# Multi-agent System Based Service Composition in the Internet of Things

Samir Berrani, Ali Yachir, Badis Djamaa, Mohamed Aissani



# Multi-agent system based service composition in the Internet of things

Samir Berrani, Ali Yachir, Badis Djamaa and Mohamed Aissani

Artificial Intelligence Laboratory, Military Polytechnic School (EMP),
PO BOX 17, Bordj-El-Bahri, 16111, Algiers, Algeria
samir.berrani@yahoo.fr, a_yachir@yahoo.fr, badis.djamaa@gmail.com, maissani@gmail.com

**Abstract.** Service composition is seen as the key issue to create innovative, efficient, flexible and dynamic applications on the Internet of things (IoT). Accordingly, we propose, in this paper, an approach for IoT service composition based on multi-agent system where several agents are engaged to satisfy the user request. This approach is designed using SysML and implemented using Netlogo platform. The use-cases scenarios and extensive tests show clearly the interest, the feasibility and the suitability of the multi-agent system for service composition.

**Keywords:** Internet of things, service composition, multi-agent system, semantic web.

## 1 Introduction

As a natural continuity of Ubiquitous Computing (UbiComp) and Ambient Intelligence (AmI), the Internet of Things (IoT) envisions a future Internet architecture integrating both physical and cyber worlds by combining sensing and actuating with digital services. IoT is the future all-IP ecosystem where smart objects (or IP networked things) connected to the Internet, exchange their data and capabilities as services, just like today's millions of Web services. The challenging question, however, is not only how to make these smart objects communicate over the Internet using open and standardized protocols, but also how their provided services can be exchanged and aggregated efficiently to create novel and dynamic IoT applications. This research problem is known as service composition.

Service composition is one of the core principles of the Service Oriented Computing (SOC) paradigm. It aims to reuse several existing component services (or atomic services) to create more complex services (or composite services) by joining them in creative ways [1]. The idea, when applied to IoT, promises to provide value-added services that none of IoT smart objects could provide individually. In fact, service composition allows the aggregation of smart object services to meet complex requirements from various application domains. It can be used to create innovative IoT applications in an efficient, flexible and dynamic manner. A robust service composition mechanism also makes it possible to support applications in a dynamic network environment, which is the key to foster the development of IoT applications [2].

In this paper, an IoT application is created, using IoT service composition, to control and/or observe specific targets defined by users. To do so, appropriate IoT devices hosting services are deployed on/in the defined targets. In addition, we propose a subdivision of

IoT services and client requests into three types, namely: actuating, processing and trigger. Accordingly, appropriate services involved in the composition process are selected using the user request target. To perform the service composition process, we propose an approach based on a multi-agent system where several agents (targets, devices, services, requests and composer) are engaged in collaboration to meet the same goal: satisfying the user (client) request.

The remainder of this paper is organized as follows. Section 2 gives a background and discuss some existing work in IoT service composition. Section 3 presents the proposed semantic model describing IoT services, messages and user requests. This is followed by an explanation of the proposed multi-agent system for IoT service composition in Section 4. Section 5 describes the proposed approach implementation and some use cases scenarios. Finally, section 6 gives conclusions and ideas for future work.

## 2 Related works

Service composition issues in classic and Pervasive Computing have been largely discussed in literature, and several solutions have been also proposed. A user-centric service composition approach [3] assists users in composing and selecting their applications by analyzing dependencies among published application templates, collaboration templates, workflows, and services. In [4], a Service Composition Planner (SCP) is proposed to generate all the feasible composite services from the user request and the set of available services. In [5], the problem of automatic service composition is addressed using a graph search-based algorithm and two different pre-processing techniques to handle the defined Same-Intension Different-Extension (SIDE) services. A Graph plan based approach is proposed in [6] for automatic service composition using branch structures in the resulting composite services to ensure their correct execution. In [7], the authors address the issue of selecting and composing web services via a genetic algorithm and give a transaction and QoS-aware selection approach. A service composition is proposed in [8] using two main phases: off-line phase and on-line phase. In the off-line phase, a global graph that links all the available abstract services is generated using a rule-based technique. In the on-line phase, a sub-graph is extracted spontaneously from the global graph according to the occurred and detected events in the environment.

Besides, the service composition approaches based on multi-agent systems are investigated regarding the similarity between services and agents. In [9], authors have tackled the dynamicity issues of ambient intelligence environment by using service-oriented architecture, semantic web services, and multi-agent systems. This approach empowers the dynamic service composition which allows the development of the AmI applications that improve autonomous reconfiguration to face the unstable and uncertain nature of AmI context and to maintain the user activities available. An automatic P2P semantic web service composition approach is presented in [10]. It is based on multi-agent systems to carry out web service composition. It uses the multi-agent System Engineering (MaSE) methodology to define the whole life cycle of the studied system including analyzing, designing and developing the expected multi-agent system. In [11], the authors propose a decentralized multi-agent model for service composition. This approach of service composition is based on local service interactions in order to form agent coalitions to create

new non-existing functionalities. The proposed decentralized multi-agent model of service composition is implemented using the Java Agent DEvelopment Framework (JADE) and the web services integration gateway. In [12], the proposed service composition approach is based on software agents. It aims to reduce the complexity of composition service global networks which are too complex to be dealt with. The proposed service compositor is based on agent coordination to calculate the composite service that fulfills the user request. This system elaborates an activity plan and controls its execution in order to verify if the user requirements are satisfied. The control strategy improves the time response of the user request and decreases the network traffic by compacting the overhead of exchanged messages.

In the previous works, IoT application based services are partially described without including its actors, such as targets, devices, services, and client requests. Moreover, relationships between such actors are not taken into account. In addition, a subdivision of services and requests into sub-categories to reduce the composition process complexity is not considered.

To overcome the aforementioned shortcomings of existing works, we propose a new model based multi-agent system for service composition in IoT. It represents the first developed part of our global solution for the applications based on IoT services. Our model takes into consideration services and client requests classification (actuating, processing and trigger) in order to enhance the composition process performances. We implement targets, devices, services and requests as agents. The request agent initiates the composition process through the composer agent which selects the appropriate service agents based on the target specified by request agent

## 3 Service and request description model

Most of IoT service systems consider that semantic or knowledge model is a basic foundation that provides indispensable means to define communication rules. The exchanged data are considered structured messages which can be requests or responses.

### 3.1 Context messages

In IoT service ecosystem, we suppose that a message is a basic entity used in communication between services. It represents an observed context state, treated data or a command at a specified time. We assume that the IoT message contains functionally dependent context parameters.

### 3.2 Service description

In this work, we adopt the service definition given in [8], in which concrete and abstract services are introduced.

**Concrete services** They implement a physical or logical function which provides context data or applies an effect on the environment. A concrete service has a real implementation that can be invoked through a specific input message in order to produce an output message.

**Abstract services** It is a class of concrete services that are functionally equivalent. Besides, they are based on input/output data and category similitude criteria. So, two concrete services are considered functionally equivalent, if and only if, they have the same input and output parameters, and reference an identical category.

### 3.3 User request description

In the system engineering discipline, any system is described through its parameters which represent pertinent means to observe the system behavior during its execution. We consider IoT services as parameters which allow us to study the evolution of the application targets. We suppose that the client does not need to know the system's architecture or the available services. Simply, he defines his request using the reference ontology. The reasoner must well understand the user's request through the semantic interpretation and calculates its relative response which can be a basic service or a composite one. We have distinguished three main types of requests as mentioned above. The trigger request concerns the commands to control actuators trough actuating services or to ask for raw context data in periodic, event-aware or on-demand manner. The processing request is dedicated to carry out context data in order to calculate another treated data or decisions (command). Finally, the actuating request is used to apply directly an effect on/in specific targets.

## 4 multi-agent system based service composition model

### 4.1 Adequacy of multi-agent system for service composition

multi-agent systems are constituted from a well-defined environment and various agents. The environment is formed by a set of passive and active objects that can be manipulated by agents. The latter are endowed with communication ability, some own resources and a satisfaction function which represents their individual tendencies or individual objectives. In addition, they can perceive the environment in a limited way, reproduce and die. Moreover, agent-agents and/or agent-environment interactions allow coordination and cooperation to achieve a global objective in a very close way to reality. For these reasons, the multi-agent approach is largely used to study phenomena and systems from several domains.

In addition, an agent is an autonomous entity that operates on an environment with an ability to communicate with other agents, receives and transmits messages, in order to improve coordination, cooperation and avoid deadlock and starvation when they are in need to use some critical resources. These mechanisms allow efficient collaboration to achieve global tasks. Comparing the IoT service systems, we can distinguish some similarities, namely: between agents and abstract services on the plans of structure, behavior, and communication way, also the collaboration to achieve a global goal by agents and the service composition process to satisfy the user's request. Based on these points, we believe that multi-agent systems constitute a faithful model for IoT service systems. In the following section, we introduce a description of a model based on the multi-agent system.

### 4.2 Functional description of the proposed multi-agent system

The proposed multi-agent system calculates a solution for a user's request. The relative responses can be simple or composite services. In this section, we define a reasoner model based on the multi-agent system. According to figure 1 in which is depicted the use case of the proposed reasoner system, we remark that this system calculates response for all kinds of requests aforementioned. The calculated responses are based on the list of available services provided by the service registry. Services involved in the composition process are selected according to the user's request subject.

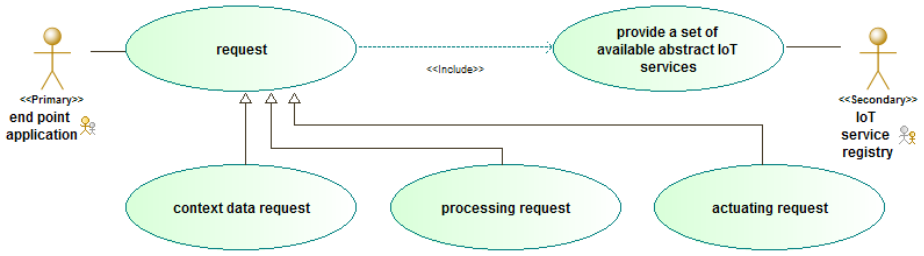


**Fig. 1.** multi-agent system use case diagram

The reasoner under construction has to provide a simple or a composite service that satisfies the user's requirements. The calculation of the user's request response must be in a finite time. The developed system must avoid the starvation and the deadlock of the user's requests.

### 4.3 Structural description of the proposed multi-agent system

As shown in diagram: A of figure 2, the proposed reasoner contains a request agent, service agents, and link agents. The reasoner agent plays the role of an observer agent which handles the entire agent world including environment and different types of existing agents.

The request agent represents the user's request. It is described by an output message and a category. The output message denotes the expected user result and the category is introduced to simplify the request complexity. We have distinguished three types of categories. Each one among these categories will be satisfied by a special composition agent. Diagram: B of figure 2 illustrates the user request types.

The service agents symbolize the services of IoT systems. The service agent is featured with an input message, an output message, a category, a number of contained concrete services (it is optional, this parameter is used in a case of abstract service) and

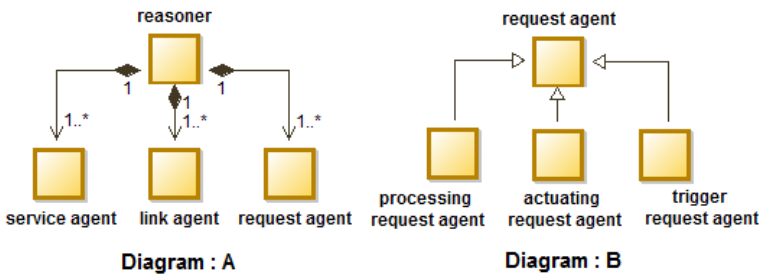


**Fig. 2.** Reasoner and request agent block definition diagram

an identifier. The service agent can be an actuating service agent that applies directly an effect on specific targets, a processing service agent that provides treated data or a command based on raw context data, or a trigger service agent which collects context data from observed targets or provides commands for actuating service agents. In diagram: A of figure 3, we illustrate the proposed service agent types.

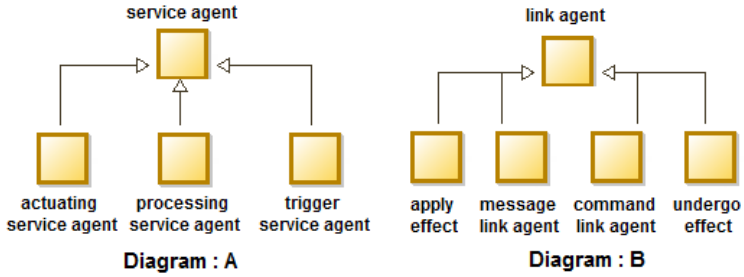


**Fig. 3.** Service and link agent block definition diagram

The link agents denote the possible links between all types of agents including service and request ones. Two agents are connected by a link agent, if and only if the output message type of one of them is similar to the input message type of the second one. On the other side, they can be seen as a communication channel which joins two service agents. In diagram: B of figure 3, we present the existing link agent types, namely a message and a command which are exchanged between service agents. However, apply an effect link agent is a link between a specific target and a service agent. In this case, we represent the physical influence of actuating service agents on a specific target. Furthermore, we have proposed an undergo link agent type to represent the influence of a specific target on trigger service agents.

### 4.4 Dynamic description of the proposed multi-agents system

The proposed reasoner based on the multi-agent system has mainly three behavior patterns. The composition process is launched by the user's request which is described by a subject, an expected result, and a category. According to this latter, the reasoner selects the appropriate services and the relevant composition process to calculate a composite agent that fulfills the user's request. As we have mentioned, a trigger request is answered by a trigger agent, a processing request is satisfied by a processing agent and an actuating request is responded by an actuating agent.

Figure 4 shows the dynamic behavior of the proposed reasoner based on the multi-agent system. Each state depicts which kind of agents is involved in the composition process. In addition to the request agent, three states can be distinguished, namely: trigger, processing and actuating. Each transition is featured by a condition that must be validated to cross it. Meeting such condition means also that the aggregation between their agents is allowed. So, we can deduce that transitions between states cited above depict link agents.

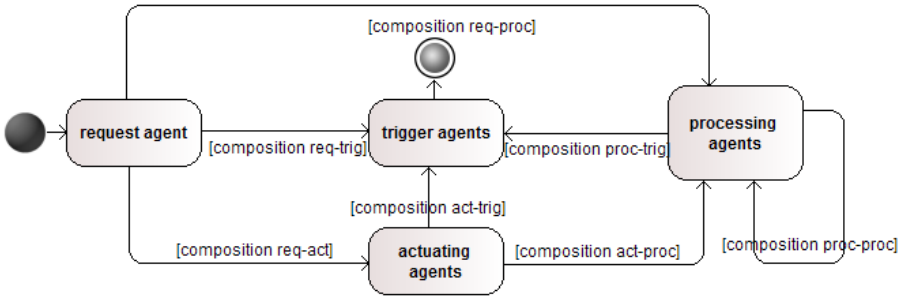


**Fig. 4.** Reasoner agent state machine diagram

In figure 5, we have described the reasoner process to calculate a composite agent that fulfills the user's request. First of all, the reasoner loads the user's request and according to its category, the reasoner creates the appropriate agents. Thereafter, the reasoner checks, if there are mappings between the request expected result and the output message of the other agents. These latter are selected according to the category of the request agent. If the result set is empty, it does mean, that no solution is found, otherwise, the process of composition continues. After each composition round, the reasoner checks, if there is a composite agent that satisfies the request agent. In other words, the reasoner verifies, if there is a connection between the trigger agents and the set of the composite agent under construction.

The processing-processing composition has two major problems. The first one concerns the direct composition loop. For example, let's suppose two agents "S1" and "S2" described respectively by "msg1" and "msg2" as inputs and "msg2" and "msg1" as

outputs. The category of both agents is "processing". We remark that it is possible to create an infinite composite using "S1" and "S2" in an alternative way such as "S1-S2-S1-S2-,...,ect".

On the other side, the second problem is related to the indirect composition loop. It does mean that we have a cycle of a sub-composite agent which can be repeated in an infinite way. To illustrate this case let's assume three agents "S1", "S2" and "S3" described respectively by "msg1", "msg2" and "msg3" as inputs and "msg2", "msg3" and "msg1" as outputs. We observe that the elementary composite is (S1-S2-S3). Therefore, we can notice the possibility to get as composition result an infinite composite (S1-S2-S3)-(S1-S2-S3)-(S1-S2-S3)-,...,ect., which are considered a negative composition.

To solve these problems, we have supposed that in a composite agent, an agent must be used at most once time. Taking into account this hypothesis, we have introduced a mechanism that detects whether the reasoner is developing a negative (infinite) composition or not. In processing-processing composition, the reasoner checks, if there are new composites after each composition cycle. If it is the case, the reasoner continues the development, otherwise, the composition process will be stopped in order to avoid negative construction.

## 5 Implementation and use cases scenarios

To implement the proposed reasoner model, we have used the Netlogo language. As known, the programming task is an interpretation and a translation of a model that includes functional, structural and behavioral aspects, in a specific language. In this work, we have implemented the trigger, processing and actuating agents as turtle agents. The link agents symbolize the communication channels between the system' agents. However, the reasoner agent is an observer agent, so it can control all the agent world. Our proposed system is based on coordination and this task is ensured by the observer agent perfectly.

Thanks to the state machine diagram 4 and the activity diagram 5 which describe the behavior of the multi-agent system, we have elaborated its implementation. We have specified for each agent of the system its own properties and methods. For the main one, the observer plays the role of the reasoner, which manages the coordination between all types of agents in order to look for an agent composite that satisfies the user's request agent.

### 5.1 Verification of the proposed multi-agent system : unit tests

During the development of the reasoner system, we have introduced unit tests to avoid code mistakes. Indeed, we have tested the composer program for each possible scenario, namely: trigger, processing and actuating requests.

In a case of trigger request, we have elaborated scenarios for negative and positive responses. In the former, it does mean no connections between request and triggers agents. However in the latter, it signifies that at least one of the trigger agents which satisfied the request agent was found.

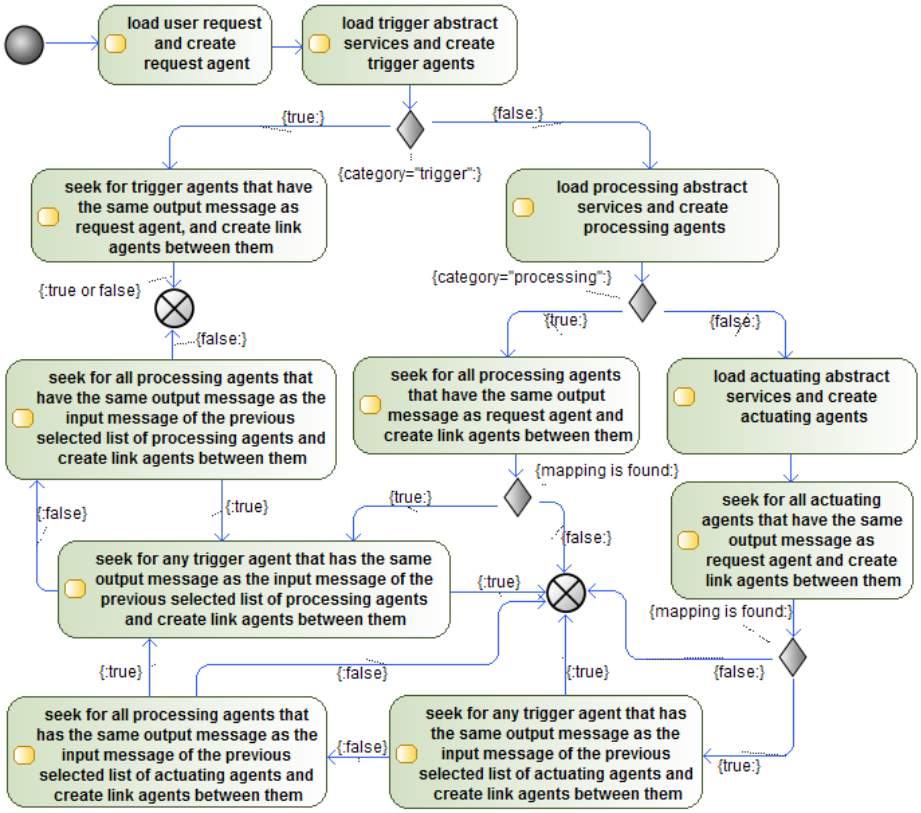


**Fig. 5.** Reasoner agent activity diagram

Concerning the second case, the processing request, we have prepared negative tests including non existing mapping between request and processing agents. Thereafter, we have tested the reasoner with non existing mapping between processing and trigger agents. The remaining tests concern the positive responses, in which the composite agent that satisfies the request was calculated.

The last verification case concerns the actuating request. In the first step, we have tested the developed reasoner in a non existing mapping between request and actuating agents, actuating and processing agents, processing and processing agents, and finally, processing and trigger agents. In addition, we have verified the reasoner in positive responses including the actuating-trigger and actuating-processing-trigger composition process. As a complementary test, we have tested the reasoner in case of processing-processing composition with negative calculation (a direct and indirect composition loop).

### 5.2 Description of the verification context

In this subsection, we illustrate our global vision and proposed solution related to the service composition in IoT. We suppose that any application based on IoT services contains targets. In our test case scenario, we aim to control and observe the temperature of a specific office. So, our target is an office, identified as "target-1".

To control and observe the "target-1", we deploy some appropriate IoT devices, namely: an air conditioner, a temperature sensor and a processing board which are identified by " AC-1", "Stemp" and "ProcB", respectively. The air conditioner "AC" provides a set of actuating services, turn on the air conditioner (m1,#,S1), turn off the air conditioner (m2,#,S2) and set preferred temperature of the air conditioner (m4,#,S3) in Fahrenheit units. The triples (m1,#,S1), (m2,#,S2) and (m3,#,S3) are actuating service descriptions, in which "S1", "S2" and "S3" are service identifiers, "m1", "m2" and "m3" are messages, # represents an effect that is applied in/on the "target-1".

In addition, a virtual controller provides a set of trigger services for controlling the actuating services. This software/hardware component supplies the following trigger services: turn on the air conditioner (#,m1,S4), turn off the air conditioner (#,m2,S5) and set preferred temperature of the air conditioner (#,m4,S6) in Fahrenheit units. These trigger services can be called directly by user requests. The triples (#,m1,S4), (#,m2,S5) and (#,m3,S6) are trigger service descriptions, in which "S4", "S5" and "S6" are service identifiers, # represents the "target-1" effects/orders,"m1", "m2" and "m3" are trigger messages.

To complete this example, we suppose that "Stemp" publishes a trigger service (#,m4,S7) that provides the ambient temperature of the office "target-1" in Fahrenheit units. The triple (#,m4,S7) is a trigger service description, in which # represent an effect applied by observed context, target-1, on sensor device, "m4" is a trigger message that represents measurements of the observed phenomena and "S7" is the service identifier.

Finally, the processing board "ProcB" provides a processing service that converts temperature values form Fahrenheit to Celsius (m4,m3,S8).

### 5.3 Request examples and obtained results

In this section, we introduce some elementary test cases in order to illustrate the process patterns and the execution of the developed reasoner. To do it, we have prepared three types of test case scenarios, namely : trigger, processing and actuating requests.

*Trigger request :* We assume the following trigger request : "Give us the ambient temperature of the office in Fahrenheit units". The translation of this request is as follows : ("target-1",m4,req1). The "target-1" represents the observed target (subject/office), the message "m4" depicts the temperature measurements in Fahrenheit unit and "req1" is the identifier of the request. Figure 6, trigger case, represents the reasoner's response to the request defined above. We remark that "req1" is satisfied by trigger service (S7) which provides the current temperature of the office in Fahrenheit units.

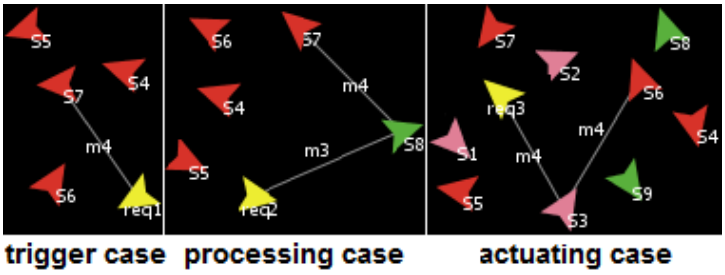


**Fig. 6.** The responses of reasoner agent to the trigger, processing and actuating requests

*Processing request :* We suppose the following processing request : "Give us the ambient temperature of the office in Celsius units". The translation of this request is as follows : ("target-1",m3,req2). The "target-1" represents the observed target (subject/office), the message "m3" depicts the temperature measurements in Celsius unit and "req2" is the identifier of the request. Figure 6, processing case, represents the reasoner's response to the request defined above. We remark that "req2" is satisfied by the processing service (S8) which converts to Celsius units the current temperature of the office provided by trigger service "S7" given in Fahrenheit units.

*Actuating request :* We suppose the following actuating request : "Set the preferred temperature of the air conditioner to 77 Fahrenheit". The translation of this request is as follows : ("target-1",m4,req3). The agent "target-1" represents the target (subject/office), the message "m4" depicts the temperature value in Fahrenheit units and "req3" is the identifier of the request. Figure 6, actuating case, represents the reasoner's response to the request defined above. We remark that "req3" is satisfied by a composite agent "S3"+"S6". The agent "S6" represents a trigger service that provides the command to set temperature of the air conditioner in Fahrenheit unit. The agent "S3" is an actuating service that applied the received order sent by the agent "S6".

## 6 Conclusion

In this work, we have presented a new multi-agent system based service composition approach. Such an approach is based on four main agents namely: reasoner (observer) agent, service agent, request agent and link agent. These agents are engaged in coordination to achieve the same goal of satisfying the user's request. We have described the role of each agent and their collaborative process. The use-cases scenarios and extensive tests show clearly the interest, feasibility and suitability of the multi-agent system for service composition. As future work, it is still necessary to conduct further empirical evaluations to study the impact of the different pre-configured parameters (number of concrete services, number of requests, etc.) on the performance of the proposed approach. We are also currently working on designing and implementing an endpoint application and an automatic concrete services classification approach.

## References


1. Lemos, A.L., Daniel, F., Benatallah, B.: Web service composition: A survey of techniques and tools. ACM Comput. Surv. **48**(3) (December 2015) 33:1–33:41
2. Chen, I., Guo, J., Bao, F.: Trust management for soa-based iot and its application to service composition. IEEE Transactions on Services Computing **9** (2016) 482–495
3. Tsai, W.T., Zhong, P., Bai, X., Elston, J.: Dependence-guided service composition for user-centric soa. IEEE Systems **8**(3) (September 2014) 889–99
4. Wang, P.W., Ding, Z.J., Jiang, C.J., Zhou, M.C.: Automated web service composition supporting conditional branch structures. Enterprise Inform. Syst **8**(1) (01 2014) 121–146
5. Wang, P.W., Ding, Z.J., Jiang, C.J., Zhou, M.C.: Constraint-aware approach to web service composition. IEEE Trans. Syst., Man Cybern.: Syst **44**(6) (06 2014) 770–784
6. Wang, P.W., Ding, Z.J., Jiang, C.J., Zhou, M.C., Zheng, Y.W.: Automatic web service composition based on uncertainty execution effects. IEEE Trans. Services Comput. (2015)
7. Ding, Z.J., Liu, J.J., Sun, Y.Q., Jiang, C.J., Zhou, M.C.: A transaction and qos-aware service selection approach based on genetic algorithm. IEEE Trans. Syst., Man, Cybern.: Syst. **45**(7) (07 2015) 1035–1046
8. Yachir, A., Amirat, Y., Chibani, A., Badache, N.: Towards an event-aware approach for ubiquitous computing based on automatic service composition and selection. Ann. Telecommun. **67**(7) (06 2012) 341–353
9. Vallée, M., Ramparany, F., Vercouter, L.: A multi-agent system for dynamic service composition in ambient intelligence environments. In: The 3rd International Conference on Pervasive Computing (PERVASIVE 2005). (2005)
10. Paikari, E., Livani, E., Moshirpour, M., Far, B.H., Ruhe, G.: Multi-agent system for semantic web service composition. In: Knowledge Science, Engineering and Management: 5th International Conference, Irvine, CA, USA, December 12-14. (2011)
11. Papadopoulos, P., Tianfield, H., Moffat, D., Barrie, P.: Decentralized multi-agent service composition. Multiagent Grid Syst. **9**(1) (1 2013) 45–100
12. Bennajeh, A., Hachicha, H.: Web service composition based on a multi-agent system. Software Engineering in Intelligent Systems: Proceedings of the 4th Computer Science Online Conference 2015 (CSOC2015), Vol 3: Software Engineering in Intelligent Systems (2015)