Journal of Engineering and Applied Sciences 13 (7): 1888-1906, 2018
ISSN: 1816-949X

A Phased Two Stages Conceptual Framework for Web Service Composition

Amina Adadi, Mohammed Berrada and Driss Chenouni
Laboratoire D'Informatique et De Physique Interdisciplinaire (LIPI), Ens De Fes,
Sidi Mohammed Ben Abdellah University, BP 2626 Route Imouzzer Fez, Morocco

Abstract: Web service composition is a key technology for creating value-added services by integrating available services. Nevertheless, it is one of the most challenging problems of the last decade in distributed and dynamic environment. Indeed in order to be more dynamic and less manual, most composition processes tend to be complex, unreliable and non-optimal. Inspired by researches in distributed systems, artificial intelligence and semantic web fields. This study addresses this issue by introducing a phased two-stage based conceptual framework for performing dynamic and automated web service composition. The key idea behind the proposed approach is to start with an abstract definition and to gradually make it concrete and executable. This increases reusability, flexibility and scalability of the composition system and leads to an optimized composition of web services. Within the framework, special emphasis is put on the composition techniques that have been designed in order to handle the main issues involved in the composition process.

Key words: Web service composition, semantic web services, multi-agent planning systems, artificial intelligence planning, self-healing

INTRODUCTION

Web service composition involves combining and coordinating a set of services with the purpose of achieving functionalities that cannot be realized through existing services. It is widely recognized, at the web-based ecosystem that in order to be viable and efficient a composition approach needs to implement dynamic and automated processes. Unfortunately, the rapid proliferation of web services makes the conception of such an approach a real challenge. Indeed while making composition more dynamic and less manual, issues of reliability, scalability, adaptability and efficiency rise. To deal with those issues, a robust and an optimal architecture is required to keep the composition process relevant.

Many approaches have been proposed to handle web service composition problem (Moghaddam and Davis, 2014; Sheng *et al*., 2014; Lemos *et al*., 2016; Jatoth *et al*., 2017) but most of them deal only with specific issues (selection, discovery, planning). Contrarily, our main contribution is to propose an architecture to carry a complete, accurate and successful composition process. The driving idea of our approach is to overcome the maximum amount of issues that are involved throughout composition life-cycle. To do so, we propose a phased two-stage architecture that tackles the composition problem in two levels: abstract and concrete, every level is composed by phases that resolve gradually the overall problem.

The proposed architecture is supported by a set of an intelligent composition techniques that have been developed in order to allow a system that implements our approach to handle dynamically and automatically issues spawned throughout the composition process. It includes: specifications analysis, planning model, discovery mechanism evaluation strategy and autonomic execution techniques. The result forms thereby the envisioned conceptual framework.

This study investigates on the other hand, the potential of applying some basic background concepts to Web service composition context. The proposal put forward by this research stems from the following underlying hypothesis: the synergy between four basic research areas belonging to semantic and artificial intelligence research sphere can lead to set up efficiently the aforementioned conceptual framework. We discuss therefore how to seamlessly and reliably transpose and interoperate the selected concepts in order to enable the expected added value.

Literature review: During the last decades, web services composition has been an active area of research and development endeavours for application integration and interoperation.

Corresponding Author: Amina Adadi, Laboratoire D'Informatique et De Physique Interdisciplinaire (LIPI), Ens De Fes, Sidi Mohammed Ben Abdellah University, BP 2626 Route Imouzzer Fez, Morocco

One important feature that distinguishes this area of research is that web service composition is a multifaceted problem. Indeed, the process of creating a composition schema in order to satisfy a series of goals set by a requester is a really complex and multifaceted problem, since, one has to deal with many different issues at once. First of all, it involves searching in an ever-growing global service repository in order to find matching services that may contribute to the complete satisfaction of the user's requirements. Assuming that these services have been found, one has to successfully combine them, resolving any conflicts and inconsistencies between them, since, they most certainly will be created by different people using different implementation languages and systems. Since, inconsistencies may occur at runtime, it may be necessary to predict such events, so as to ensure that the system will run correctly. Finally, even after having overcome these issues, we have to be able to adapt to the dynamic characteristics of service-based systems with services going offline, new services becoming online and existing services changing their characteristics.

Based on service composition literature (Moghaddam and Davis, 2014; Sheng *et al*., 2014; Lemos *et al*., 2016; Jatoth *et al*., 2017), A number of approaches have been presented to solve web service composition problem but most of them focus only on one aspect of the multifaceted composition process. Moreover, by surveying and analysis the proposed approaches in this research area, we notice that composition could be performed with different methods and technologies, according to the way the problem is perceived and the aspect on which the approach focuses.

Composition as planning: Composition can be seen as a planning problem which means decomposing the user request service into multiple tasks and assembling them in a plan to satisfy the user request. There are two planning methods applied to solve the composition problem: workflow-based methods and artificial intelligence planning methods, the latter class of method include basically: planning with control algorithms, planning as satisfiability algorithms, graph-based planning algorithms, logic-based planning algorithms and state-space based planning algorithms (Ghallab *et al*., 2016).

Composition as discovering: Discovery is the process of searching for services where functional and non-functional requirement meets user's needs. By considering the composition as a discovery problem, one of the following method could be used (Pakari *et al*., 2014): syntactic-based method, context-aware method, peer-to-peer method and semantic-based method that includes agent-based and ontology based solutions.

Composition as selection: The approaches that focus on the selection of the optimal atomic service to be combined with other services to perform complex composite service with the most satisfaction of quality of service values (Moghaddam and Davis, 2014), mostly use optimization based methods: Integer Linear Programming, genetic algorithm, constraint satisfaction and stochastic programming (Moghaddam and Davis, 2014).

Composition as execution: Some composition approaches focus on executing and monitoring the composite service. This process involves binding with participant services providers, invoking services, passing data between services and verifying the existence and quality of services of the composition plan (AlSedrani and Touir, 2016). There are four methods to monitor a dynamic execution: task-based, goal-based, specification-based and event-driven (AlSedrani and Touir, 2016).

Another important limitation related to the existent web service composition approaches, is the lack of the composition techniques. Since, most of web service composition works deal with a specific sub-problem of the composition process, usually the proposed solutions consist of faithfully using some technologies without any adaptation. So in order to be solved, the composition problem is projected on some other fields instead of being conceive holistically by transposing seamlessly the technologies belonging to other fields in the web service world and making theme work synergistically.

To illustrate the remarks put forward in this section, we present some of the recent significant works that propose a solutions to the dynamic web service composition issues:

By Mier (2016), the researchers propose a novel control-centric approach based on a genetic algorithm to generate composition workflows minimizing the number of services and maximizing the parallel execution of services. This approach focuses on the web service discovery process and uses semantic web service and genetic programming.

Study (Michael and Siva, 2016) proposes an agent based model that uses the dependency relation algorithm to identify the web services which can participate in the composition. It focuses on the web service discovery and execution processes and uses the intelligent agent technology.

The research presented by Lei (2016) uses an uncertainty planning method to deal with the uncertain planning problem for service composition. It focuses on web service selection and planning processes by managing the uncertain values of the web service QoS (Quality of Services). This approach is based essentially on artificial intelligence planning algorithms.

A context-aware solution to improve web service discovery and user-service interaction is proposed by Torre *et al.* (2016), this approaches supports the discovery process using a context aware method while (Chandrasekar and JayaShree, 2016) supports the same process using a clustering method, it claims that it is necessary to cluster the web services to recognize the specific web service and make the service discovery much more effective. The clustering method proposed by the work employs web services ontology.

One of the recent interesting works in the web service composition fields is describes by Chatzidimitriou and Koumpis (2008). This study deals with testing web services execution in a dynamic environment in order to detect fault at runtime, it proposes an adaptive random testing algorithm for testing composite web services. To support the continuous operation and changes of the dynamic execution, the proposed system performs automated reconfigurations for service based applications at runtime.

The above researches along with the review of recent literature in the web services composition field (Moghaddam and Davis, 2014; Pakari *et al.*, 2014; Sheng *et al.*, 2014; Lemos *et al.*, 2016; Jatoth *et al.*, 2017; AlSedrani and Touir, 2016) confirm the three remarks outlined previously, namely: most of the proposed approaches deal only with specific issues (selection, discovery, planning) of the composition lifecycle they adopt a specific technology to implement the composition process (planning, semantic web services, agents) and they do not emphasize the composition techniques used during the composition process.

In the research, we attempt to overcome all the sub-problems involved in the web service composition. Indeed, toward dynamic and automated web service composition is the subject of our researchers as it has been argued by Adadi *et al.* (2014, 2015a, b) our goal is to automate the whole composition process by proposing an end to end composition approach that handles the composition lifecycle from the reception of the user request until the return of the desired service to the requester.

To this aim, we conceived a multi-phase lifecycle model of service composition and designed a conceptual framework driven by five composition techniques that support this lifecycle. Furthermore and to solve the composition problem four research directions inspired us; Intelligent agents technology (Yu *et al.*, 2016), Semantic Web (SW) vision (Laufer, 2015), Autonomic Computing (AC) paradigm (Abeywickrama and Ovaska, 2017) and Artificial Intelligence Planning (AIP) algorithms (Ghallab *et al.*, 2016). By combining the strengths of all the above basic domains we believe we can reach an efficient web service composition system.

According to our perception, designing self-healing, multi-agent based SWS composition solution using AIP techniques is a relevant trade-off. The multi-agent based architecture is a step toward dynamic, decentralized and scalable service composition where using SWS approach makes the process more automatic. AIP methods allow building service composition without a manual effort and self-healing property ensures the reliability of the composition process.

Figure 1 illustrates the relation between the background domains. Agents act on behalf of SWS while SWS specifications represent the requirements and capabilities of an agent. Semantic web supports the interoperation of multiple agents by mean of shared ontologies. Self-healing behavior is implemented by one of the architecture's agents which is considered as an autonomic agent. Self-healing mechanism adapts (fixes) composition plan produced by AIP methods. Hence, using AIP techniques in the context of a multi-agent architecture forms a multi-agent planning system (Weerdt and Clement, 2009).

To fully harness and sharpen the strengths of the four aforementioned fundamental research areas, we introduced in a previous research (Adadi *et al.*, 2014) a conceptual architecture for web service composition populated by a set of intelligent agents and ontologies. Indeed six types of agents capable of automatically discover, compose, invoke and monitor WS have been designed, namely Request Handler Agent (RHA), discover agent, planner agent, executer agent, evaluator agent and manager agent. Besides agents the other key element of the proposed architecture is ontologies. In order to successfully carry out their assigned tasks,

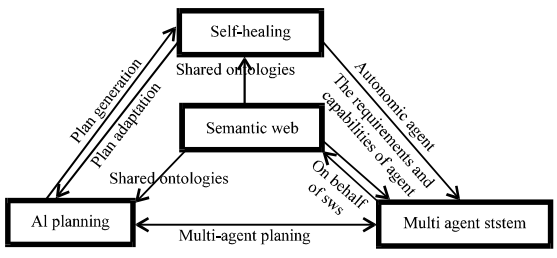

Fig. 1: Interaction between the four background concepts

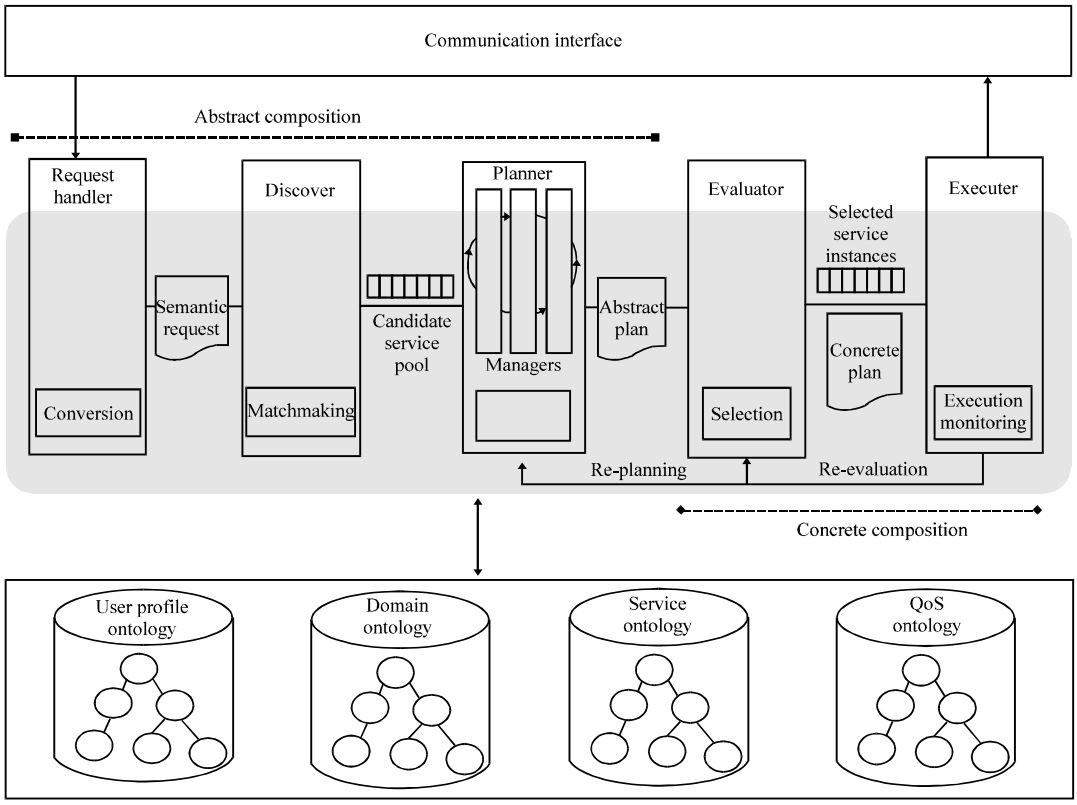

Fig. 2: The high level architecture of the proposed approach

intelligent agents access to various information stored in four repositories; domain ontology contains the domain's types, concepts and relations among them. Service ontology plays the role of a UDDI registry with extended functionalities to allow storing of service non-functional attributes (e.g., Quality of Service (QoS)). QOS ontology specifies shared knowledge and vocabularies about QoS properties and their relationship with respect to semantic services. User profile ontology incorporates concepts and properties used to model the user preferences. These constituent elements will be recalled in Fig. 2.

Next, we will shed light on the architectural style used to organize the identified elements in order to achieve the expected synergy between them.

MATERIALS AND METHODS

The proposed approach

A phased two-stage approach: Convinced as we are that composition of web services is a complex problem that involves other sub-problems and that modularity, scalability and reusability are key factors for the success of its resolution. We propose to organize the identified key actors into a phased system based on a two-staged approach.

According to the proposed approach the composition process is conceived as a cycle carried out in four phases, namely analysis, planning, evaluation and execution. Each phase is responsible for the resolution of a part of the problem in a continuous and independent way that guarantees the resolution of the overall problem. Indeed, a phase can use its own specific methods to accomplish its role in the composition but its outputs must necessarily form/feed seamlessly the inputs of the next phases in a continuous process. We herein define the responsibilities of each phase.

Analysis phase: We must distinguish between internal and external specification languages of services. External languages are used by users to express their needs they are often natural languages that lack accuracy and completeness. On the other hand, internal specifications, should be formal and precise to ensure the success of composition process. The shift from the user specifications to the system specifications is treated in this phase, it is considered as a perquisite for the rest of the phases as it defines clearly the goal of the ongoing composition process in a system understandable language. In this research, we use OWL-S as a semantic language to describe semantically the requested service as well as published web services.

Planning phase: Planning is a core sub-process, during this second phase a composition plan is generated in two steps namely, discovery and plan construction, in the first step all candidate services that can potentially lead to satisfying the desired functionality are filtered, in the second step the system composes and consolidates discovered services in order to build a valid plan.

Evaluation phase: Given a composition plan, an important challenge for the system is to find the "best" composite service execution plan with respect of user's specifications. The evaluation phase deals with this issue. It evaluates all alternatives solution and returns the best composite service for execution. As a result we obtain an execution plan.

Execution phase: Finally in the execution phase the system invokes the selected service instances according to the execution plan and delivers the result back to the user. But since, inconsistencies may occur at runtime, eventual re-planning and re-evaluating operations may be necessary to ensure that the system run correctly.

The idea behind the phased approach is to start from an abstract definition and make it progressively concrete and executable. This leads to the second characteristic of the proposed architectural style: the two-stage composition.

In order to solve the end to end service composition problem, we propose a principled two-stage composition approach. The staged approach is designed keeping in mind the best knowledge engineering practices of modularity, conciseness, and scalability. The composition first proceeds to generate an abstract plan based on web service classes (abstract composition) which is subsequently concretized into an executable plan by selecting the appropriate web service instances (concrete composition). In our two-stage web service composition approach we differentiate between web service classes (or types) which are groupings of similar (in terms of functionalities) web services and the actual web service instances that can be invoked. We believe that the separate representation of web service class definitions from instance definitions helps in handling different requirements and different means to optimize them. This in turn, allows us to work efficiently with large collection of web services and thus increasing the system scalability.

Practically, the system implements this approach in terms of two logical modules: abstract composer: that provides functional composition of service class to create new functionality that is currently not available, it is composed by analysis and planning phases and delivers an abstract composition plan. Concrete composer: this module enables the selection of component service instances based on non-functional requirements that would then be bound together for deploying the newly created composite service. It includes evaluation and execution phases.

As a result, the proposed system takes an end to end view that synergistically combines the strengths of the four basic background concepts into a phased approach splited into two stages of composition. Figure 2 presents the main building blocks of the proposed approach from a conceptual viewpoint.

The overall composition process: As depicted in Fig. 3, the composition process starts with an abstract specification and proceeds to a concrete (executable) one, it is initiated when a user (human or software agent) interacts with the communication interface requesting a service in order to answer the request the system must first understand it. To this end, the RHA uses specifications processing tools to translate the user request expressed in informal way into a semantic format (1) with the respect of domain and QoS ontologies. Based on this preliminary analysis, the system obtains a semantic, formal representation of the user goal. In case where user preferences option is enabled the RHA enriches the sematic request by adding user preferences retrieved from information stored in user profile ontology. As a result the request is formalized as a set of functional requirements and a set of non-functional requirements (user preferences), the functional part is sent to the discover agent (1.1) and the non-functional part is sent to the evaluator agent (1.2).

Discover agent first checks if any atomic/composite service in the service ontology repository matches exactly the requested service. If one or more services are found they are directly sent to the evaluator by this way the system can avoid unnecessarily wasted planning effort. Otherwise the discover agent looks for candidate services that can potentially participate to construct the new composite service. The search is based on a matchmaking algorithm using the service ontology repository and the domain ontology repository. The discovered services are sent to the planner agent.

Planner agent coordinates a set of Manager agents that act on behalf of discovered WS. They work cooperatively using AIP techniques to co-build an abstract composition plan which is then sent to the evaluator. At that moment, the second stage of the process begins where we deal with concrete instances of web services.

In order to select the best composite service, evaluator agent uses the non-functional part of the request sent by the RHA and QoS ontology to rank the concrete composite service alternatives through the use of QoS techniques. The highest ranked plan (closest to the user's goal) is sent to the executer that takes in charge the execution of the selected plan, it is also responsible

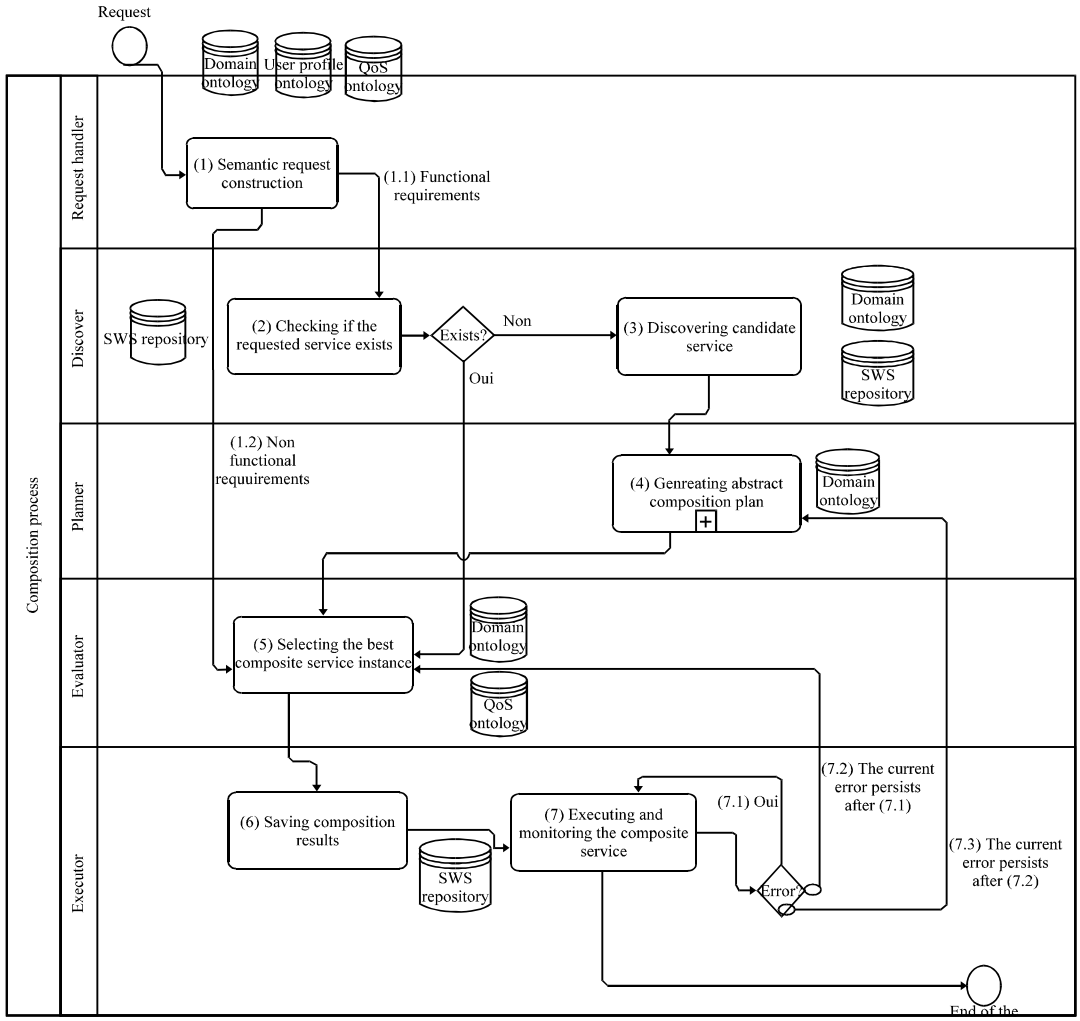

Fig. 3: The system's dynamic

of storing the generated composite service in the service repository for sharing and possible future reuse in others requests. Whenever a fault is detected, during the execution, executer agent tries first to re-execute the plan, this kind of solution is adequate when a web service is temporarily unavailable. If the fault persists, it tries respectively to re-evaluate and/or re-plan starting from the state of the world after the failure. The results are passed from the executer agent to the user through the communication interface.

RESULTS AND DISCUSSION

Composition techniques: A distinctive characteristic of our approach is that composition occurs in two stages: abstract and concrete stage. The concrete composition implements the result of the abstract one, it is based on

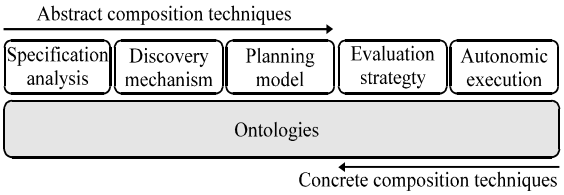

Fig. 4: Composition techniques

managing service non-functional parameters and fault tolerance techniques. The abstract composition aims, on the other hand, to produce an abstract composition plan, it makes use of semantic matchmaking techniques and AIP algorithms.

As illustrated in Fig. 4 three techniques lead the abstraction composition namely: specifications analysis, discovery mechanism and planning model. The

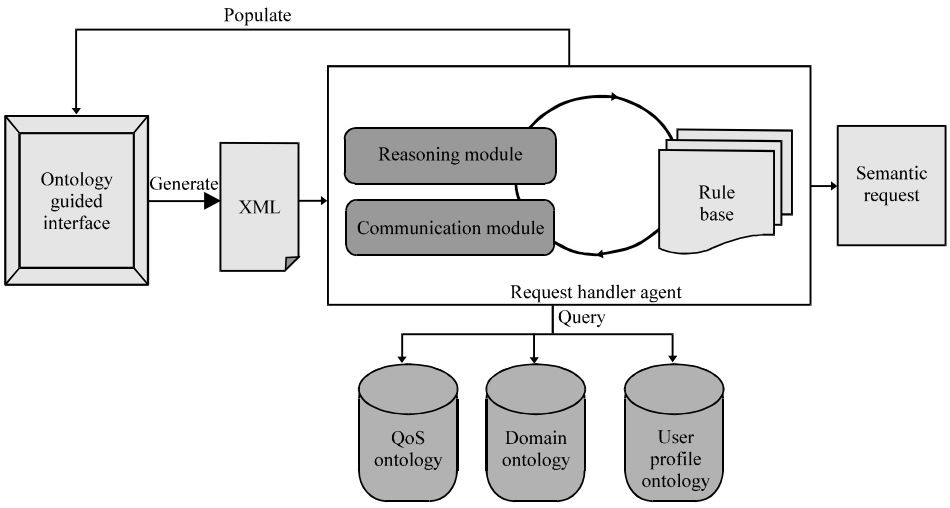

Fig. 5: Request handler agent architecture

evaluation strategy and the autonomic execution are the two techniques that animate the concrete composition. In this study, we present the five proposed composition techniques that have been developed in order to implement the composition process described.

Abstract composition techniques

Specification analysis: The RHA is responsible of converting the user request to a semantic one. To do so, it follows the three-step process shown in the following figure.

As described in Fig. 5, the RHA structure is composed by: a rules base including a set of rules used by the reasoning module to do the necessary transformations in order to obtain the OWL-S file that contains the semantic description of the request in terms of Input, Output, Preconditions and Effects (IOPE).

The reasoning module forms the agent's brain, it allows this latter to reason using the rules base and the domain and user profile ontologies in order to construct the semantic request.

Finally, the communication module is a module that every agent within our architecture have it allows an agent to exchange messages with the other agents involved in the composition cycle.

This structure allows the RHA to convert the user's request into a sematic one in three steps: XML file generation, preferences and metric data conversion and autocompleting of the semantic request.

XML file generation: In this research, we are not dealing with the natural language processing, thus, to express his/her needs, the user picks terms from a list provided by the communication interface these terms are populated based on the domain and the QoS ontologies this is why it is considered as an "Ontology-Guided Interface". This interface is also characterized by its dynamic incremental terms proposal based on the hierarchy and the relations defined in the ontologies.

The request description must include both the functional and non-function requirements. The former describes the functional characteristics of the demanded service such as name, textual description, inputs and outputs. The latter mainly focuses on the customer's preferences. In the research, the user doesn't have to give the value of each desired preference but he should get the means to specify that a preference attribute is more important than another one. Indeed, he gives a weight for each preference attribute. Weights range from 1-5 where higher weights represent greater importance.

Once the requirements are filled and the request is submitted, a XML file containing details about the information entered through the ontology guided interface is automatically generated.

Preferences and metric data conversion: The structure of the generated XML file looks like the following algorithm: The RHA browses the file and converts it using the rules stored in the rules base. We state below the two main rules used in the conversion:

```
<!-- User's request description-->
<request>
        <!-- request id -->
        <ID> xxx </ID>
        <description> texte </description>
        <!-- User's request functional requirement -->
        <Params>
                <Params>
                        <type>in1</type>
<value>v1<value>
                </Param>
                <Param>
```

```
                <type>out1</type>
            </Param>
</Param>
    <!-- User's preferences -->
    <preferences>
    <preference>
            <type> Pr1</type>
            <! -- A value between 1 and 5-->
            <weight> 2</weight>
    </preference>
  </preference>
</request>
```

Rule 1: For each tag <preference>, the sub-tag <type> defines the type of the preference: Pr_i and the sub-tag <weight> defines the associated weight w_i, add then to the list PRF that stores the preferences, the item: <Pr_i, w_i>.

By applying rule 1, we obtain the list of preferences and their weights, this list is then encapsulated into a message and sent directly to the evaluator if no automatic preferences are proposed.

Rule 2: The <param> tags with a <value> sub-tag defines the request's input while the empty <param> tags (without a sub-tag <value>) represent the request's output.

Knowing that the structure of the target semantic query (functional) is defined as: R = {R.in, R.out, R.pre, R.effe} where R.in R.out, R.pre and R.effe are respectively the semantic set of inputs, outputs, preconditions and effects. This rule helps the agent to define the metric part (input and outputs) of the semantic request.

Autocompleting of the semantic request: Now that the preferences list is initiated and the inputs and the outputs of the request are defined, the RHA completes automatically the description of the semantic request based on the domain ontology, by integrating the necessary conditions and the possible effects associated to the concepts that represent the predefined inputs and outputs. The user profile ontology is used thereafter to refine adjust (adapt) the resulting preconditions and effects. The stored elements in this ontology contribute also to the automatic enrichment of user preferences.

As defined, the three steps process allows the RHA to convert the XML file representing the user's request into a semantic one. This request that represents the starting point of the composition process is encapsulated in a message and sent to the discover while the list of preferences is sent directly to the evaluator.

Discovery mechanism: Discovering is a critical step in the proposed web services composition process. Instead of letting the planner totally unguided with a list of all available services registered to the system in the discovery phase the discover tries to reduce as much as possible the search space of services that need to be explored to generate the composition plan. This is considered as a second search space optimization as the first one is already guaranteed by the use of the two stage approach where only services type (that have different IOPEs) are treated at the first composition stage.

For this purpose, we propose a technique to determinate candidate services that can participate to achieve the desired functionalities based on semantic characteristics: Inputs, Outputs, Pre-conditions and Effects (IOPEs). The discovery mechanism is based on a matchmaking algorithm that takes as input a request (semantic request) as well as a service registry and returns as output a list of matched services types.

To ease the reading and the understanding of the proposed algorithm, we present first some formalisms used by the discovery mechanism.

Formalisms and considerations:

- Let S = {s_1, …, s_n} denotes the set of abstract services which are available (registered to our system)
- Each service $s_i \subset S$ is a triple: s_i = {s_i.in, s_i.out, s_i.pre, s_i.effe}, where s_i.in, s_i.out, s_i.pre and s_i.effe are respectively the semantic set of inputs, outputs, preconditions and effects of a s_i service
- Requested service (R) is semantically described by a triple: R = {R.in, R.out, R.pre, R.effe} where R.in, R.out, R.pre and R.effe are respectively the semantic set of inputs, outputs, preconditions and effects of the semantic request

We note that, since, the discovery mechanism is supposed to be as automatic as possible, we use semantic based similarity to filter relevant services, thus we distinguish four levels of semantic similarities between two concepts C_1 and C_2 based on their relationship defined in the domain ontology:

- Exact(C_1, C_2): if C_1 and C_2 have the same classes or C_1 is a direct sub-class of C_2
- Plug In(C_1, C_2): Si C_2 subsume C_1
- Subsume(C_1, C_2): Si C_1 subsume C_2
- Disjoint(C_1, C_2): no relation between C_2 and C_1 is defined in the ontology domain

We define then an overloading of "=" operator as follows:

$$C_1 = C_2 \begin{cases} \text{If Exact } (C_1, C_2) \\ \text{Or PlugIn } (C_1, C_2) \\ \text{Otherwise } C_1 \neq C_2 \end{cases}$$

Therefore, the inclusion operator ($\subset$) and the intersection operator (n) take in consideration the new "="operator.

The proposed matchmaking algorithm: The algorithm 6 is based on the idea that no matter what planning model we use, the composition plan can be represented as a graph, therefore in order to belong to the graph an abstract service needs to retrieve its input (preconditions) parameters from previous nodes and its output (effects) parameters have to be used as input parameters of successors nodes. To do this the algorithm constructs two abstract services communities, the input parameters of the services belonging to the first one can be retrieved (directly or indirectly) from the request's inputs (preconditions) and the output (effects) parameters of the services belonging to the second one are retrieved (directly or indirectly) from the request's outputs (effects) Clearly the intersection of the two communities belongs to the planning graph, since, its services have the following property: its IOPEs can be retrieved from the request's IOPEs as result the candidate services pool is constituted of the abstract services belonging to the intersection of the previous tree and the successor tree and its antecedents.

Finally, it is important to mention that besides filtering candidate services, the proposed algorithms gives an effective answer to a decisive question: do we have to do the planning task? Indeed, before any relevance computation the algorithm verifies first if any existent service can satisfy the request if so, the planning phase is skipped and the evaluator is directly called. The algorithm also verifies, at the end that candidate services pool is not empty, the emptiness of this latter means that the planning is not possible.

The proposed matchmaking algorithm:

Algo matchmaking algorithm (R, S)
(1) if $\exists s_z \in S$ where:
$s_z.in \subset R.in$ and
$s_z.pre \subset R.pre$ and
$R.out \subset s_z.out$ and
$R.effe \subset s_z.effe$
then call_Evaluator (s_z); // planning is not necessary
Else
(2) Construct PreviousTree where:
R is the root node
$\forall s_i \in S$, $s_i \in$ PreviousTree if and only if:
$s_i.pre \subset R.pre$ or $s_i.in \subset R.in$ then child(R) = s_i
Or $\exists s_j \in$ PreviousTree where:
$s_i.prens_j.effe \neq \varnothing$ Or $s_i.innS_j.out \neq \varnothing$ then child (s_j) = s_i
(3) Construct NextTree where:
R is the root node
$\forall s_i' \in S$, $s_i' \in$ NextTree if and only if:
$s_i'.effen\ R.effe \neq \varnothing$ Or $s_i.out \subset R.out$ then child(R)= s_i'
Or $\exists s_j' \in$ NextTree where:
$s_i'.effens_j'.pre \neq \varnothing$ Or $s_i'.outns_j'.in \neq \varnothing$
then child(s_j')=s_i'
(4) Construct CandidatePool where:
$\forall s_i \in S$, $s_i \in$ CandidatePool if :
$s_i \in$ (PreviousTree nNextTree)
Or $s_i \in$ PreviousTree and $\exists s_{ii} \in$ CandidatePool
where child(si) = s_{ii}
Or $s_i \in$ NextTree and $\exists s_{ii} \in$ CandidatePool
where child(s_i) = s_{ii}
(5) if CandidatePool is empty // planning is not possible
return failure
else call_Planner (R, CandidatePool);
end if
end if
end algo

We note that in this composition technique, the priority was given to the correctness and the completeness of the proposed matchmaking algorithm which is directly influenced by the presence of the false negatives. Indeed, the latter threaten the success of the planning phase which could fall if potential services have skipped the filtering and have not been put in the "candidatpool". The designed discovery mechanism does not present these types of risks as the filtering method used by the matchmaking algorithm considers the candidate services that satisfy directly or indirectly, completely or partially the request by reasoning semantically. The correctness and the completeness of the proposed matchmaking algorithm are therefore be guaranteed.

Planning model: The planning task lies at the heart of the proposed composition process. To accomplish this task a planning model has been set up in this research we draw our inspiration from the proof theory described by Pellier and Fiorino (2004). The model produces a valid composition plan based on an assumption-based planning approach where agents exchange proposals and counter-proposals in order to co-build a plan. In the proposed planning model, a planning problem is composed by a goal to reach and a set of agents that have to cooperate in order to reach that goal. Each agent has a knowledge base and a capacity base. This latter is formed by a set of operators, actions that an agent is capable to perform autonomously in a given environment. A solution plan resolves a planning problem while a partial plan or a conjecture resolves a planning problem under certain assumptions. In what follows, we consider the following formal definition of a conjecture, the pivotal concepts of the planning model: A conjecture is a tuple $\chi = (A, <, I, C)$ such as:

- $A = \{a_0, \ldots, a_n\}$ is a set of actions
- $<$ is a set of order constraints on the actions A like $a_i < a_j$, i.e., a_i precede a_j
- I is a set of instantiation constraints on variables of actions A like $x = y$, $x \neq y$ or $x = cst$ such as $cst \in D_x$ and D_x is the domain of x

- C is a set of causal links $a_i \rightarrow a_j^p$ such as a_i and a_j are two actions of A, the order constraint $a_i<a_j$ exists in <, the property p is an effect of a_i and a precondition of a_j and finally the instantiation constraints which connect variables of a_i and of a_j concerning the property p are contained in I

We note that:

- An assumption formulated by a conjecture χ = (A, <, I, C) is defined as a precondition p of an action $a_j \in A$ such as for all actions $a_i \subset A$; the causal link $\notin C$
- A conjecture χ = (A, <, B, C) is a solution plan to a planning problem, if χ has no assumption

The model's principle: The key idea behind the proposed planning model relies on the agent's capability to elaborate plans under partial knowledge and/or to produce plans that partially contradict its knowledge. In other words in order to reach a goal such an agent is able to provide a plan which could be executed if certain conditions were met, so, process does not fail if some conditions are not asserted in the knowledge base but rather proposes an assumption-based plan or a conjecture that becomes new goal for the other agents.

The planning sub-architecture is constituted of two kind of agents, manager agents which simulate the discovered web services and planner agent which has a twofold purpose: to monitor the status of all manager's interactions and to interact with the other agents involved in the platform (discover, evaluator). This should not be understood as a centralized control mechanism that hampers the decentralized vision of MAS. Instead, our aim is to centralize the global view of the planning subsystem on one entity in order to ease the communication task and to enhance the system modularity. The planner is initiated with the semantic description of the user goal (requested service). While the manager is initiated with the semantic description of the discovered service it simulates (Adadi *et al*., 2014).

By using their reasoning capabilities agents interact through acts of dialogue, the informational content of those acts is conjectures that refine the current plan. The planner has an additional component; the proof board, a public table that stores the different exchanges between agents. Accordingly, in order to implement the planning model, we need to design a reasoning algorithm to determinate dialogue rules that manage agent's interactions and to define the structure of the proof board.

Planning tools

Acts of communication: Acts of communication defines rules of dialogue that must be respected during the agents' interaction, since, we are using JADE (Anonymous, 2017) platform to implement a system prototype, the dialogue rules concept is simulated using the FIPA performatives, the used set of acts of communication is presented in Table 1.

Proof board: The planner has an additional component, the proof board, a table that stores managers' proposals. From an algorithmic point of view, the proof board is seen as a directed graph AND/OR whose AND vertices are conjectures and OR vertices are the assumptions formulated by the father conjecture vertices, the edges correspond to the transition operation proposed by the Managers. An outgoing edge from a vertex χ is a refinement that transforms χ into a successor χ'. A refinement is a transition operation that adds operators to prove an assumption. Therefore, multi-agent assumption-based planning is a search in the proof board from an initial conjecture to a node recognized as a solution plan.

The proof board is initiated by a particular conjecture that defines the goal. Since, preconditions are possibly assumptions, the propositions corresponding to the goal are represented as preconditions of a dummy operator a_n. Similarly, the initial state is represented as the effects of a dummy action a_0. The effects of a_0 define the union of the agent's beliefs. We make the assumption that the agent's beliefs are consistent (Fig. 6).

Reasoning algorithm: Defining dialogue rules and proof board is necessary to structure the communication between agents but does not specify the mechanism that guides the refinement of the plan, to answer this issue, we introduce a reasoning algorithm (Adadi *et al*., 2015b) that describes the manger agents reasoning loop. As shown by the algorithm in Fig. 9, at each iteration, an agent chooses a conjecture in the proof board and tries to refine it by adding causal links based on its capacity base, thereby advancing the search for a solution plan. The

Table 1: Acts of communication

Acts of communication	Roles
CFP	The action of calling for proposals to refine a given conjecture
Propose	Submitting a proposal to refine a given conjecture the
Failure	action of telling another agent that refinement of a given conjecture was attempted but the attempt failed
Query_if	The action of asking manager agents whether or not they can continue refine the current plan
Confirm	The manager informs the planner that it cannot refine the current plan anymore
Disconfirm	The manager informs the planner that it can continue refine the current plan
Inform	The action of an agent informing that the proof board contains a solution plan
Cancel	The action of the planner ordering managers to stop planning

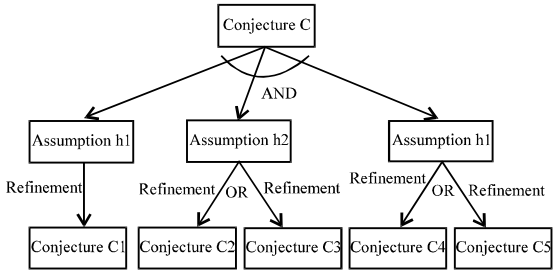

Fig. 6: The proof board structure

reasoning continues until finding a solution plan in the proof table. If all agents cannot refine the plan anymore the planning fails.

Algorithm 1: The XML file structure containing the user's request

```
While raisonne = true do
    plans←an unresolved conjecture from the proof board // (1)
    if plans is empty then
        submit a failure proposal
        raisonne←false
    else
        select a conjecture π∈ plans
        threats←Assumption (π)// Assumption (π) is the set of
        assumption formed by π
        if threats is empty then
                submit a success proposal of π
                raisonne←false; // (3)
        else
                select an assumption φ∈threats
                    refinements←Refine (φ, π); // by adding causal links
                to the conjecture (2)
                if refinements is empty then
                        submit a failure proposal of φ
                else
                        select a raffinement ρ as a refinements
                        submit ρ as a refinement of φ
end while
```

Planning process: After introducing our planning tools, let's now put all components together and try to describe the process through which the solution plan that describes the order of execution of the discovered services is built. The process is explained through an automate that defines the agents behaviour throughout the planning process (Fig. 7). Initially, agents are in IDLE state waiting for a goal to resolve. Upon the receipt of a call for proposal from the planner, the dialogue is opened and agents begin to exchange refinements and update the proof board based on defined dialogue rules. When the dialog is opened, agents go to the planning state. Where they keep executing their reasoning loop until a proposal of failure or success is sent. In the first case, the agents go to the IF state, the planner makes sure then that the current plan cannot be refined by any manager by asking

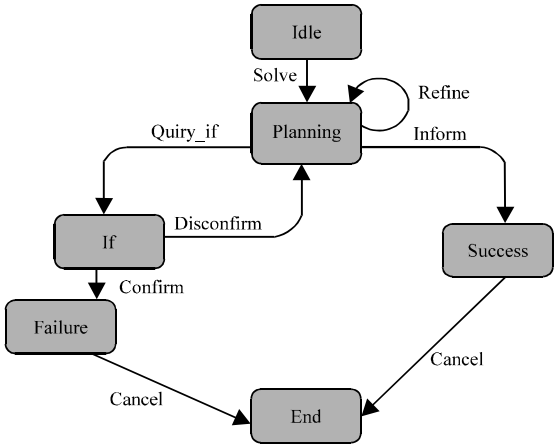

Fig. 7: Automate of agent's behavior

an acknowledgement from all agents. If all mangers acknowledge that they cannot refine the current plan, the dialogue is closed on failure. Otherwise, the agents go back to the planning state. In the second case the agents go directly to the success state, the planner makes sure that the conjecture existing in the proof board verifies the solution plan conditions and closes the dialogue on success.

Concrete composition techniques

Selection strategy: Web service selection is the problem of selecting the best offer made by a service provider given a request. In basic form, service selection involves mapping a set of services to a service based on the evaluation of the non-functional parameters (QoS). Multitude of service selection techniques and algorithms are proposed in the literature such as the use of optimization algorithm (Baldoni *et al*., 2007) for service selection, integer linear programming (Talantikite *et al*., 2009), broker-based architecture (Dong-Hoon *et al*., 2009) and negotiation model for service selection (Swarnamugi *et al*., 2010). However, the composition context brings to the table some fundamental questions.

How can we resolve semantic conflicts that might exist between providers and consumers understanding of QOS requirements, especially, the domain specific QoS? How to estimate the QoS of a composite service from those of the services bound to it? How to match between required QoS information and published QoS information of services?

Those questions have oriented our reflection towards the conception of an adapted selection strategy that takes in account the above considerations.

In order to select the proper one from the composite web service alternatives that verify the same functionalities, we propose firstly to set up a QoS ontology for web services and to use QoS extension of OWL-S to describe services non-functional parameters, then to compute the QoS attributes of the composite service alternatives, for this end we use the aggregation formulae proposed in Cardoso *et al*. (2004). Finally, the algorithm that evaluates comprehensively the alternative instances takes place to select the best offer.

Therefore, the model is based on three fundamental elements namely: QoS ontology, QoS computation and selection algorithm.

QOS ontology: QoS ontology is established to overcome the ontological conflict that may occur between consumers and providers. QoS ontology defines the QoS properties their relationships and establishes shared conceptions between the different stakeholders. The ontology ensures that providers uses a standard mean to express their QoS properties and ensures that users use the right mean to express their QoS preferences and therefore match that to the QoS information of providers. Our QoS ontology includes generic QoS attributes like response time, availability, price, reputation etc as well as domain-specific QoS attributes, for example, the accuracy of translation for a translation web services as long as these attributes can be quantified and represented by real numbers.

QoS computation of composite service: The QoS values of a composite service are determined by the QoS values of its component services and by the composition structure (e.g. sequential, parallel, conditional and/or loops). Here, we focus on the sequential composition model. Other models may be reduced or transformed to the sequential model. The QoS vector for a composite service CS = $\{s_1,\ldots,s_n\}$ is defined as Q_{cs}= $\{q_1(CS),\ldots, q_m(CS)\}$ where $q_k(CS)$ is the estimated end-to-end value of k^{th} QoS property and can be computed by aggregating the corresponding values of the component services qp (sj). In our model, we consider three types of QoS aggregation functions: summation, multiplication and minimum relation. Examples are given in Table 2.

Selection algorithm: The selection algorithm determines which composite web service alternative CS_i, from all possible alternatives CS = $\{CS_1,\ldots, CS_n\}$ is selected. For that purpose, the evaluator constructs Q_{nm} matrix where n represents the total number of composite web services alternatives (CS) that have the same functional properties and m represent the total number of QoS properties:

Table 2: Aggregation formulas of the QoS parameters in the composition

Aggregating type/Example	Function
Summation	
Response time	$q(CS)=1/n\sum_{j=1}^{n} qp(sf)$
Price	
Reputation	
Multiplication	
Availability	$q(CS)=\min_{j=1}^{n} qp(sf)$
Reliability	
Minimum	
Throughput	

$$Q=\begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1m} \\ q_{21} & q_{22} & \cdots & q_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ q_{n1} & q_{n2} & \cdots & q_{nm} \end{bmatrix} \quad (1)$$

Each row in this matrix represents a composite web service CS while each column represents one of the QoS properties q_j.

To compensate between different measurement units between different QoS properties values ($q_{i,j}$), the values need to be normalized to be in the range 0, 1. We will use the following equations to normalize them:

$$v_{i,j}=\begin{cases} \dfrac{q_{i,j}-q_j^{mm}}{q_j^{max}-q_j^{min}} & \text{si } q_j^{max}-q_j^{min}\neq 0 \\ 1 & \text{si } q_j^{max}-q_j^{min}=0 \end{cases} \quad (2)$$

$$v_{i,j}=\begin{cases} \dfrac{q_j^{max}-q_{i,j}}{q_j^{max}-q_j^{min}} & \text{si } q_j^{max}-q_j^{min}\neq 0 \\ 1 & \text{si } q_j^{max}-q_j^{min}=0 \end{cases} \quad (3)$$

where $q_{i,j}$ is the QoS property that we wish to normalize by minimization using Eq. 2 or maximization using Eq. 3, for example, response time needs to be normalized by minimization using Eq. 2 while availability needs to be normalized by maximization using Eq. 3.

q_j^{max} is the $q_{i,j}$ that has the maximum value among all values on column j and q_j^{min} is the $p_{i,j}$ that has the minimum value among all values on column j. We note Q' the resulted normalized matrix.

The key idea of the selection algorithm is to find the nearest CS_i to the QoS specifications of the user (preferences). To do so we use euclidean distance to calculate the distance between the user specified QoS properties (represented by a w vector that contains the weights for each QoS property) and the existing QoS properties for each vector in the matrix then we choose the CS with the minimum euclidean distance:

$$QoS(CS_i) = distance(Q', W) = \sum_{j=1}^{m} (v_{ij} \times w_j) \quad (4)$$

Where:
Q' = The normlized QoS matrix
w_j = The weight of the jth QoS property normlized with 1/5, i.e., $w_j \in [0, 1]$ and $w_j = 1$

Next, the obtained vector is sorted and the composite service with highest value (nearest to the user expectations) is sent to the executer. As a result, given a set of user preferences, the algorithm works as follows:

Step-1: Construct Q matrix
Step-2: Normalize QoS matrix using Eq. 2 and 3
Step-3: Compute the euclidian distance between each weight vector and the normalized QoS matrix using formulae 4
Step-4: Find CSi with the minimum distance

Based in this model the evaluator takes the abstract definition of the composite web service as an input and selects the best composite service instance that satisfies the user requirement. Practically the evaluator:

- Obtains the list of required QoS including user preferences with the respect of QoS ontology
- Estimates (calculates) the QoS of the composite service alternatives from those of the component services bound to it based on the composite service QoS computation mechanism
- Uses the selection algorithm to evaluate all alternative services and choose the best one

Autonomic execution: Autonomic is a basic concept in the proposed approach, indeed the defined composition process can be qualified as an autonomic one, since, it verifies the autonomic computing properties including self-configuration as the system discovers automatically the elements that participate in the process and configures them to meet the goal of the composition and self-optimization as the evaluation strategy chooses dynamically the best service instance that maximizes the quality. Self-healing on the other hand is a behaviour that allows the system to resist dynamically from runtime errors and it is at the execution phase that has been implemented.

The success the concrete composition plan execution is the responsibility of the executer, thus in order to maintain the execution of the composite service as it has been planned this one has to implement self-healing techniques which means it needs to monitor the service execution in order to predict problems and take action before complete service composition failure takes place. In autonomic computing paradigm this is known as the MAPE loop (Abeywickrama and Ovaska, 2017) which consists of four key activities (monitoring, analysis, planning and execution). However, even though the mape-loop is used by various applications in self-healing systems, it has its limitations when it comes to web service composition. One main drawback is that when a service fails to deliver it will then perform analysis and try to find another plan. However, re-planning might not be necessary if the service was delayed due to a slow network. In this case, a re-execution of the current plan might have solved the problem. Given this, an alternative self-healing architecture is needed for our web service composition approach. Next, we present the proposed self-healing architecture for the executer agent.

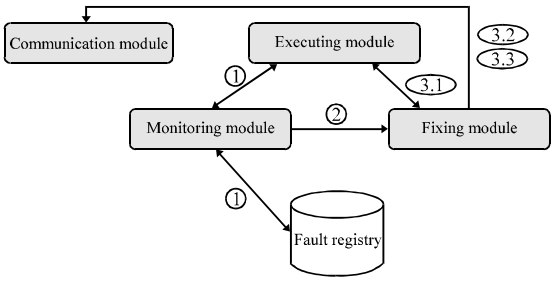

Fig. 8: Executer agent architecture

Executer agent architecture: The internal architecture of the executer that encapsulates the self-healing behaviour is shown in Fig. 8. The agent is composed by three modules, namely executing module, monitoring module and fixing module. In addition of the communication module that serves to communicate with the other agents.

Executing module: Executes the concrete plan using a service process engine.

Monitoring module: is fundamental since, in order to implement a self-healing behaviour, one must first know if something goes wrong as soon as possible in parallel to the execution of the composition process, a monitoring component listens to the events and detects abnormal ones according to the fault taxonomy that stores possible errors organized in categories. Since, this is done in parallel to the execution, almost no impact on performance is expected.

Fixing module: Allows the system to continue execution even after erroneous behaviours are discovered. In our research, we foresee three possible recovery actions:

- Re execute: simply tries to re-execute the service that is causing the problem

- Re select: tries to bind the execution to another service, capable of providing the same functionality of the faulty service and acceptable non-functional qualities
- Re plan: consists of re-invoking the planner to generate a new plan that, once executed correctly, provides the same functionality of the faulty service and acceptable non-functional qualities

These recovery strategies are applied hierarchically. If one strategy does not work, a second one can be tried subsequently.

Executer agent architecture: As shown in Fig. 11, the internal process of the executer can be described as follows: during the execution phase the plan is monitored at run-time to ensure that it is executed as expected based on fault registry. If at any stage during the execution, the monitoring module detects some unexpected occurrences, the monitoring module notifies the fixing module and the recovery actions takes place. There are three ways to overcome the problem, based on its severity: the first action to be taken is the re-execution of the current plan. In many cases, failure caused by an unresponsive web service or service delay due to a slow network can often be resolved after a few re-executions of the current plan. However, in the case where re-execution fails, fixing module checks for changes in the user's context, since, the last evaluation. If changes have occurred, it notifies the evaluator which then generates a new ranked alternative-services list. If no changes have occurred, the evaluator retrieves the next alternative composite service from the selected list and sends it to execution. In both cases, if it is the abstract composite service (all its instances) that causes the problem, the fixing module asks the planner to develop a partially new or completely new plan, from the current state to incorporate the latest available information.

Application and prototype

Application to e-Governement domain: The ultimate goal of the research goesbeyondthe introduction of a conceptual framework to carry dynamically and automatically the web services composition process. It intends also to prove its usefulness to solve real industrial problems.

It is obvious that building composite web services can save significant time and cost for developing new applications and enhancing the interoperability and collaboration among e-Business partners. However, we believe that e-Governement domain is where our web service composition approach can produce the best results to solve significant number of real world issues. Indeed, e-Government has some specific features as opposed to traditional e-Business scenarios, the e-Government domain is a large, heterogeneous, dynamic and shared information space with various semantic differences of interpretation. Because of the challenge it faces in achieving interoperability, integration and security, e-Government seems to be an obvious and promising application field for our web service composition approach.

We are particularly interested in the "one stop shop" concept (Chatzidimitriou and Koumpis, 2008) that supports the transition to the citizen-centric vision. In fact, in order to deliver seamless services to citizens it is necessary to enable a real interagency cooperation of government department which allows services and information sharing while reducing the need for users to provide redundant information already held by the authorities. Technically this requires the shift from isolated silos in public administration to one integrated government that delivers one-stop e-Government services (Chatzidimitriou and Koumpis, 2008) to the public encapsulating the size and complexity of government. This one-stop user experience will enhance overall user satisfaction of e-Government services.

Technology play a pivotal role in enabling this envisioned one-stop government as it can brings real solutions to integration, interoperability, coordination and security issues. In that way, we propose our conceptual framework as a technical tool to set up a one-stop government portal.

The application of the proposed approach in the e-governement domain to solve the integration, interoperability issues is guided by the following citizen/government relationship schema (Fig. 9):

A C2G and G2C interactions that mark respectively the beginning and the end of the administrative process they illustrates a front-office integration.

And a G2G interaction that encapsulates the mechanism of integrating (composing) the different public administration services in order to accomplish the administrative procedure requested by the citizen. This interaction illustrates a back-office integration.

In order to effectively implement the above relationship schema, we introduce the concept of "layer"

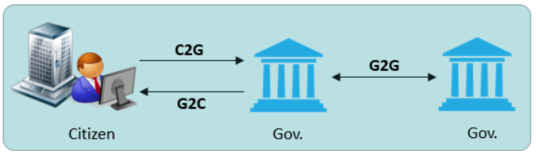

Fig. 9: Citizen/government relationship schema

in the proposed architecture illustrated in Fig. 2, thus, our composition architecture will be organized in four layers.

Legacy system layer: Contains the legacy applications available in each of the Public Administrations (PA) involved in the system.

Ontology layer: Ontology is the basic mechanism, we propose to capture semantic content of e-Government web services in a manner that ensure interoperability. That means that our system should integrates in addition of the three other ontologies: an e-Government domain ontology that contains the e-Government domain's types, concepts and relations among them.

Composition layer: Using the five enabling composition techniques this layer play the role of the system's brain by providing the required functionalities for a back office integration such as discovery, composition and execution of services.

Communication layer: The ontological user interface that manages the interaction between the citizen and the government. This layer ensures a front office integration by proposing a single and unique access point to the e-Governement services that integrates and merges any other access channel to these services.

As result by implementing the proposed conceptual framework in an e-Government context, the system will lead the citizen from his/her problem to the appropriate service or services by encapsulating the complexity of public institutions. Indeed with an architecture based essentially on intelligent agents and ontologies supported by a composition techniques toolkit, the middleware element formed by ontology and composition layers provides mechanisms to coordinate existing services in order to provide a new personalized service that is more convenient for user's needs and perceptions. This contributes to increase efficiency in terms of costs and productivity by reusing autonomous public administration services while keeping their internal processes and legacy systems intact.

Prototype: As a research in progress, the introduced framework has only begun to demonstrate its usefulness. Indeed, in addition of using the proposed approach to solve real world problematical issues, it is necessary on the other hand, to implement the different components of the proposed framework in order to develop real applications based on the conceptual composition theories invoked in this research.

By adopting an agile iterative approach, we have conducted experiments regarding a part of proposed composition process that focuses on the planning model technique and the creation of the domain ontology. The developed tool allows to produce an abstract composition planning that defines the order and the contribution of a set of a semantic described service types giving as input based on an e-Government domain ontology and a user request. The prototype includes the implementation of some utilities that ensure the translation from a semantic to a planning modelling language and vice versa.

In the rest of this study, we will expose the main deliverablesproduced throughout the prototype, namely: WebGov: the domain ontology and SWSComposer: the planning engine.

e-Government domain ontology: Building an e-Government domain ontology is an essential prerequisite to implement the integrated and interoperable government vision that we defend technically based on our conceptual framework.

Even if it is generally accepted that the formalism of new ontology by reusing existing ones involved less time, effort and cost, it is not always the best path to take. Most existing ontologies are hard to reuse, the benefits of ontology reengineering are then often unclear, since, the overhead of seeking and understanding existing ontologies may be even greater than simply building an ontology from scratch. As we are working on a proof of concept we chose to design and develop partial e-Government ontology instead of reusing existing e-Government ontologies. Which we believe is more appropriate in our case.

The creation of WebGov, the designed e-government domain ontology was guided by methonlogy (Sawsaa and Lu, 2012), recognized as the most mature and complete ontology development methodology.

Besides ontology development methodology, an ontology development language and ontology development tool have to be chosen too. We adopted OWL (Sawsaa and Lu, 2012) as the ontology development language. OWL was selected because of its computational quality of consistency in checking and classification which is crucial in developing coherent and useful ontological models for complex domains like government domain. In addition, Protege (Sawsaa Lu, 2012) was chosen as implementation tool because it is supported by methontology and it is widely used due to its platform-independent characteristics. The process of WebGov creation was splited into three main phases:

Specification: Where we acquired domain knowledge and we identified the scope of coverage knowledge. This phase was based on domain expert's interviews.

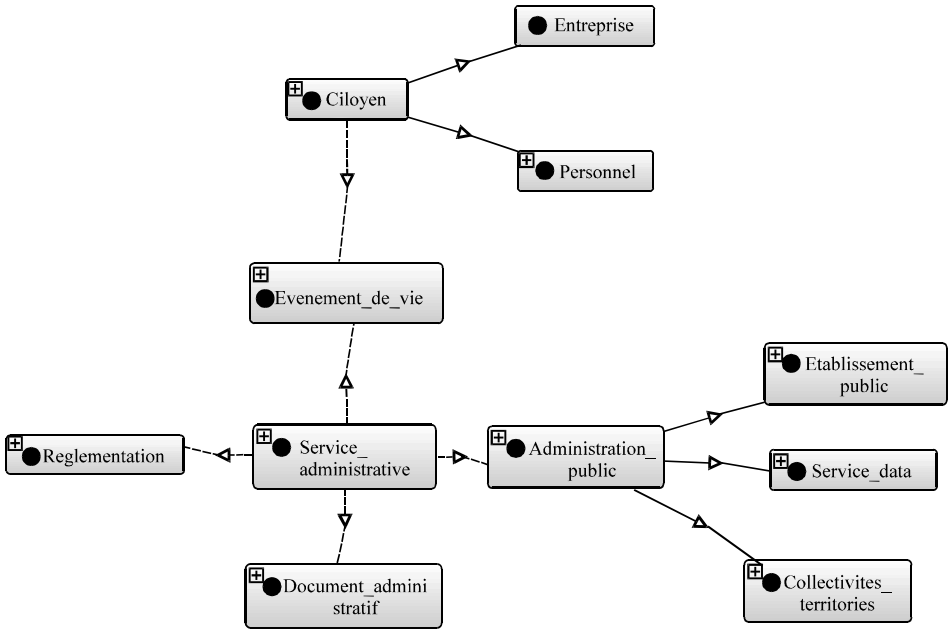

Fig. 10: WebGov domain ontology upper-level elements

Conceptualization: in this phase the acquired knowledge was structured in semi-formal specifications using intermediate representations that the domain expert and ontology developers can understand. This includes creating glossary of domain terms, building the concept classification tree and creating concepts dictionary.

Formalization and implementation: Finally, based on the conceptualization phase, protege formalizes the ontology by transforming the conceptual model into an implemented model.

Due to lack of space, the ontology upper-level elements are shown in Fig. 10. A fragment of the generated OWL file is given in Algorthm 4. The ontology consists of 52 concepts, 18 attributes, 17 relationships and 140 individuals. A much more detailed ontology would be needed if the system is to be applied in real-world e-Government settings. Actually, the purpose of the experiment is to demonstrate the capability of the framework to deal with the challenges it may face in this domain and the technical feasibility of the approach.

Fragment of OWL description of WebGov ontology

<owl: Object property rdf: about = "http: //local host/gov. Owl # Pssepar">
<rdfs: domain rdf: resource = "http: //local host/gov. Owl # citoyen
<rdfs: range rdf: resource = "http: //local host/gov. Owl # Evenement-de-vie"/>
<!-- http: //local host /gov. Owl # Produit -->
<Owl: Object property rdf: resource = "http: //local host/ gov. Owl # product">
<rdfs: range rdf: resources = resources "http: //local host/ gov. Owl # Documents-Administratif"/>
<rdfs: domain rdf: resources = "http: //local host/ gov. Owl # service-Administratif"/>
</ owl: object property>
<!--http: //local host/gov. Owl # Evenement-de-vie-->
<owl: Class rdf: about = "http:// local host/ gov. Owl # Evenement-de-vie"/>
<!-- http: //local host/ gov. Owl # personne -->
<owl: Class rdf: about = "http: //local host/gov. Owl # personne">
<rdfs: subclassof rdf: resource = "http: //local host/gov. Owl # citoyen"/>
</owl: class
<!-- http: //local host/ gov. Owl # Service-Administratif -->
<owl: Class rdf: about = "http: //local host/gov. Owl # Service-Administratif"/>

We note that the validation of WebGov was carried out at three levels: the validation of the business knowledge representing the informal level of the ontology was based on the domain expert's opinion. The validation of the logical and structural aspects of ontology such as the logical problems inconsistency, incompleteness or redundancy was automatically treated by the development tools used. As regards the validation of the ontology content, at the formal level, i.e., its adequacy with the part of the real world that it represents, it was decided by the designers.

The built e-Government domain ontology will be used as a semantic shared knowledge representation by our five composition techniques. Notably, the planning model technique will use WebGov as a source to build agent's knowledge base. More detail about the deliverable related to this technique is given next.

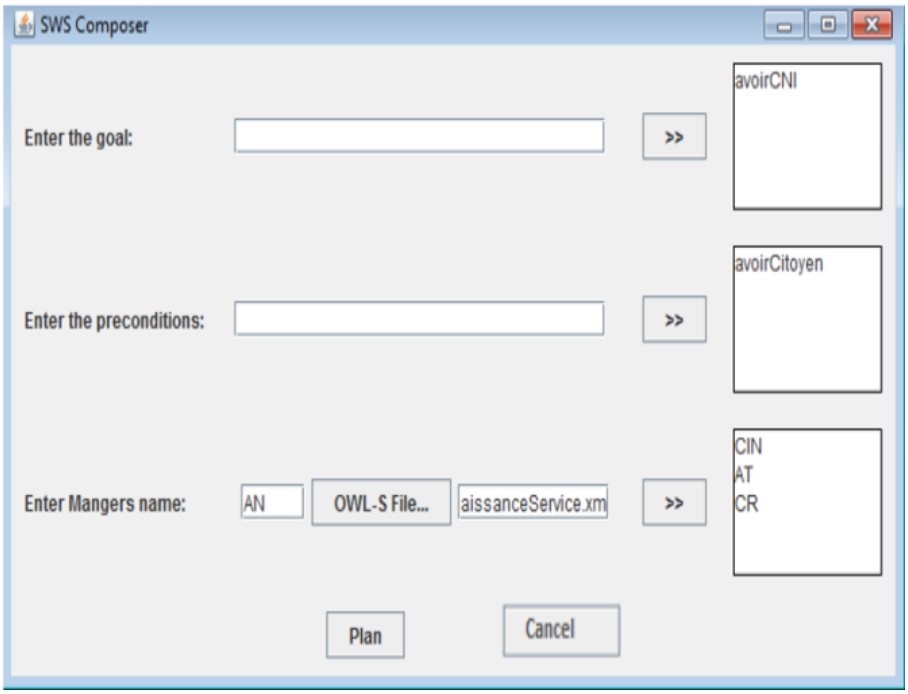

Fig. 11: The intermediate GUI for the planning engine (SWS composer)

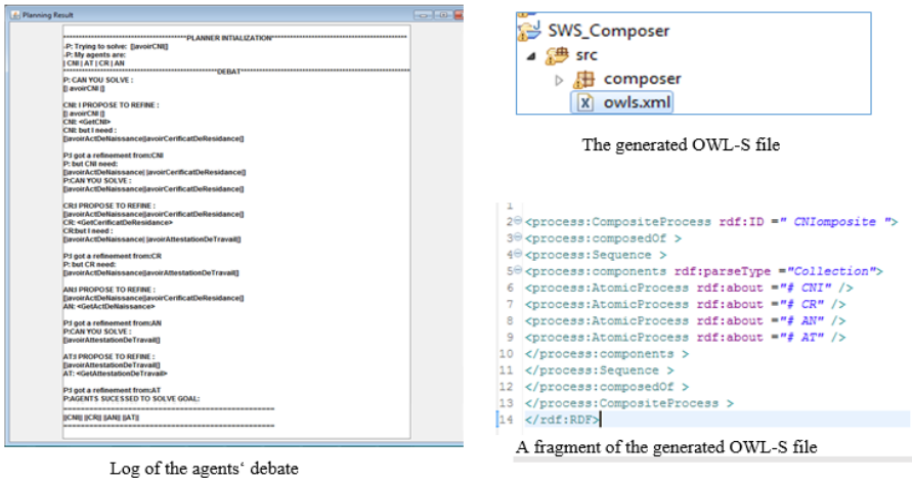

Fig. 12: The planning engine (SWS composer) outputs

Planning model implementation: As part of our proof of concept, the planning model was implemented using JADE platform (Anonymous, 2017). This includes the implementation of the planning tools detailed in section "4.1.3. Planning model", namely:

- The proof board, using Java data structures
- The MAS formed by Mangers agents and Planner agent, using Agent and Behaviors classes offered by JADE platform and FIPA-ACL performatives (Anonymous, 2017)
- The reasoning loop algorithm was also implemented in Java

In order to interact directly with the constructed planning engine (SWSComposer) an intermediate GUI was

set up, it allows to enter the semantic request in terms of goal and preconditions and to specify the semantic description of the component web services involved in the current composition process (Fig. 11).

By clicking "Plan", the planning model is launched. It makes use of the previously explained planning model, to generate the composition plan and then it uses an implementation of a "planning to semantic" algorithm to convert the composition plan to an OWL-S file that describes the composite services semantically. Figure 12 shows the planning model outputs, it includes the semantic description of the generated composition plan and a log of agents' debate during the planning process. This latter transcribes the agent's interactions through a composition lifecycle, it is useful for verifying the functioning of the SWS composer engine. Indeed By applying a use case belonging to e-government domain detailed by Adadi *et al.* (2015a, b), on the current prototype and after analyzing the generated conversation between the different agents involved in the composition process, we can confirm that the theatrical and the practical results are consistent.

CONCLUSION

In this research, we presented an overview of our phased two stage approach for Web service composition. We combined different emerging concepts such as ontologies, autonomic computing and artificial intelligence technologies to enable the semantic reconciliation among heterogeneous web services and to make composite web service construction dynamic and automated.

The proposed conceptual framework relies on five composition techniques: Specification analysis converts the user's request into a semantic one. Discovery mechanism filters irrelevant service types by using a matchmaking algorithm in order to propose only potential candidate services for the planning operation. Planning model allows to construct an abstract composition plan based on a distributed multi-agent planning method. Selection strategy chooses the best service offer based on non-functional requirements (QoS). Finally, autonomic execution ensures the reliability of the execution process by enabling a self-healing behaviour.

Considering the introduced techniques, we believe that our proposal is portraying a promising picture towards solving composition problem in autonomic and adapted way.

RECOMMENDATIONS

Our future research revolves basically around two aspects. Firstly, we will continue enhancing the pilot experiment to include the rest of the framework components and thus, covering the whole composition process. Secondly, we plan to experience our solution with a range of business processes such as e-Commerce, e-Learning, e-Health and e-Entertainment, to extensively evaluate its effectiveness.

REFERENCES

Abeywickrama, D.B. and E. Ovaska, 2017. A survey of autonomic computing methods in digital service ecosystems. Serv. Oriented Comput. Appl., 11: 1-31.

Adadi, A., M. Berrada and D. Chenouni, 2014. A multi-agent planning architecture for semantic web service composition. Intl. Rev. Comput. Software, 9: 347-354.

Adadi, A., M. Berrada, D. Chenouni and B. Bounabat, 2015b. A semantic web service composition for E-government services. J. Theor. Appl. Inf. Technol., 71: 460-467.

Adadi, A., M. Berrada, D. Chenouni and B. Bounabat, 2015a. Ontology based composition of E-government services using AI Planning. Proceedings of the 10th International Conference on Intelligent Systems: Theories and Applications (SITA'15), October 20-21, 2015, IEEE, Rabat, Morocco, ISBN:978-1-5090-0219-1, pp: 1-8.

Al-Sedrani, A. and A. Touir, 2016. Web service composition in dynamic environment: A comparative study. Proceedings of the 4th International Conference on Database and Data Mining, April 23-24, 2016, Coral Dubai Deira Hotel, Dubai, UAE., pp: 75-84.

Anonymous, 2017. Java agent development framework. Jade Ltd., Lagos, Nigeria. http://jade.tilab.com/.

Baldoni, M., C. Baroglio, A. Martelli and V. Patti, 2007. Reasoning about interaction protocols for customizing web service selection and composition. J. Logic Algebraic Programming, 70: 53-73.

Cardoso, J., A. Sheth and J. Miller, 2002. Modeling quality of service for workflows and Web service processes. Web Semant. Sci. Serv. Agents World Wide Web J., 1: 281-308.

Chandrasekar, M. and K. JayaShree, 2016. Clustering web services for effective service discovery. Intl. J. Adv. Res. Comput. Eng. Technol., 5: 2500-2503.

Chatzidimitriou, M. and A. Koumpis, 2008. Marketing one-stop E-government solutions: The European One StopGov project.IAENG. Intl. J.Comput. Sci., 35: 1-6.

Ghallab, M., D. Nau and P. Traverso, 2016. Automated Planning and Acting. Cambridge University Press, Cambridge, UK., ISBN:9781107037274, Pages: 354.

Jatoth, C., G.R. Gangadharan and R. Buyya, 2017. Computational intelligence based QoS-aware web service composition: A systematic literature review. IEEE. Trans. Serv. Comput., 10: 475-492.

Lei, Y., 2016. There can be composite services when you believe. Intl. J. Serv. Comput., 4: 49-58.

Lemos, A.L., F. Daniel and B. Benatallah, 2016. Web service composition: A survey of techniques and tools. ACM Comput. Surveys, Vol. 48. 10.1145/2831270

Michael, R.T.F. and P.P. Siva, 2016. Implementation of agent based web service composition using dependency relation algorithm. Intl. J. Technol. Enhancements Emerging Eng. Res., 4: 22-27.

Mier, P.R., 2016. A graph-based framework for optimal semantic web service composition. Ph.D Thesis, Universidade de Santiago de Compostela, Santiago de Compostela, Spain.

Moghaddam, M. and J.G. Davis, 2014. Service Selection in Web Service Composition: A Comparative Review of Existing Approaches. In: Web Services Foundations, Bouguettaya, A., Q. Sheng and F. Daniel (Eds.). Springer, New York, USA., ISBN:978-1-4614-7517-0, pp: 321-346.

Pakari, S., E. Kheirkhah and M. Jalali, 2014. Web service discovery methods and techniques: A review. Intl. J. Comput. Sci. Eng. Inf. Technol., 4: 1-14.

Pellier. D. and H. Fiorino, 2004. Assumption-based planning. Proceedings of the 2004 International Conference on Advances in Intelligence Systems Theory and Applications, November 25-18, 2004, Research Center Public Henri Tudor, Luxemburg, Europe, pp: 1-2.

Sawsaa, A. and J. Lu, 2012. Building Information Science ontology (OIS) with methontology and protege. J. Internet Technol. Secured Trans., 1: 100-109.

Sheng, Q.Z., X. Qiao, A.V. Vasilakos, C. Szabo and S. Bourne *et al*., 2014. Web services composition: A decade's overview. Inf. Sci., 280: 218-238.

Swarnamugi, M., M. Sathya and P. Dhavachelvan, 2010. A negotiation model for web service selection. Proceedings of the 2010 International Conference on Recent Trends in Soft Computing and Information Technology (RTSCIT'10), January 9-10, 2010, Institute of Science and Technology (AIST), Bhopal, India, pp: 251-256.

Talantikite, H.N., D. Aissani and N. Boudjlida, 2009. Semantic annotations for web services discovery and composition. Comput. Stand. Interfaces, 31: 1108-1117.

Torre, G.L., S. Monteleone, M. Cavallo, V. D'Amico and V. Catania, 2016. A context-aware solution to improve web service discovery and user-service interaction. Proceedings of the 2016 International IEEE Conference on Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld'16), July 18-21, 2016, IEEE, Toulouse, France, ISBN:978-1-5090-2772-9, pp: 180-187.

Weerdt, M.D. and B. Clement, 2009. Introduction to planning in multiagent systems. Multiagent Grid Syst., 5: 345-355.

Yu, W., G. Wen, G. Chen and J. Cao, 2016. Distributed Cooperative Control of Multi-Agent Systems. John Wiley & Sons, Hoboken, New Jersey, USA., ISBN:9781119246237, Pages: 264.