Towards a multi-agent planning based architecture for semantic Web Service Composition

Amina Adadi, Mohammed Berrada, Driss Chenouni
SIS Laboratory
ENSA Fez, Sidi Mohammed Ben Abdellah University
Fez, Morocco
Amina.adadi@gmail.com, mohammed-berrada@usmba.ac.ma, d_chenouni@yahoo.fr

Abstract - With the rising popularity of Web services due to their potential to enable interoperability between applications implemented on different platforms, new issues far more complex and crucial than interoperability have appeared. The present paper considers an important problem related to the further development of the semantic Web service technology – the automated dynamic composition of Web services. We propose a multi-agent planning architecture to compose semantic Web Services based on ontologies and Autonomic computing techniques.

Keywords—Web Service Composition; Semantic Web Services; Multi-agent systems; AI Planning; Self-Healing

I. INTRODUCTION

The Web service (WS) composition task consists of finding an appropriate combination of existing Web services to fulfil the user request, when none of the existing services is able to do this alone. One of the most challenging problems in designing a platform that support such process is how to compose services dynamically and automatically. Dynamic composition requires that existing services are combined in run-time (on-the-fly) and on demand whereas automatic composition supposes that the system is enough intelligent to carry the composition operation without human intervention. The full automation of this process stills an ongoing research activity.

In order to cope with a wide range of composition issues described above, we propose a multi-agent based architecture for WS composition. Our proposal stems from a basic underlying hypothesis: The synergy between four basic research areas namely; Semantic Web (SW) [1], Multi-Agent System (MAS) [2], Autonomic Computing [3] specially self-healing propriety and Artificial Intelligent Planning (AIP) techniques [4], can leads to the development of an effective platform that supports automated dynamic WS composition.

The current Web services infrastructure relies on WSDL [5] and UDDI [6] suffers from the follow limitation: It does not account for semantics information, it provide only the syntactic description of WS. Due to this lack of semantic expressiveness, human interaction is necessary to understand the functionality a service offers. So in order to support greater automation of service composition, we need richer semantic specifications of WS. In the field of the Web, the Semantic Web Services (SWS) approach [7] makes it possible by introducing Ontology [8] a formal explicit specification of a shared conceptualization that specifies semantics of a domain such as the terms and concepts of interest, their meanings and relationships between them.

A MAS is a distributed system composed of autonomous entities, called agents. Since agent possesses the ability to understand and interact with its environment several arguments [9][10][11] have been made to support the idea of integration of WS and agent infrastructure, including [12] where a web service is viewed as an abstract notion that must be implemented by an agent. Because of being context-aware, autonomous and able to interpret semantics with the help of ontological knowledge representation, we admit that agents are necessary to complete the vision of semantic web service composition. Hence within a framework of multi-agent planning, we can exploit similarities between composition problem and AIP planning problem [13] by using planning methods s [14], [15], [16], [17] to build the composition plan in AIP fashion.

Another important fact to be taking in consideration is that most of the business processes are required to work in the face of failure of their constituent parts. Therefore, it is important to make composite services resilient to failure in their execution environment. The Autonomic Computing architecture has been proposed as an approach for the development of systems and applications with automatic management. Self-healing is one of the four autonomic computing properties which makes the system to heal itself from the faults. According to the Autonomic Computing paradigm [18], to make composition process automatic the composition platform should implements the MAPE loop, which is the foundation of a self-healing system [19].

Considering these aspects, we believe that 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 composition systems more autonomous based on ontologies that describe the domain in a formal manner. AIP
methods allow building service composition without a manual effort and self-healing propriety ensures failure recovery.

The rest of the paper is organized as follows. First, we analyze related work in this area. Then, we present our proposed approach and we discuss the planning model using in our architecture. Finally, we present our conclusions and directions for future work.

II. RELATED WORK

This section discusses classes of composition systems. We compare these proposals to justify our architecture construction strategy.

Techniques of Web service composition can be grouped in two sub-families: techniques using an approach based on workflows (Business Process: orchestration and choreography) and those based on artificial intelligence techniques. In our work, we are interested in the second type of composition as it opens ways towards automated dynamic composition.

Works presented in [20],[21] and [22] demonstrates that AI Planning provides interesting tools for dynamic and automatic WS composition, most of these works uses intelligent agents and advanced the fact that the syntactic description (WSDL) of the services is not sufficient and proposed solutions based on a semantic description (OWL-S [23]). However, the most proposed architectures repose on a centralized composition which is not suitable in a context where we have to deal with distributed and heterogeneous WS. To overcome this problem, a dialectical theory for plan synthesis based on a multi-agent approach was introduced in [24]. It is a fully distributed model where agents reason collectively to produce a global shared plan by applying conjecture/refutation cycles. However, this approach needs to be adapted in order to apply it to our problem [25].

From the looking to the efforts that have been made in Web service composition field we notice that there are many open issues in the WS composition area, one common issue is that the integration of the different parts of the life-cycle for service composition has not been addressed that often. This is in our opinion a very important step to create and evaluate suitable solutions for real world applications. For this reason, we propose an approach that collects the best characteristics of the proposals analyzed in order to design an integrated WS composition with the respect of automation, dynamicity and distribution criteria.

III. PROPOSED APPROACH

Many approaches have been proposed to tackle Web Service composition problem, but most only deal with specific issues (selection, discovery, planning…), our main contribution is to propose an architecture to carry a complete, accurate and successful composition process. The driving idea of the approach we’re presenting in this paper is to overcome a maximum of issues that are involved throughout composition life-cycle, to do so we chose to enhance and leverage existing efforts in Web Service composition and make our proposed techniques more likely to be used in real world applications instead of designing a new system from scratch. In the rest of this section the foundations of the proposed approach are presented and the main elements of the architecture are enumerated.

A. Composition life cycle

The proposed approach [31] conceives the composition process as a cycle carried out in four phases showed in Fig.2.

![Fig. 1. Composition life-cycle](image)

The aim of the first phase is to define semantically the requested service starting from an informal, incomplete and ambiguous request. During the second phase a composition plan is generated in two steps namely, discovery and plan construction, the first step is to find all candidate services that can potentially lead to satisfying the desired functionality, the second step is to compose and consolidate discovered services in order to build a plan. If there is more than one composite service that meets user requirements then the system evaluates them and returns the best selected service for execution. Finally in execution phase the system invokes the service and delivers the result back to the user. Since inconsistencies may occur at runtime, eventual re-planning and reevaluating operations may be necessary to ensure that the system run correctly.

The idea behind the phased approach to service composition is to start with an abstract definition and gradually make it concrete and executable. This increase reusability and flexibility of the system.

B. Overall Architecture

Fig. 3 shows the high level architecture of our composition process, highlighting the main agents, repositories and their interactions.

![Fig. 2. Overall Architecture](image)
The conceptual architecture proposed is composed of three main components: a set of intelligent agents, four ontologies repositories and an interface for interacting with the user.

Six types of agents capable of automatically discover, compose, invoke and monitor WS, have been identified, namely Request Handler agent, Discover agent, Planner agent, Executer agent, Evaluator agent and Manager agent. A description of the all types of agents is given in Table 1. In this table only abstract, high-level descriptions of the tasks that each agent must perform are given.

Beside agents another key element of the proposed architecture is that it’s formed by a set of ontologies. In order to successfully carry out their assigned tasks, intelligent agents must have access to various information stored in four repositories; (a) Domain Ontology contains the domain’s types, concepts and relations among them. (b) Service Ontology plays the role of a UDDI registry with extended functionalities to allow the storing of service non-functional attributes (e.g. Quality of Service (QoS)). (c) QoS Ontology specifies shared knowledge and vocabularies about QoS proprieties and their relationship with respect to semantic services. (d) User profile Ontology incorporates concepts and properties used to model the user preferences.

The user interacts with the system through the User Interface and can initiates, cancels and restarts the composition. The results are passed to the user through the same interface.

As shown in Fig. 3, the composition process starts from an abstract specification and proceeds to a concrete (executable) one, it is initiated when a user (human or software agent) interacts with the user interface requesting a service, in order to answer the request the system must first understand it, to do this the Request Handler agent (1) uses natural language processing tools to translate the user request expressed in natural language sentence into a semantic format with the respect of domain ontology. 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 Request Handler agent enrich the semantic request by adding user preferences retrieved from information stored in user profile ontology (1.a). As a result the request is formalized as a set of functional requirements and a set of non-functional requirements (e.g. QoS parameters and user preferences), the functional part is sent to the Discover agent (2) and the non-functional part is sent to the Evaluator agent (5).

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 wasting planning effort. Otherwise the Discover looks for candidate services that can potentially participate to construct the new composite service. The search is based on a matchmaking algorithm [26] using the service ontology repository and the domain ontology repository. The discovered services are sent to the Planner agent (3).

Planner agent coordinates a set of Manager agents that act on behalf of discovered WS. They work cooperatively using IAP techniques [27] to co-build an abstract composition plan which is then sent to the Evaluator (4). At that moment, the second phase of the process begins, where we deal with concrete instances of WS.

In order to select the best composite service, Evaluator agent uses non-functional part of the request sent by the Request Handler agent (5) and QoS ontology to rank the alternative concrete composite service through the use of Quality of Service techniques [28]. The highest ranked plan (closest to user goals) is sent to the Executer that takes in charge the execution of the selected plan (6), it’s also responsible for storing the generated composite service in the service repository for sharing and possible future reuse in latter to other request. Whenever a fault is detected, during the execution, Executer agent tries first to re-execute (7.a) 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 (7.b) and/or re-plan (7.c) starting from the state of the world after the failure. The results are passed from the executer agent to the user through the User Interface (8).

<table>
<thead>
<tr>
<th>Agent type</th>
<th>Capabilities</th>
</tr>
</thead>
</table>
| Request handler| • Can retrieve functional requirements by matching the parameters in the request with the concepts of the domain ontology and describe them semantically in term of Input, Output, Precondition and Effects;  
• In order to adapt the request to his needs, the user may enable automatic preferences generating option, in this case the agent add personalized user preferences to the request using the user profile ontology and domain ontology;  
• Send the semantic functional requirements description to the discover and the preferences (if there is any) to the evaluator. |
| Discover       | • It has the capability of checking the service registry to see if any existing single WS or previously created composite service plans can fulfill the request;  
• It makes use of a specific matching algorithm to determinate a set of abstract services which once composed can offer the requested functionality. |
| Planner        | • The planning model produce a valid composition plan based on an assumption-based planning approach where manager agents exchange proposals and counter-proposals in order to co-build a plan. (our inspiration is drawn from the proof theory described in [36]);  
• The planner monitors the status of all the interactions and interacts with other types of agent (Discover, Evaluator);  
• It has also the responsibility to initialize the composition process by sending to the manager agents the goal to be realized. |
| Manager        | • It is initialized with the semantic description of the WS it stimulates. The translation mechanism allows to build the agent capability base (planning domain) based on owls-file[29];  
• Each agent has its own knowledge base and they interact with each other within a respect of the dialogue rules. |
Evaluator
- Obtains the list of required QoS including user preferences (if they are any);
- Estimates (calculates) the QoS of a composite service from those of the services bound to it based on QoS computation mechanism;
- Uses ranked matchmaking algorithm to construct the QoS function of all alternative composite service and choose the best one (with the highest QoS value).

Executor
- Invokes the concrete selected services with respect of the generated composition plan;
- Has the capability to detect failures autonomously and to recover from them due to its particular architecture based on MAPE loop.

IV. PLANNING MODEL

A. Principal
The planning task is at the heart of the architecture. In this paper, we draw our inspiration from the proof theory described in [29]. The model produce 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.

The key idea behind our approach relies on the agent’s capabilities 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 conjecture that becomes new goals for the other agents.

Based on this we consider the planning problem as a dialectical and collaborative goal directed reasoning about actions. Each agent can refine refute or repair the ongoing team plan. If the repair of a previously refuted plan succeeds, it becomes more robust but it can still be refuted later. If the repair of the refuted plan fails, agents leave this part of the reasoning and explore another possibility. Finally “bad” sub-plans are ruled out because there is no agent able to push the investigation process further. As in an argumentation with opponents and proponents, the current plan is considered as an acceptable solution when the proposal/counter-proposal cycles end and there are no more objections. Therefore, the planning process is considered as an iterative non monotonous process of conjectures - refutations – repairs.

B. Agents’ architecture and functioning

As shown in Fig.3 the planning sub-architecture is constitutes of two types of agent, Manager agents which stimulate the discovered abstract Web services and the Planner agent which have a twofold purpose: to monitor the status of all the interactions and to interact with other types of agent (Discover, Evaluator). This should not be understood as a centralized control mechanism that hampers the decentralized vision of MAS. Instead, our aim was to centralize the global view of the planning subsystem on one entity to facilitate communication task. The Planner agent is initialized with the semantic description of the user goal to be realized (requested service). While the Manager agent is initialize by the semantic description of the Web service it stimulates. Each agent has its own knowledge base and they interact with each other within a respect of the dialogue rules [29].

The Manager agent (Fig. 4) consists of two main modules [30]:
- The reasoning module and the dialogue administrator. While the Planner agent (Fig. 5) has an additional component; the proof board.

a. Proof board
The proof board represents the public part of the communication where the sub-conjectures proposed by the various Manager agents are stored.

b. Dialogue administrator
The dialogue administrator is the module which allows exchanging the propositions between the Planner agent and the Manager agents.
c. **Reasoning module**

- The Planner’s reasoning module leans on i) defining from the proof board the conjecture to be refined (which makes the minimum assumption for example) ii) and selecting the subset of the Manager agents with whom it will continue the process of refinement.
- The Manager’s reasoning module leans on refinement of the conjectures received from the Planner by adding constraints of orders and causal links [29].

The planning dialogue is initiated by the Planner agent who subjects to all concerned Manager agents a first conjecture. This conjecture constitutes the user goal to be realized. At the reception of this first conjecture, the Manager agents begin to refine it (reasoning module). When a Manager agent calculates a sub-conjecture, it submits it to the Planner agent.

At the reception of a refinement, the Planner agent updates its proof board. Then send back the new sub-conjecture to refine to the Manager agents that it holds for the next cycle of refinement.

C. **Introductive example**

In order to ease the understanding of our planning mechanism we present an introductive example.

Assume a scenario where Ali who lives in Montreal has to go to Fez for a conference. He decides to organize his travel through Internet by using three Web Services. Each service is represented by a Manager agent: an Airways agent offering a service of plan tickets reservation, ONCEF offering a service of train tickets reservation and a Bank agent (representing Ali’s bank) that handles the payment of the various reservations made by Ali. The problem that the user submits to the user interface can be summarized in the following way:

- Initial state: Ali is at Montreal
- Final state: Ali is at Fez

Let us imagine now the dialogue that agents could build in order to plan Ali trip.

Ali: « I’m at Montreal and I want to go to Fez. Can you help me? »

Planner Agent: « I can’t answer you immediately; I will invoke my Manager agents»

ONCEF: «Sorry, I don’t know how to get there»

Airways: « I can take the user from Montreal to Casablanca provided that he is capable to continue to fez by using another mean of transport and that he is able to pays 1000$ »

Bank Agent: « I can pay the amount of 1000$, Ali account is credible. »

ONCEF: « I can help to fulfill the first condition, I can take the user from Casablanca to Fez but he has to pay 15$»

Bank Agent: « Ok, I think that we keep the solution. I can pay the amount of 15$ too. »

So the solution plan is: « Take the plan from Montreal to Casablanca then the train from Casablanca to Fez. » Its construction reposes on a centralized planning of the Planner agent with the cooperation of other Manager agents.

V. **Conclusion and Future Work**

Finally it seems that joining together multi-agent planning and semantic Web Service technologies leads to a more powerful architecture for Web Services composition.

In this paper, we introduced a novel proposal to solve the problem of automated dynamic composition of Web Services based on multi-agent planning architecture and semantic specification of WS. The system employs efficient decoupling of functional and non-functional requirements, and leads to improve failure handling by implanting self-healing propriety.

In our approach we pointed out a need of a matchmaking algorithm, a planning model, a selection strategy and an autonomic execution. In this paper we focus at the planning mechanisms, as the main mechanisms in our Web service composition architecture.

Our future work includes the completion of the proof-of-concept prototype, but before we plan first to explore and investigate the methods that are able to improve the approach in terms of its performance and usability in several domains.

**REFERENCES**

[16] X. Tang, F. Tang, L. Bing, and D. Chen, Dynamic Web Service Composition Based on Service Integration and HTN Planning, Seventh

[18] IBM, An architectural blueprint for autonomic computing, April 2003


