedly) returns the result instances. Thus, the Goal Sender functions as an Axis2 (http://ws.apache.org/axis2) Web Service client invoking the ‘achieveGoal’ method of the WSMX Web Service with the goal(s) received from the Control Module. The URL of the WSMX Axis Web Service is set in the configuration file of the Web application but can be updated by the user at any time during the system execution.

3.3 Control Module

This component manages the different interactions among the components. Firstly, it is informed by the GUI that the wish of the user has been described in plain text. Then, it instructs the Language analyzer to attempt the recognition of the major concepts in the text and communicates with the Goal Loader and the Goal Matcher to orchestrate the different goals that will be sent to WSMX through the Goal Sender. Then, it communicates with the GUI so that the user receives a view of the selected goals and decides if they are correct and comply with his/her expectations. Finally, if the user approves them, they are sequentially sent.

4. PRACTICAL USAGE EXAMPLE WITH GGODO

In this section, we illustrate an usage example of our approach. In a simple B2C scenario, a customer wants to arrange a trip. In an ideal case, the users would specify their wishes (goals) in natural language by typing in the web-based GGODO GUI sentences such as: “I want to buy a plane ticket from Galway to Madrid, then book a hotel in the centre and then rent a C class car”. The main web page of GGODO is shown in Fig. 4. As shown in the top part of this figure, a radio button activates the guided input, and a text area is provided to the users to express their goals. After clicking in the text area, the users are shown a list with the first possible valid values to use. The lower part of the referred figure shows a list of predefined goals that can be used as illustrative examples. When one of these examples is selected, the goal text is placed into the text area, and it can be modified by the user.

Once the user has finished entering the goal, the system processes it. Current technologies such as web search engines would index this natural language expression trying to find pure syntactical relationships with the WSML goals available in the system. In contrast, the GGODO approach takes advantage of Semantic Web technologies following a three-fold strategy:
(1) *It represents the users’ wishes in the form of an ontology.* As it was described above, the sentence introduced by the user, either in the form of plain natural language or using the guided approach, is translated into a lightweight ontology containing the main concepts and relationships that are present at the user input. In both cases, GGODO uses Jena to deal with ontology development and management and the ontology will be represented using OWL (DL variant). It is also worth pointing out that the ontologies that contain the users’ wishes are fully aligned with the system domain ontology (represented at the bottom of Fig. 1).

(2) *It accesses a set of goal templates defined in WSML.* GGODO has access to a goal repository that contains a number of goal templates complying with the WSMO specification. As it was stated before, these goals are expressed in WSML, which is a formalism slightly different to OWL to represent ontologies. However, both WSML and OWL are based on Description Logics, which made feasible the conception of a set of mapping rules to translate WSML into OWL and vice versa, and the development of translator tools to support such conversion process. GGODO requires the application of one of such tools to obtain the OWL representation of the WSML goals in the repository in order to successfully carry out the last step of this three-fold strategy.

(3) *It follows a semantic approach to goal discovery.* Given that we have both the users’ wishes and the goals represented in the form of ontologies, the matching process to find the goals that fit with the wishes can take advantage of the underlying semantics. For this first GGODO prototype, we take a simplistic approach, we assume that the same ontology is used to represent the wishes and the goals. This assumption eases the matching process since it reduces to a straightforward syntactic problem. GGODO looks for the goals whose postconditions and effects match with the concepts and relationships in the ontology representing the user’s wish.

This three-steps strategy covers the phases of user input, goal loading and goal matching. Once these goals have been found and ordered, the Control Module chains the goals and orchestrates them properly, sending them to the Goal Sender. Before sending these goals to WSMX, the goals are verified by the user, confirming that those are the right goals and that they are in the correct execution order. Fig. 7 shows the goals identified in the example and the execution order. The user accepts the proposed goals by clicking the “Execute” button; and can modify them by clicking the “Cancel” button.

A test bed can be found at [http://nadir.uc3m.es/godo](http://nadir.uc3m.es/godo). This environment, which is focused in the tourism domain, was used to evaluate GGODO as section 5 shows.

### 5. EVALUATION

The evaluation of GGODO was focused on comparing the quality of the users’ goals to the previous non guided GODO version.

#### 5.1 Experimental Setup

Ten post graduate students were selected from one of our institutions and divided in two groups. The first group, named A, was trained in GODO, in a 10-min talk. The sec-

---

2. [http://tools.sti-innsbruck.at/wsml/wsml2owl-translator/v0.1/](http://tools.sti-innsbruck.at/wsml/wsml2owl-translator/v0.1/).
ond group, named B, was trained in the GGODO usage for 10 minutes. The users of both groups were instructed to create goals by using the assigned tool. These goals were divided in three levels: level 1 considered only one sentence, level 2 considered goals expressed in two sentences, and level 3 considered goals expressed by using three sentences. For each level the user had to create 5 different goals. Therefore, each user had to create 15 goals related to the “travelling” topic with no time limit for this task. A sentence is considered as any string finished by “dot”, “then” or “and”. For example, the example shown in Fig. 4 comprises 6 goals, and the example in Fig. 7 comprises 3 goals. Users had to build up simple goals like the last one.

The success of a user goal was computed for each level as the relative number of times that the user achieved the level. For example, for level 2, users can get 0, 1, or 2 goals. If a user obtains 2 goals, it is computed as 100% success. In level 3, with three sentences, a user obtaining 2 goals gets a 66.66% success.

The goals were numbered by means of an ID in the same order they were created, i.e. the first ID belongs to the first goal created by the user in the level 1. The last ID (5) belongs to the fifth goal in the level 3.

The system was loaded with a WSMO repository containing 40 goals. The ontology used by OWLPath had 30 classes, 33 properties (25 data properties and 8 object properties), and 50 individuals.

5.2 Experimental Results

Fig. 8 shows the success (in the vertical axis) for each user goal ID (in the horizontal axis) for the three levels. Level 1 is shown in Fig. 8 (a), level 2 in Fig. 8 (b), and level 3 in Fig. 8 (c). Each bar is the five users average success value, with dark gray bars for group A (GODO users) and light-gray bars for group B (GGODO users).

In each sub-figure, GGODO users have higher success values than GODO users. The straight lines in each sub-figure are computed as the linear approximation for each group and level. GGODO users have this line over the GODO users for the three levels, showing a lower distance for higher IDs. This can be explained viewing sub-figures in order (a), (b), (c). This order corresponds to the chronological evolution of the experiment.

![Fig. 8. Average success for three different goal sizes. Goals expressed in one, two, or three sentences are shown respectively in (a), (b) and (c).](image-url)
Fig. 8. (Cont’d) Average success for three different goal sizes. Goals expressed in one, two, or three sentences are shown respectively in (a), (b) and (c).

For all goal sizes, GGDODO produces better average results than GODO. This is especially remarkable for one-sentence goals, in which the first time an average user creates a goal, he/she obtains a successful result for the 80% of the cases while GODO gets only a 20%.

Fig. 8 (a) shows that the lower IDs, the higher differences; that is, GGODO users are more precise expressing goals than GODO users creating their first goals, and this difference decrease when more experience is acquired by users (higher IDs).

Therefore, we conclude that GGODO provide users with a better mechanism to create success goals compared to GODO. This result is better for small goals (comprising few sentences) compared to creating goals with bigger sentences. This result is due to the increasing statistical difficulty to achieve a successful result when many goals must be satisfied. However, although GODO has lower success, the success increases when more (short) sentences are added.

6. CONCLUSIONS AND FUTURE WORK

The Semantic Web Services model holds the promise of increasing IT productivity and impact. Despite the effort to provide reliable technology, such as the Web Services Modeling Execution (WSMX) platform, the topic of the interaction between these systems and common users expressing their goals in natural language has not been covered.

However, WSMX does not provide a reliable and intelligent human-oriented entry point, precisely where the GGODO platform steps in. Similar to utility services provided by power or telephone companies, GGODO can be regarded as an intelligent knowledge-based application for the user to easily use the full potential of the Semantic Web Services technology and focus on those activities that provide business impact and added value.

GGODO provides users with a guided-input, enhancing the usability features of the previous non-guided tool named GODO. Experiments show that user guidance provides users with a better way to express their goals, which produces better results and improve the user confidence on these tools.

Our future work will focus on determining the feasibility of a semantic match of light-
weight ontologies extracted from natural language text and ontologies defining goals in particular domains. This work is related to existing efforts about ontology merging and alignment. Besides, we plan to introduce goal templates so that these templates can be completed by using the information extracted from the natural language processing stage. It is also intended to incorporate a new component to further assist users to express their wishes. In particular, an ontology-based component has been envisioned, which offers users a set of possible words for composing a new statement. This set of recommendations would be based on the structure of an ontology that represents knowledge about feasible actions. Furthermore, other approaches for natural language processing will be analyzed for their use within the GODO architecture. Especially, an option that will be taken into account is GATE (General Architecture for Text Engineering), an infrastructure for developing and deploying software components that process human language [28]. In the future, the intent is to decouple GGODO from the specific middleware for Semantic Web Services used, thus allowing the system to interact with other middleware such as OWL-S Virtual Machine [29] or METEOR-S [30]. Also, ways to overcome the limitations of having one single step execution per goal will be explored.

REFERENCES


Mariano Rico received the B.S. degree in Physics Sciences from Universidad Autónoma de Madrid (UAM) in 1992. He joined the Department of Computer Science at UAM in 2003 and received his M.S. and Ph.D. degrees in Computer Science and Telecommunications in 2004 and 2009 respectively. Previously he was working for private R&D centers in Spain in Telecommunication and Computer Science areas. His research interests include semantic web technologies, human-computer Interaction, web architectures, and web services. He had two stays in the world leading Digital Enterprise Research Institute (DERI), in Galway, Ireland, where he worked on Interaction with Semantic Web Services and Wiki Technologies.

Francisco García-Sánchez received his B.A., M.Sc. and Ph.D. degrees in Computer Science from the University of Murcia. He is a lecturer at the Department of Informatics and Systems, University of Murcia. His research interests include agent technology, service oriented architectures and the semantic web. He has conducted a number of research stays in world leading re-search institutes in Ireland, Austria, USA and Australia, and has published over 30 articles in journals, international and national conferences and workshops.

Juan Miguel Gómez received his Ph.D. in Computer Science from the National University of Ireland, Galway while at the Digital Enterprise Research Institute (DERI), his Master Thesis in Software Engineering at the Swiss Federal Institute of Technology (EPFL) in Lausanne (Switzerland) and a M.Sc. in Telecommunications Engineering from the Universidad Politécnica de Madrid (UPM). He is an Associate Professor at the Computer Science Department at the Universidad Carlos III de Madrid. His research interests include the semantic web, semantic web services, business process modeling, formal methods for eCommerce and eBusiness. Dr. Gomez has been involved in EU FP5 and FP6 projects, published more than thirty international contributions and has been the organizer of related major scientific events in the area.
Rafael Valencia-Garcia received his B.A., M.Sc. and Ph.D. degrees in Computer Science from the University of Murcia. He is a lecturer at the Department of Informatics and Systems, University of Murcia. His main research interests are Natural Language Processing and the application of Knowledge Technologies. He has published over 20 articles in journals, conferences and book chapters. He is the author or co-author of several books.

Jesualdo Tomás Fernández-Breis received his B.A., M.Sc. and Ph.D. degrees in Computer Science from the University of Murcia. He is a Lecturer at the Department of Informatics and Systems, University of Murcia. His research interests include the development and application of knowledge technologies to different fields such as medicine, the semantic web, e-learning and Bioinformatics. He is currently the prime investigator of three national projects and has published over 40 articles in journals, conferences and book chapters. He is the author or co-author of several books.