A Multi-Agent System for the Composition of Semantic Web Services Based on Complexity Functions and Learning Algorithms

Andrei-Horia MOGOS, Adina Magda FLOREA
Faculty of Automatic Control and Computers
University POLITEHNICA of Bucharest, Romania
andrei.mogos@cs.pub.ro, adina.florea@cs.pub.ro

Semantic web services represent an important and actual research area in computer science. A very popular topic in this area is the composition of semantic web services, which can be used for obtaining new semantic web services from existing ones. Based on a representation method for the semantic descriptions of semantic web services, that we had previously proposed, we propose a multi-agent system for the composition of semantic web services based on complexity functions and learning algorithms. Our system starts as a semi-automatic composition system, but after it gathers (using learning algorithms) sufficient information about the knowledge domain in which it is used, the system is able to perform compositions of semantic web services automatically. Based on the previously proposed representation method, this paper describes the structure and the main algorithms of the proposed system. The paper also presents an example of using the proposed system and some experimental results.
Keywords: *Semantic Web Service, Composition Of Semantic Web Services, Multi-Agent System, Complexity Functions, Learning Algorithms*

1 Introduction

These days, semantic web services represent an important and actual research area in computer science. One of the main topics in this area is the composition of semantic web services, the main goal being to obtain new semantic web services by composing existing ones.

In [1] the we propose the following classes of semantic web services composition methods: semi-automatic composition (see [2]), AI planning (see [3][4][5]), agents and multi-agent systems (see [6][7]), logic languages and rules (see [8][9]), and bio-inspired methods (see [10][11]). One of the main conclusions of our analysis presented in [1] is that most of the semantic web services composition methods are automatic methods.

In [12] we propose a method for representing the semantic description of a semantic web service using complexity functions; for information related to complexity functions, see, for example [13][14]. This new representation method is presented in [12] in a formal way, by proposing several definitions and theorems.

In this paper we propose a multi-agent system for the composition of semantic web services based on the semantic descriptions representation method proposed in [12]. At the beginning, our method is semi-automatic, since it gathers information about the knowledge domain in which it is used. Later, after several uses of the method for the same knowledge domain, the method becomes automatic and it doesn't ask new information from the user. The main idea is that our method starts by being semi-automatic, learns the knowledge domain in which it works by using several learning algorithms and then it becomes automatic and doesn't need new information from the user for solving the semantic web services composition problem. Consequently, given a knowledge domain to work with, and a period of training, our method can be considered an automatic composition method.

Taking into account the analysis made in [1], we can say that our automatic method belongs to the class 'agents and multi-agent systems'. An important element of originality of our composition method is that it uses a new representation method for the semantic descriptions, the one proposed in [12].

The paper is organized as follows. Section 2 presents some concepts proposed in [12] that

DOI: 10.12948/issn14531305/18.2.2014.07

are necessary for this paper. Section 3 proposes a way to enrich the semantic description of a semantic web service (represented using the method proposed in [12]). Section 4 describes the proposed multi-agent system for the composition of semantic web services. In Section 5 we present the main algorithms used by our system. Section 6 contains an example of using the proposed system. In Section 7 we present some experimental results. Finally, Section 8 contains the conclusions of the paper.

2 Representation of the Semantic Description Of A Semantic Web Service

In this section we present some concepts proposed in [12] that are necessary for understanding the system proposed in this paper.

2.1 Dictionary

In [12], we first consider a set of words *W*, in which each word is considered to have a single meaning for the discussed knowledge domain. Then, we define a binary relation $=_s$ that verifies if two words from *W* have the same meaning; this relation is an equivalence relation on *W*. We also consider the family of equivalence classes determined by $=_s$ on *W*, $(C_i)_{1 \le i \le NC}$, where *NC* represents the number of equivalence classes (all the words of a given class C_i have the same meaning). In this way $(C_i)_{1 \le i \le NC}$ can be seen as a partition of the set of words *W*. The family of equivalence classes $(C_i)_{1 \le i \le NC}$ is called a *dictionary*. The meaning of a word is the index of the equivalence class to which it belongs (for the formal definition of the meaning of a word, see [12]).

2.2 A Method to Represent a Semantic Description

In [12], we also propose a way of representing a semantic description as a complexity function, i.e. a function $f : N^* \to R_+^*$, where N^* is the set of positive integers and R_+^* is the set of positive real numbers. We will explain this representation method using an example. First we present a version of the function "mod" called "mod*"[12]: "$n \bmod^* NC = n \bmod NC$, if $n \bmod NC \neq 0$ and $n \bmod^* NC = NC$, if $n \bmod NC = 0$". Next, we consider the following example: we assume that the semantic description (of a web service) expressed in words is $w_1 w_2 w_3$ where w_1, w_2, w_3 are words from *W* such that $w_1 \in C_1$, $w_2 \in C_5$, $w_3 \in C_7$ (we assume that NC ≥ 7); then, the corresponding semantic description expressed as a complexity function has the form: $sd(n) = n + 1$, if $(n \bmod^* NC) \in \{1, 5, 7\}$ and $sd(n) = 1 / (n + 1)$, otherwise.

In addition, in [12], we propose two approximations of a semantic description. We explain here these approximations using two examples: 1) if the initial semantic description contains the word *animal*, an approximation of type 1 is a semantic description, similar with the initial one, that has the word *cat* instead of the word *animal* (the word *cat* is less general in terms of meaning than the word *animal*); 2) if the initial semantic description contains the word *cat*, an approximation of type 2 is a semantic description, similar with the initial one, that has the word *animal* instead of the word *cat* (the word *animal* is more general in terms of meaning than the word *cat*).

3 Enriching the Semantics of the Semantic Descriptions

In this section we enrich the semantics of the semantic descriptions (represented using the method proposed in [12]) by adding for each semantic description the following information: for each input value *n* of the function that represents the semantic description, we store the number of words of the initial semantic description expressed in words that belong to the equivalence class $C_{n \bmod^* NC}$.

We define the function *noApp* with the following form (see (1)):

DOI: 10.12948/issn14531305/18.2.2014.07

$$noApp: S \times N^* \to N^*$$

$$noApp(sd, n) = the\ number\ of\ words\ from\ sdw\ that\ belong\ to\ the\ equivalence\ class\ C_{n \bmod^* NC} \quad (1)$$

where by *S* we denoted the set of all semantic descriptions represented as complexity functions.

In Example 1 we show how this additional semantic information can be used by our system in order to extract a sub-semantic description from a given semantic description.

Example 1

Let be $NC = 100$. Consider two semantic descriptions $sd_1, sd_2 \in S$ with the following forms (see (2), (3)):

$$sd_1(n) = \begin{cases} n+1, & (n \bmod^* NC) \in \{25, 73, 91\} \\ 1/(n+1), & otherwise \end{cases} \quad (2)$$

$$sd_2(n) = \begin{cases} n+1, & (n \bmod^* NC) \in \{25, 91\} \\ 1/(n+1), & otherwise \end{cases} \quad (3)$$

The non-zero values of the function *noApp* that we need are (see (4)):

$$noApp(sd_1,25) = 2,\ noApp(sd_1,73) = 1,\ noApp(sd_1,91) = 1$$
$$noApp(sd_2,25) = 1,\ noApp(sd_2,91) = 1 \quad (4)$$

It is easy to observe that sd_2 is a sub-semantic description of sd_1. The semantic description that result after extracting sd_2 from sd_1 has the following form (see (5)):

$$sd_3(n) = \begin{cases} n+1, & (n \bmod^* NC) \in \{25, 73\} \\ 1/(n+1), & otherwise \end{cases} \quad (5)$$

and the non-zero values of the function *noApp* are (see (6)):

$$noApp(sd_3,25) = 1,\ noApp(sd_3,73) = 1 \quad (6)$$

Remark 1

For practical reasons, given a semantic description expressed as complexity function, $sd: N^* \to R_+^*$, our system will use a restriction of *sd*, defined on the set {1, 2, …, *NC*}. This restriction will represent all the information of the initial semantic description expressed in words, and it has the advantage that can be represented as a finite vector of real numbers. For examples and explanations, we will use the semantic description *sd* (not the restriction of this function).

4 Proposed System

The structure of the software system used for generating a web service by decomposing its semantic description is described in Figure 1 and it is composed by 8 modules: Processing Module, Decomposition Module, Semantic Descriptions Comparing Module, Words Module, Semantic Descriptions Approximation Module, Composition Module, Semantic Web Services Module, and Feedback Module.

4.1 Processing Module

Processing Module has a single component: *Processing Agent*. This agent receives the semantic description *SD* of the web service that must be generated by the system, runs the main algorithm, communicates with other agents (Decomposition Agent, Semantic Descriptions Comparing Agent, Words Inequality Comparing Agent, Words Equality Comparing Agent, Words Searching Agent, Semantic Descriptions Approximating

DOI: 10.12948/issn14531305/18.2.2014.07

Agent, Composition Agent, and Semantic Web Services Searching Agent) in order to solve the problem, and sends to the user the complete semantic description *CD* (i.e. all the information needed for using the composed semantic web service that corresponds to the semantic description *SD*). Processing Agent has also the role of translating a semantic description from the form expressed in words, into the form presented in Section 2.

During the solving process, Processing Agent sends to the user a proposal of decomposition of the semantic description *SD*. If the user response is affirmative then Processing Agent initiate the composition of the semantic web services indicated by the decomposition of *SD*. Otherwise, Semantic Descriptions Dictionary is updated and the part of the decomposition that was not satisfactory is repaired. It is also possible that Processing Agent to send an incomplete decomposition to the user in the case in which the decomposition algorithm cannot find a solution because of a wrong decomposition choice. In that case, the response of the user can improve the decomposition process.

The Processing Agent also has the following role: if for a given semantic description correspond several semantic web services, then the agent asks the user which is the correct choice in a given situation. Using the interactions with the human user related to this problem, the agent computes a ranking of the semantic web services for each such semantic description. Of course, this problem doesn't appear very often; consequently, the effect over the quantity of computational resources used by the system is negligible.

4.2 Decomposition Module

Decomposition Module has three components: *Semantic Descriptions Dictionary*, *Decomposition Agent*, and *Semantic Descriptions Generator Agent*. Semantic Descriptions Dictionary contains all the semantic descriptions that have been used, in the past, by the system, decreasingly ordered by importance (see, Algorithm 2). Given the semantic description *SD*, Semantic Descriptions Generator Agent generates, if asked by Decomposition Agent, semantic descriptions using some of the words within the semantic description *SD*.

Decomposition Agent must offer to Processing Agent a decomposition of the semantic description *SD*: first it searches in the Semantic Description Dictionary semantic descriptions that match with *SD* or with parts of *SD*, starting with the most trusted semantic descriptions; if, after this process, the decomposition is not complete, it asks Semantic Descriptions Generator Agent to generate disjoint semantic descriptions (in terms of similar words) using the words provided by Decomposition Agent. Decomposition Agent communicates with Processing Agent in order to solve words comparison (equality or inequality) or semantic descriptions comparisons (equality or inequality).

4.3 Semantic Descriptions Comparing Module

Semantic Descriptions Comparing Module has one component: *Semantic Descriptions Comparing Agent*. This agent compares two semantic descriptions (vectors of *NC* elements) in the same way in which two elements of the set $(R_+^*)^{NC}$ are compared. It receives from Processing Agent the two semantic descriptions and it sends back the result of the comparison.

DOI: 10.12948/issn14531305/18.2.2014.07

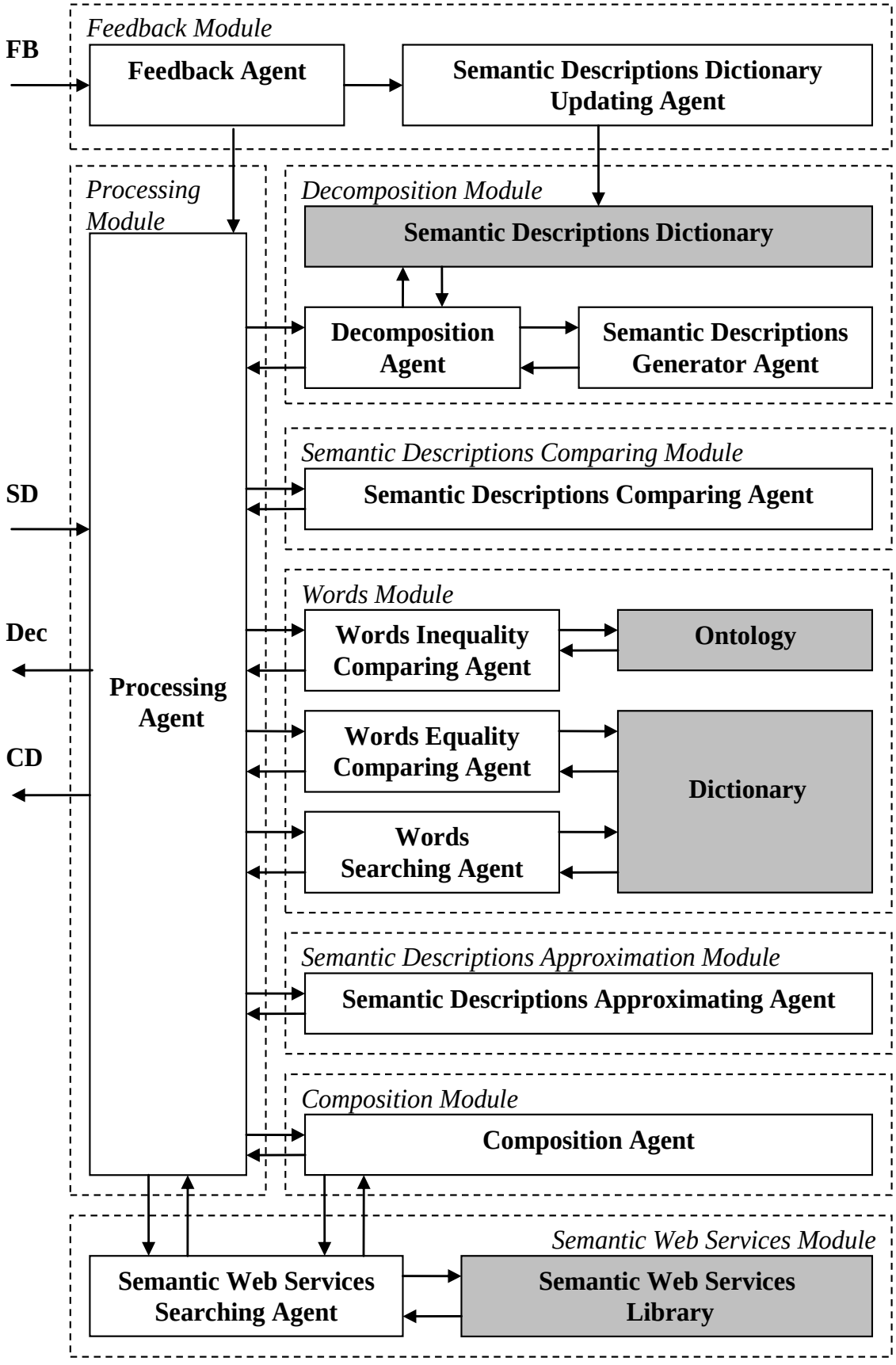

Fig. 1. System structure

DOI: 10.12948/issn14531305/18.2.2014.07

4.4 Words Module

Words Module has 5 components: *Dictionary*, *Ontology*, *Words Inequality Comparing Agent*, *Words Equality Comparing Agent*, and *Words Searching Agent*. Dictionary contains the words permitted by the system, grouped in classes of words with the same meaning. Ontology describes the inequality relations between the meanings corresponding to the classes of words from Dictionary, i.e. it provides all the meanings pairs ($meaning_1, meaning_2$) where $meaning_2$ is more general than $meaning_1$ (the meaning of the word *animal* is more general than the meaning of the word *cat*). Words Inequality Comparing Agent receives two words from Processing Agent, searches into Ontology, and finds the inequality relation between the two words, if such a relation exists. Words Equality Agent receives two words from Processing Agent, searches into Dictionary, and verifies if the two words belong to the same class from Dictionary (i.e. the two words have the same meaning). Words Searching Agent receives a word from Processing Agent, searches into Dictionary, and finds the corresponding class of that word.

4.5 Semantic Descriptions Approximation Module

Semantic Descriptions Approximation Module has a single component: *Semantic Descriptions Approximating Agent*. This agent receives from Processing Agent a semantic description, two indexes, and the approximation type needed (type 1 or type 2) and returns the approximation of that semantic description.

4.6 Composition Module

Composition Module has a single component: *Composition Agent*. This agent receives from Processing Agent a decomposition of the semantic description *SD*, and the information necessary for directly finding (i.e. without searching) the semantic web services (that correspond to the elementary semantic descriptions from the decomposition) in Semantic Web Services Library, makes the composition of these semantic web services, and returns the complete description *CD* of the composed semantic web service.

4.7 Semantic Web Services Module

Semantic Web Services Module has 2 components: *Semantic Web Services Library* and *Semantic Web Services Searching Agent*. Semantic Web Services Library is the library of semantic web services available to be used by the software system. Semantic Web Services Searching Agent has two roles: 1) it receives from Processing Agent a semantic description, searches into Semantic Web Services Library for the corresponding semantic web service, and if the service was found, it returns the information related to the position of the service in the library; 2) it receives from Composition Agent a semantic description of a semantic web service and the information related to the position of the semantic web service in Semantic Web Services Library, and returns the complete description of that service.

4.8 Feedback Module

Feedback Module has 2 components: *Feedback Agent* and *Semantic Descriptions Dictionary Updating Agent*. After Processing Agent sends to the user the complete description *CD*, Feedback Agent receives from the user a feedback message *FB* that contains some information related to the last process of generating a composed semantic web service related to the semantic description *SD*. If the result was accepted by the user, then the process of generating the composed semantic web service stops. If the result is not sufficiently good, then the process continues in order to solve the problems related to some parts of the semantic description *SD*. In both cases, Feedback Agent announces the situation to Processing Agent. We consider that, when the result is not sufficiently good, some parts of the semantic description *SD* were not correctly translated in terms of semantic web services from Semantic Web Services Library.

DOI: 10.12948/issn14531305/18.2.2014.07

Semantic Descriptions Dictionary Updating Agent receives from Feedback Agent the feedback message *FB* and modifies Semantic Descriptions Dictionary after the following rules: 1) if an elementary semantic description from the decomposition of *SD* was accepted by the user, then the agent searches for that description into Semantic Descriptions Dictionary: if the description is found then its trust coefficient is incremented; if the description is not found then the description is added to Semantic Description Dictionary and an initial trust coefficient is associated to the description; 2) if an elementary semantic description from the decomposition of *SD* was not accepted by the user, then Semantic Descriptions Dictionary is not modified with respect to that semantic description.

5 Main Algorithms

This section presents the main algorithm that describes the functionality of the system and the algorithms that describe the learning process of the system.

Dictionary contains words grouped by meaning in equivalence classes, Ontology contains the relations in terms of semantic inequalities between the representative words of the classes from Dictionary, and Semantic Descriptions Dictionary contains semantic descriptions as vectors of *NC* positive real numbers (see, Remark 1). The semantic descriptions from Semantic Descriptions Dictionary are decreasingly ordered by the trust coefficient (i.e. by importance).

For the algorithms we will use the following notations:

- SD: the initial semantic description represented in the form discussed in Remark 1
- SDD: Semantic Descriptions Dictionary
- SWSLib: Semantic Web Services Library
- SWS: the composed semantic web service that corresponds to the semantic description SD

5.1 The algorithm that describes the functionality of the system

Algorithm 1 represents the main algorithm used by the software system. In lines 3-6 the system searches for semantic descriptions in *SDD* that are included in the semantic description *SD*. If the entire semantic description *SD* could be represented using such semantic descriptions, then the decomposition process finishes. Otherwise, in lines 7-12, the system searches for semantic descriptions in *SDD* that approximate semantic descriptions included in *SD*.

If *SD* is still not empty, then in lines 13-30, the system generates a decomposition of *SD* and then it searches for the available semantic descriptions in *SWSLib* that matches exact or approximate the descriptions from the decomposition. It is possible that this part of the algorithm behaves like an infinite loop. For this reason, two temporal limits were added to the algorithm: 'limit' and 'finalLimit'. After this step of the algorithm we have two possible outcomes: 1) if *SD1* is empty, then the system created a complete decomposition of *SD*; 2) if *SD1* is not empty, then the system created only a partial decomposition of *SD*.

In lines 31-36 the system sends the decomposition *Dec* to the user. If the answer of the user is negative then the system applies again the decomposition steps from lines 3-30 in order to correct the parts of *Dec* that were not satisfactory. If the answer of the user is affirmative, then, in lines 37-38, the system creates the composed semantic web service that corresponds to the semantic description *SD* and it sends the composed service to the user.

DOI: 10.12948/issn14531305/18.2.2014.07

```
1: MainAlgorithm(SemanticDescription) {
2:    SD = Words2Function(SemanticDescription)
3:    for-each SDi in SDD, from the highest to the lowest trust coefficient
4:        while (SDi is a part of SD) {
5:            extract SDi from SD
6:            add SDi to the decomposition Dec }
7:    if (SD is not empty) {
8:        for-each SDj in SDD, from the highest to the lowest trust coefficient
9:            if (SDj is an approximation of SD) {
10:               add SDj to the decomposition Dec
11:               SD = empty
12:               break }
13:       SD1 = SD
14:       while ((SD1 is not empty) and (time_in_while < finalLimit)) {
15:           Dec1 = generate a decomposition of SD1
16:           for-each SDk in Dec1 {
17:               if(SDk in SWSLib) {
18:                   extract SDk from Dec1
19:                   add SDk to the decomposition Dec }
20:               if(SDk in Dec1)
21:                   if(SDk has an approximation in SWSLib) {
22:                       extract SDk from Dec1
23:                       add SDk to the decomposition Dec }}
24:           if(Dec1 is empty)
25:               SD1 = empty
26:           else if(time_for_the_same_SD1 < limit)
27:               SD1 = the description formed using the descriptions in Dec1
28:           else {
29:               SD1 = SD
30:               remove from Dec all SDk put in Dec during while }
31:    send Dec to user
32:    while(negative answer from user) {
33:        update SDD
34:        run lines 3-30 for the parts of Dec that are not satisfactory
35:        modify Dec
36:        send Dec to user }
37:    SWS = compose(Dec)
38:    send SWS to user
```

Algorithm 1. The algorithm that describes the functionality of the system

The system learns from the interactions with the human user. At the beginning the Semantic Descriptions Dictionary is empty. After several decomposition processes the system becomes sufficiently "intelligent" to propose decompositions with minor problems. Once the system has a certain experience in decomposing semantic descriptions, it will offer high quality decompositions, and with each mistake, thanks to the interaction with the user, it will perform even better.

5.2. The Algorithms that Describe the Learning Process of the System

The learning process consists of two parts:

- learning the importance of each semantic description that it uses, information stored in the Semantic Descriptions Dictionary; this process is described in Algorithm 2;

DOI: 10.12948/issn14531305/18.2.2014.07

- learning the best semantic web service that corresponds to each semantic description that it has ever used, if there are several semantic web services that correspond to this semantic description; this process is described in Algorithm 3.

Learning the importance of each semantic description that it uses

The system is trained by a human user for several sets of tests until it obtain a performance superior to a given bound. This type of learning process is can be done only during the training period. A step of the learning process takes place when the agent sends the proposed decomposition of the initial semantic description to the human user for acceptance. In Algorithm 2, we present this type of interaction between the system and the human user.

```
1: SDDLearningAlgorithm(Dec){
2:     Response = sendToHumanUser(Dec)
3:     for-each SDi in Dec{
4:         if(Response(SDi) = affirmative){
5:             if SDi in SDD
6:                 increaseScoreInSDD(SDi)
7:             else{
8:                 add(SDi, SDD)
9:                     initScoreInSDD(SDi)
}}}}
```

Algorithm 2. SDD Learning Algorithm

If the human user accepts the decomposition Dec, then for all the components of Dec the importance is increased in the Semantic Descriptions Dictionary (SDD): if a component is not in the SDD then it is added and its score is initialized; if a component is already in the SDD then its score is increased. If the human rejects the decomposition, then, for the correct components, the importance is increased in the Semantic Descriptions Dictionary as specified above, and for the incorrect components there is no change with respect to the importance in the SDD.

Learning the best semantic web service for a given semantic description

This type of learning process is done, occasionally, when the system needs to consult the human user. When the system uses a semantic description that corresponds to several semantic web services from the Semantic Web Services Library (SWSLib) it uses the following rules:

- if it never used that semantic description before, it asks the human user about the decision problem, initialize a ranking of semantic web services for that semantic description and uses the service with the highest score in the ranking;
- if the number of times that it used that semantic description is below a given bound *B* then it asks the human user about the decision problem, re-computes the ranking of semantic web services for that semantic description using the information that it already had and the information received from the user; then, it uses the service indicated by the human user;
- if the number of times that it used that semantic description is superior to *B* then the system uses the service with the highest score from the ranking of semantic web services that corresponds to that semantic description.

In Algorithm 3, we present this type of learning process:

```
1: BestSWSLearningAlgorithm(SD){
2:     Services = SWSsForSDfromSWSLib(SD)
3:     if(card(Services) > 1){
4:         if(noTimes(SD) = 1){
5:             Response = askHumanUser(SD, Services)
6:             initRanking(SD, Services, Response)
7:             return service(Response)}
8:         else if (noTimes(SD) < B){
9:             Response = askHumanUser(SD, Services)
```

DOI: 10.12948/issn14531305/18.2.2014.07

```
10:                recomputeRanking(SD, Services, Response)
11:                return service(Response)}
12:          else if (noTimes(SD) = B)
13:                return serviceWithHighestScore(SD)}
14:     return firstElement(Services) }
```

Algorithm 3. Learning the best SWS for a given semantic description

In the next section, we present an example of using the system proposed in this paper.

6 An Example of Using the Proposed System

In this section we present an example of using the system proposed in this paper for solving the problem of generating a web service by decomposing its semantic description. Starting from the "traveling scenario" presented in [15] and from the BravoAir service [16], we have created a simple traveling scenario: consider that a person wants to travel to another city and for this reason he wants to make a flight reservation, a car reservation, and a hotel reservation. The initial semantic description of a semantic web service that can accomplish this composed task is presented in Table 1:

Table 1. The initial semantic description expressed in words

This service offers flight reservation, car reservation, and hotel reservation

For simplicity, in this example we consider that the semantic description contains only a brief description of the task that the corresponding semantic web service can accomplish. Nevertheless, our system can also work with full information semantic descriptions.

Consider that we have 1000 equivalence classes in the dictionary *D*. In Table 2 we show the meaning of the words of our initial semantic description. The initial semantic description, expressed using complexity functions, which corresponds to the semantic description from Table 1 has the following form (see (7)):

$$sd(n) = \begin{cases} n+1, & (n \bmod^{*} 1000) \in \{24, 123, 152, 743\} \\ 1/(n+1), & otherwise \end{cases} \quad (7)$$

The non-zero values of the *noApp* function, which we need, are the following (see (8)):

$$noApp(sd, 24) = 1;\ noApp(sd, 123) = 1;\ noApp(sd, 152) = 1;\ noApp(sd, 743) = 3 \quad (8)$$

Table 2. The meaning of the words of the initial semantic description

Word	**Meaning**
this	-
service	-
offers	-
flight	123
reservation	743
car	24

DOI: 10.12948/issn14531305/18.2.2014.07

reservation	743
and	-
hotel	152
reservation	743

The Semantic Web Services Library is presented in Table 3:

Table 3. The Semantic Web Services Library

SWS Semantic Description	**SWS Name**
…	…
This service provides flight reservations	FlightReservation1
This service provides flight reservations	FlightReservation2
The service offers flight reservation	FlightReservation3
…	…
The service provides hotel reservation	HotelReservation1
…	…
This service offers car reservation	CarReservation1
The service provides car reservation	CarReservation2
…	…
The service offers hotel car	HotelCar1
…	…
The service provides reservations	Reservations1
…	…

The three semantic web services that offer flight reservation have the same semantic description, which is represented in (9). The semantic description of the semantic web service that offers hotel reservations is represented in (10). The two semantic web services that offer car reservations have the same semantic description, which is represented in (11).

$$sd_1(n) = \begin{cases} n+1, & (n \bmod^* 1000) \in \{123, 743\} \\ 1/(n+1), & otherwise \end{cases} \quad (9)$$

$$sd_2(n) = \begin{cases} n+1, & (n \bmod^* 1000) \in \{152, 743\} \\ 1/(n+1), & otherwise \end{cases} \quad (10)$$

$$sd_3(n) = \begin{cases} n+1, & (n \bmod^* 1000) \in \{24, 743\} \\ 1/(n+1), & otherwise \end{cases} \quad (11)$$

The non-zero values of the *noApp* function, which we need, are the following (see (12)):

$$noApp(sd_1, 123) = 1;\ noApp(sd_1, 743) = 1;$$
$$noApp(sd_2, 152) = 1;\ noApp(sd_2, 743) = 1;$$
$$noApp(sd_3, 24) = 1;\ noApp(sd_3, 743) = 1 \quad (12)$$

DOI: 10.12948/issn14531305/18.2.2014.07

We consider an intermediate case when our system can solve without human intervention a part of the decomposition, based of the information that it already learned from the human user; for the other part of the decomposition, the human intervention is necessary.

We consider that the Semantic Descriptions Dictionary contains the semantic description sd_1. Also, we consider that sd_1 has already been used for several times; therefore, the system already knows the best semantic web service that corresponds to sd_1; suppose that the best service for sd_1 is *FlightReservation1*. The semantic description sd_1 will be eliminated from *sd* and added to the decomposition. The new form of *sd* is the following (see (13), (14)):

$$sd(n) = \begin{cases} n+1, & (n \bmod^* 1000) \in \{24, 152, 743\} \\ 1/(n+1), & otherwise \end{cases} \quad (13)$$

$$noApp(sd, 24) = 1;\ noApp(sd, 152) = 1;\ noApp(sd, 743) = 2 \quad (14)$$

In this moment, the system must generate decomposition, because it has no useful information related to the most probable decomposition. Suppose that it proposes to the user the decomposition: sd_1, sd_4, sd_5, sd_5, where sd_4 and sd_5 are presented in (14), (15), (16). We assume that the system has found in the Semantic Web Services Library the semantic web services that correspond to sd_4 and sd_5.

$$sd_4(n) = \begin{cases} n+1, & (n \bmod^* 1000) \in \{24, 152\} \\ 1/(n+1), & otherwise \end{cases} \quad (15)$$

$$sd_5(n) = \begin{cases} n+1, & (n \bmod^* 1000) = 743 \\ 1/(n+1), & otherwise \end{cases} \quad (16)$$

The human user rejects the decomposition proposed by the system. Next, assume that the new decomposition proposed by the system is sd_1, sd_2, sd_3. The user accepts this new decomposition. The system finds in the Semantic Web Services Library the service *HotelReservation1* that corresponds to sd_2 and two semantic web services that correspond to sd_3. Therefore, the system must ask the human user for the best service with respect to this semantic description; suppose that the best choice is *CarReservation2*.

The composition workflow obtained by the system is presented in Table 4. For putting the semantic web services of the workflow in the correct order, the system must also analyze the initial semantic description expressed in words.

Table 4. The composition workflow

FlightReservation1, CarReservation2, HotelReservation1

In this example, the system needed two human interventions. After learning sufficient semantic information from the human user, the system will be able to solve this problem using only its own reasoning capacities.

DOI: 10.12948/issn14531305/18.2.2014.07

7 Experimental Results

In this section we present several experimental results based on the example presented in the previous section. We consider in the set of words *W* only words with reach semantics. The modified version of the initial semantic description from Table 1 is presented in Table 5:

Table 5. The initial semantic description expressed in words (basic form)

flight reservation car reservation hotel reservation

We will call this form *the basic form*: this form contains only the words with reach semantics (i.e. words relevant for the knowledge domain). We use the notation *SDBF* for this basic form. The meanings of the words from *SDBF* are presented in Table 2. The system will use the representation of a semantic description discussed in Remark 1. To easy the way a semantic description expressed using complexity functions is presented to the user, we introduce a new way of representing a semantic description (used only for displaying purposes): we consider only the equivalence classes C_i with the property from (17):

$$\exists w \in SDBF \; such \; that \; w \in C_i \quad (17)$$

For each such equivalence class we use a 3-tuple: (*index, value, classNoApp*), where *index* is the index of the equivalence class, value is (*index* + 1), and *classNoApp* is the number of words from *SDBF* contained in that equivalence class (for approximations of type 1, the system sometimes uses *value* = $(index + 1)^2$, see for details [12]; thus, it is not redundancy the use of (*index, index* + 1, ...) for an equivalence class for exact semantic descriptions). The first element of the description is the number of the classes used for the description. We call this type of description *user-style semantic description.* As an example we will first present the initial semantic description expressed using complexity functions, as in (7) and (8):

$$sd(n) = \begin{cases} n+1, & (n \bmod^* 1000) \in \{24, 123, 152, 743\} \\ 1/(n+1), & otherwise \end{cases}$$

$$noApp(sd, 24) = 1; \; noApp(sd, 123) = 1;$$
$$noApp(sd, 152) = 1; \; noApp(sd, 743) = 3$$

The corresponding user-style semantic description is presented in Table 6:

Table 6. The user-style semantic description of *SDBF*

4 24 25 1 123 124 1 152 153 1 743 744 3

Our experimental results will follow the same scenario used in the example presented the previous section: a part of the decomposition can be made by our system without human intervention and another part needs some information from the human user; the human intervention may be necessary for the decomposition process and for finding the best semantic web service for a given semantic description.

We modify the semantic web services from the Semantic Web Services Library (see Table 3) in order to use the basic form of a semantic description. For this new form, see Table 7.

DOI: 10.12948/issn14531305/18.2.2014.07

Table 7. The Semantic Web Service Library (basic forms)

SWS Semantic Description	SWS Name
…	…
flight reservations	FlightReservation1
flight reservations	FlightReservation2
flight reservation	FlightReservation3
…	…
hotel reservation	HotelReservation1
…	…
car reservation	CarReservation1
car reservation	CarReservation2
…	…
hotel car	HotelCar1
…	…
reservations	Reservations1
…	…

In Table 8 we present some information from *SDD* (Semantic Descriptions Dictionary). The semantic descriptions are sorted decreasingly according to the values associated to them. The semantic descriptions with the highest values are the first semantic descriptions used by the system. In other words, the value of a semantic description represents the priority associated to it by the system (high priority means high value).

Table 8. Some information from *SDD*

Value	Semantic Description
…	…
10	2 123 124 1 743 744 1
…	…
5	2 152 153 1 24 25 1
…	…
4	1 743 744 1
…	…
3	2 24 25 1 743 744 1
…	…
2	2 152 153 1 743 744 1
…	…

The type of a semantic web service can be: 0 – if the match is exact, 1 – for approximation of type 1, and 2 -for approximation of type 2.

For the given initial semantic description, the behavior of our system is presented in Table 9:

Table 9. Experimental results

Initial SD Length: 6 flight reservation car reservation hotel reservation The solution for the initial semantic description

DOI: 10.12948/issn14531305/18.2.2014.07

```
SD 1: 1 743 744 1 -> type: 0
SD 2: 1 743 744 1 -> type: 0
SD 3: 2 123 124 1 743 744 1 -> type: 0
SD 4: 2 24 25 1 152 153 1 -> type: 0

Your response (1: accept / 0: reject) is:
SD 1: 0
SD 2: 0
SD 3: 1
SD 4: 0

What is the best SWS for the SD: 2 24 25 1 743 744 1 ?
The choices are the following: CarReservation1(1) CarReservation2(2)
The best choice is (1-2): 2

The solution for the initial semantic description:
SD 1: 2 123 124 1 743 744 1 -> type: 0
SD 2: 2 24 25 1 743 744 1 -> type: 0
SD 3: 2 152 153 1 743 744 1 -> type: 0

Your response (1: accept / 0: reject) is:
SD 1: 1 (already validated)
SD 2: 1
SD 3: 1

Initial SD Length: 6
flight reservation car reservation hotel reservation

The result of the composition is the following (swsName (type)):
(Type: 0 – exact matching, 1 – approx type 1, 2 – approx type 2)
FlightReservation1 (0); CarReservation2 (0); HotelReservation1 (0);
```

The system already had the information related to the best semantic web service for the semantic description: 2 123 124 1 743 744 1; for the semantic description 2 152 153 1 743 744 1 there was only one semantic web service in the Semantic Web Service Library; for the semantic description 2 24 25 1 743 744 1 the system found two semantic web services in the Semantic Web Services Library, and for this reason it asked the human user to choose one of the two services.

This example is focused only on exact matching. The system is also capable of using semantic description approximations; this feature of the system is especially useful when using a big number of semantic web services and a big number of semantic descriptions. By using semantic description approximations, the system minimizes the time necessary for finding a solution; we consider that an exact solution that takes to much time to be obtained is weaker than an approximate solution obtained faster.

8 Conclusions

In this paper we proposed a multi-agent system for the composition of semantic web services. Our system uses the semantic descriptions representation method proposed by use in [12]. This design decision has an important advantage: our system represents the semantic descriptions using numbers, while most of the systems proposed in the literature

DOI: 10.12948/issn14531305/18.2.2014.07

for the composition of semantic web services use words for representing the semantic descriptions. Thus, our system has an advantage related to the running time.

On the other hand, given a knowledge domain, our system can be seen initially as semi-automatic. Then, using two learning algorithms, it gathers all the information needed related to that knowledge domain, and after several uses, the system can perform automatic compositions of semantic web services. In other words, given a certain knowledge domain, after a period of training, our system becomes automatic.

As future work, one can test the system for various knowledge domains, in order to see for which types of knowledge domains the system works better. Another future work can be the analysis of the training period of the system depending of the dimension of the knowledge domain, given a certain type of knowledge domain.

References

[1] A. H. Mogos and A. M. Florea, "Classification and Comparison of Several Semantic Web Services Composition Methods", accepted paper to *IE2014 – The 13th International Conference on Informatics in Economy*, Bucharest, Romania, 2014

[2] A. Gómez-Pérez, R. González-Cabero and M. Lama, "A Framework for Design and Composition of Semantic Web Services", in *Proc. The First International Semantic Web Services Symposium, AAAI 2004*, Spring Symposium Series, California, USA, 2004

[3] Z. Liu, A. Ranganathan and A. Riabov, "A Planning Approach for Message-Oriented Semantic Web Service Composition", in *Proc. The 22nd Conference on Artificial Intelligence (AAAI-07)*, vol. 2, Vancouver, British Columbia, Canada, 2007, pp. 1389-1394

[4] H. Mcheick and A. Hannech, "Semantic Web Services Adaptation and Composition Method", in *Proc. The Eighth International Conference on Internet and Web Applications and Services*, Rome, Italy, 2013, pp. 45-51

[5] F. Lécué and A. Delteil, "Making the Difference in Semantic Web Service Composition", in *Proc. The 22nd Conference on Artificial Intelligence (AAAI-07)*, vol. 2, Vancouver, British Columbia, Canada, 2007, pp. 1383-1388

[6] S. Kumar and R. B. Mishra, "Multi-Agent Based Semantic Web Service Composition Models", *Journal of Computer Science, Infocomp*, vol. 7, no. 3, pp. 42-51, 2008

[7] J. O. Gutierrez-Garcia, F. F. Ramos-Corchado and J. L. Koning, "Obligation-based Agent Conversations for Semantic Web Service Composition", in *Proc. The IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, Milano, Italy, 2009, pp. 411-417

[8] S. Sohrabi, N. Prokoshyna and S. A. McIlraith, "Web Service Composition via the Customization of Golog Programs with User Preferences", *Conceptual Modeling: Foundations and Applications: Essays in Honor of John Mylopoulos*, A. T. Borgida, V. K. Chaudhri, P. Giorgini and E. S. Yu (eds.), Springer-Verlag, 2009, pp. 319-334

[9] S. Liu, J. Wang, X. Feng, H. Park and D. Hyun, "Description Logic Rule Based Semantic Web Service Composition Method", *Journal of Computational Information Systems*, vol. 6, no. 8, pp. 2713-2725, 2010

[10] C. B. Pop, V. R. Chifu, I. Salomie, R. B. Baico, M. Dinsoreanu and G. Copil, "A Hybrid Firefly-inspired Approach for Optimal Semantic Web Service Composition", *Scalable Computing: Practice and Experience*, vol. 12, no. 3, pp. 363-369, 2011

[11] S. R. Dhore and M. U. Kharat, "QoS Based Web Services Composition using Ant Colony Optimization: Mobile Agent Approach", *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 1, no. 7, pp. 519-527, 2012

DOI: 10.12948/issn14531305/18.2.2014.07

[12] A. H. Mogos and A. M. Florea, "A Method to Represent the Semantic Description of a Web Service Based on Complexity Functions", submitted paper to *UPB Scientific Bulletin, Series A: Applied Mathematics and Physics*, 2013

[13] A. H. Mogos and A. M. Florea, "A Method to Compare Two Complexity Functions Using Complexity Classes", *UPB Scientific Bulletin, Series A: Applied Mathematics and Physics*, vol. 72, iss. 2, pp. 69-84, 2010

[14] C. A. Giumale. *Introducere in Analiza Algoritmilor. Teorie si Aplicatie* (Introduction to the Analysis of Algorithms. Theory and Application). Polirom, Bucharest, Romania, 2004, pp. 32-38

[15] R. Studer, S. Grimm and A. Abecker (eds.), *Semantic Web Services. Concepts, Technologies, and Applications*. Springer, 2007, pp. 52

[16] Bravo Air Service Example, OWL-S 1.1 Release. Internet: http://www.daml.org/services/owl-s/1.1/BravoAirService.owl, October 2004 [April 2014]

Andrei-Horia MOGOS, PhD, is Lecturer at the Faculty of Automatic Control and Computers of University POLITEHNICA of Bucharest. He is a member of the Artificial Intelligence and Multi-Agent Systems Laboratory (http://aimas.cs.pub.ro). His research interests include intelligent agents, semantic web services, complexity functions, and functional programming. He is the author of 17 research papers and 1 book, and has participated in several R&D projects.

Adina Magda FLOREA, PhD, is Dean of Faculty of Automatic Control and Computers of University POLITEHNICA of Bucharest and head of the Artificial Intelligence and Multi-Agent Systems Laboratory (http://aimas.cs.pub.ro). She has as research interests intelligent agents and agent-based computing, knowledge modeling, and ambient intelligence. Professor Florea is the author of more than 60 research papers and 4 books, and has coordinated several national and international R&D projects, among which the FP7 project ERRIC: Empowering Romanian Research on Intelligent Information Technologies.

DOI: 10.12948/issn14531305/18.2.2014.07