A Web Services Composition Approach based on Software Agents and Context

Zakaria Maamar
Zayed University, U.A.E
zakaria.maamar@zu.ac.ae

Soraya Kouadri M.
Fribourg University, Switzerland
kouadris@acm.org

Hamdi Yahyaoui
Laval University, Canada
hamdi.yahyaoui@ift.ulaval.ca

ABSTRACT

We present an agent-based and context-oriented approach for Web services composition. A Web service is an accessible application that other applications and humans can discover and trigger to satisfy various needs. Due to the complexity of Web services composition, we consider two concepts to reduce this complexity: software agent and context. A software agent is an autonomous entity that acts on behalf of users, whereas context is any information relevant to characterize a situation. During composition, software agents engage conversations with their peers to agree on the Web services that will participate in the composition.

Keywords

Web service, software agent, composition, context, security.

1. INTRODUCTION

The increasing demand of users for high quality and timely information is putting businesses under the pressure of adjusting their know-how and seeking as well for more support from other businesses due to various arguments such as cost-effectivity and expertise-availability. A strategy that implements such a support is to merge business processes despite well-known obstacles (e.g., lack of a common ontology). In this paper, we illustrate a business process with a Web service. A Web service is an accessible application that other applications and humans can automatically discover and invoke [3]. In general, composing multiple Web services (also called services in the rest of this paper) rather than accessing a single service is essential and provides more benefits to users. Discovering the component services, integrating the services into a *composite service*, triggering the composite service for execution, and last but not least monitoring this execution for the needs of exception handling are among the operations that users will have to take care. Most of these operations are complex, although repetitive with a large segment suitable to computer aid and automation. Therefore, *Software Agents* (SAs) are deemed appropriate to assist users in their operations [5].

Entrusting the composition of Web services to software agents is not obvious. Indeed, different issues are raised including which businesses have the capacity to provision Web services, when and where the provisioning of Web services occurs, how Web services from separate businesses coordinate their activities so conflicts can be avoided. To address certain of these issues, agents need to be aware of the *context* [4] in which the composition and execution of the Web services will occur. For instance, before provisioning a service for execution the computing capabilities of the resources *vs.* the computing requirements of the service is assessed. In this paper, we present our *agent-based and context-oriented approach for Web services composition*.

In this approach, we leverage the interactions between agents during the composition of services to the level of *conversations*. A conversation is a consistent exchange of messages between participants involved in joint operations and consequently, have common interests. Ardisson et al. observe in [2] that current Web services standards support simple interactions and are mostly structured as question-answer pairs. These limitations hinder the possibility of expressing complex situations that require more that two turns of interactions. To address these limitations, we illustrate in this paper how agents engage conversations with their peers when it comes for example to search for the component services, to check the availability of these services, and to trigger these services for execution.

Section 2 overviews various concepts such as Web services and conversations. Section 3 presents the agentification of Web services composition. Section 4 talks about the security of the computing resources on which the Web services are executed. Finally, Section 5 draws our conclusions. It should be noted that the mechanisms (e.g., UDDI registries) for discovering the component Web services, while important, do not fall within this paper's scope.

2. PRELIMINARIES

Web services. A Web service is an accessible application that other applications and humans as well can discover and invoke. Benatallah et al. suggest the following properties for a Web service [3]: (i) independent as much as possible from specific platforms and computing paradigms; (ii) developed mainly for inter-organizational situations; and (iii) easily composable (i.e., no need for complex adapters).

For our projects on Web services, we developed *Service*

SAC'04, March 14-17, 2004, Nicosia, Cyprus.

Chart Diagrams (SCDs) as a specification means of the composition of the Web services [7]. A SCD leverages an UML state chart diagram, putting the focus on the execution context of a service rather than only on the states of the service. A SCD wraps the states of a Web service into four perspectives: *flow*, *business*, *information*, and *performance*.

Software agents. A SA is a piece of software that autonomously acts to carry out tasks on users' behalf [5]. A SA exhibits several features that make it different from other traditional components: autonomy, goal-orientation, collaboration, flexibility, self-starting, temporal continuity, character, communication, adaptation, and mobility.

Context. Composed of *con* (with) and *text*, context is the meaning that can be inferred from an adjacent text. Dey considers in [4] context as any information that is relevant to the interactions between the user and the environment. This information can be related to the circumstances, objects, or conditions by which a user is surrounded.

Conversations. Smith et al. define in [10] conversations as a sequence of messages involving participants who intend to achieve a particular purpose. In general, a conversation either succeeds or fails. On one side, a conversation succeeds because what was expected from the conversation in term of outcome has been achieved. On the other side, a conversation fails because the conversation faced several technical difficulties or didn't achieve what was expected.

3. AGENTIFICATION OF WEB SERVICES

Three types of input sources can contribute to the development of a context: service, user, or both user and service. In [8], it is noted that a user-centric context promotes applications that move with users, adapt according to changes in the available resources, and provide configuration mechanisms based on users' personal-preferences. We advocate that a service-centric context promotes applications that allow service adaptability, deal with service availability, and support an on-the-fly service composition. In this paper, the focus is on the context of services.

3.1 Agent-based deployment

The rationale of the agentification of Web services composition is to determine the appropriate types and roles of agents that will deploy this composition. Currently, we put forward three types of agents: *composite-service-agent*, *master-service-agent*, and *service-agent*.

We consider a Web service as a component that is instantiated each time it is being called to participate in a composition. Before the instantiation happens, several elements related to the Web service have to be checked. These elements constitute a part of the context, denoted by $\mathcal{W}$-context, of the Web service and are as follows: (i) the number of service instances currently running *vs.* the maximum number of service instances that simultaneously can be run,(ii) the execution status and location of each service instance deployed, and (iii) the time of request of the service instance *vs.* the time of availability of the next service instance.

The role of the master-service-agent is to track the Web service instances that are obtained from a Web service. Web services, Master-service-agents, and $\mathcal{W}$-contexts are all stored in a pool (Fig. 1). A master-service-agent processes the requests of instantiation that are submitted to a Web service. These requests originate from composite-service-agents that identify the composite services to set-up. For instance, the master-service-agent makes decisions on whether a Web service is authorized to join a composite service. In case of approval, a service instance and a context, denoted by $\mathcal{I}$-context, are created. An authorization can be rejected for different reasons: period of non-availability, overloaded status, or exception situation.

To be informed about the running instances of a Web service so the $\mathcal{W}$-context can be updated, the master-service-agent associates each instance created with two components: a service-agent and an $\mathcal{I}$-context. The service-agent manages the service chart diagram and the $\mathcal{I}$-context of the service instance. For example, the service-agent knows the states that the service instance should take, and the Web services that need to join the composite service after the execution of this service instance is completed.

Master-service-agents and service-agents are in constant interactions. Indeed, the content of $\mathcal{I}$-contexts feed the content of $\mathcal{W}$-contexts with various details: (i) what is the execution status of a service instance? (ii) When is the execution of a service instance supposed to resume in case it has been suspended? And, (iii) when is the execution of a service instance expected to complete?

With regard to composite-service-agents, their role is to trigger the specification of the composite services and monitor their deployment. A composite-service-agent ensures that the appropriate component services are involved and collaborating according to a specific specification. When a composite-service-agent downloads the specification of a composite service, it (i) establishes a context denoted by $\mathcal{C}$-context for this composite service, and (ii) identifies the first Web services to be triggered. For the sake of simplicity, we assume that the Web services constitute a sequence. However, our description is service chronology-independent. When the first Web service is known, the composite-service-agent interacts with the master-service-agent of this Web service in the objective to ask for a service instantiation. If the master-service-agent agrees on this instantiation after checking the $\mathcal{W}$-context (Section 3.3), a service-agent and an $\mathcal{I}$-context are set up. Afterwards, the service chart diagram of the new service-instance is transmitted to the service-agent. The service-agent initiates the execution of the service instance and notifies the master-service-agent about the execution status. Because of the regular notifications between service-agents and master-service-agents, exceptional situations are immediately handled so corrective actions are timely carried-out on time. In addition, while the Web service instance is being performed, the service-agent identifies the Web services that are due for execution after this service instance. In case there are Web services due for execution, the service-agent requests from the composite-service-agent to engage conversations with their respective master-service-agent.

Fig. 1 represents an execution session of composite service CS_1 that has 4 primitive component services: $\text{service}_{\{1,2,3,4\}}$. Each service instance has a service chart diagram. The clouds in the same figure correspond to contexts. $\mathcal{I}$-context is the core context that the service-agent uses for updating the $\mathcal{C}$-context and $\mathcal{W}$-context of the respective composite-service-agent and master-service-agent. The exchange of in-

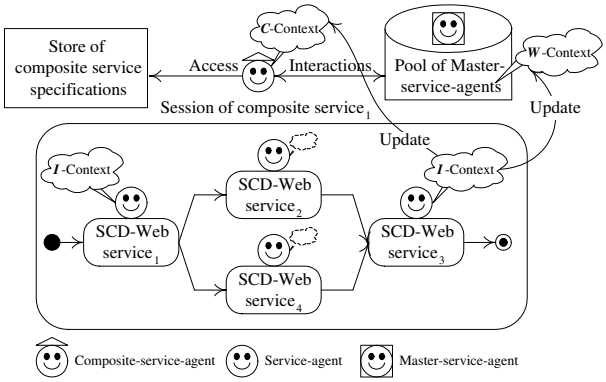

Figure 1: Agents for Web services composition

formation that occurs between a master-service-agent and a service-agent has already been discussed in the previous paragraphs. In addition to a copy (or a part based on the level of detail that needs to be tracked) of that exchange that is sent to composite-service-agents, these ones receive extra details from service-agents: (i) the next services to be called for execution, and (ii) the type of these services whether mandatory or optional.

3.2 Details on $\mathcal{I}/\mathcal{W}/\mathcal{C}$-contexts

Besides the three types of agents that are identified during the agentification of Web services composition (Fig. 1), three types of services are considered: composite service, Web service, and Web service instance. Each service is attached to a specific context. The $\mathcal{I}$-context is the most-grained one, whereas the $\mathcal{C}$-context is the least-grained one. The $\mathcal{W}$-context is in between the two contexts. Details on an $\mathcal{I}$-context are used for updating a $\mathcal{W}$-context, whereas details on a $\mathcal{W}$-context are used for updating a $\mathcal{C}$-context. We use *Tuple Spaces* to implement the update operations between contexts [1]. However, because of lack of space these operations are not discussed[1], and only the structure of $\mathcal{I}$-context is presented.

The $\mathcal{I}$-context of a Web service instance consists of the parameters label, service-agent label, status, previous service instances, next service instances, regular actions, begin time, end time (expected and effective), reasons of failure/suspension, corrective actions, and date.

We pointed out in the beginning of Section 3 that a service-centric context is adopted. This has been shown with the $\mathcal{W}$-context of a Web service that constitutes the link between the $\mathcal{I}$-contexts and $\mathcal{C}$-contexts. A service-centric context promotes service adaptability, availability, and on-the-fly composition. We meet these requirements in our agentification approach of Web services composition. A composite service may have to adapt its list of component Web services due to the availability of certain of these components. Availability is illustrated with two cases: a service is either mandatory or optional, and the maximum number of services instances that can be created compared to the current number of service instances that are running. Since Web services are instantiated on a request-basis, this means that an on-the-fly composition is supported.

In our work, we see a $\mathcal{W}$-context of a Web service along three perspectives: *instance*, *execution*, and *time*. The rationale of each perspective is as follows.

1. Instance perspective: deals with creating service instances, assigning them to composite services, and getting ready the next service instances.
2. Execution perspective: deals with meeting the computing resource requirements of service instances, tracking their execution, and avoiding conflicts on these computing resources.
3. Time perspective: deals with time-related parameters of and constraints on service instances.

3.3 Conversations between agents

In the following and for the sake of simplicity, a component service always refers to a Web service. In a reactive composition such as the one that features our agentification approach, the selection of the component services of a composite service is done on-the-fly. We outsource the selection operations to composite-service-agents that engage *conversations* with the respective master-service-agent of the appropriate Web services. In these conversations, master-service-agents decide if their Web service will join the composition process after checking the $\mathcal{C}$-contexts. In case of a positive decision, Web service instances, service-agents, and $\mathcal{I}$-contexts are deployed.

When a Web service instance is being executed, its service-agent checks its service chart diagram. The purpose is to verify if additional Web services have to be executed. In case of a positive verification, the service-agent requests from the composite-service-agent to engage conversations with the master-service-agents of these Web services. These conversations have two aims: (i) invite master-service-agents and thus, their Web service to participate in the composition process; and (ii) ensure that the Web services are got ready for instantiation in case of invitation acceptance.

Fig. 2 depicts a conversation diagram between a service-agent, a composite-service-agent, and a master-service-agent. The composite-service-agent is in charge of a composite service that has n component Web services$_{(1,\ \cdots,\ i,\ j,\ \cdots,\ n)}$. In this figure, rounded rectangles are states (states with underlined labels belong to Web service instances, whereas other states belong to agents), italic sentences are conversations, and numbers are the chronology of conversations. Initially, Web service instance$_i$ takes an execution state. Furthermore, service-agent$_i$ and the composite-service-agent take each a conversation state. In these conversation states, activities to request the participation of the next Web services (i.e., Web service$_j$) are performed.

Upon receiving a request from service-agent$_i$ about the need of involving Web service$_j$ (0), the composite-service-agent engages conversations with master-service-agent$_j$ (1). This service is an element of the composite service under preparation. A composite service is decomposed into three parts. The first part is the Web service instances that have completed their execution (Web services$_{1,\ \cdots,\ i-1}$). The second part is the Web service instance that is now being executed (Web service instance$_i$). And, the third part is the

[1]Example: modified($\mathcal{I}$-context i, WebServiceInstance wsi)[true]—update($\mathcal{W}$context w, WebService ws) means that if the $\mathcal{I}$-context i of the Web service instance wsi has been modified, therefore the respective $\mathcal{W}$-context w of the web service ws needs also to be updated after collecting information from this $\mathcal{I}$-context i.

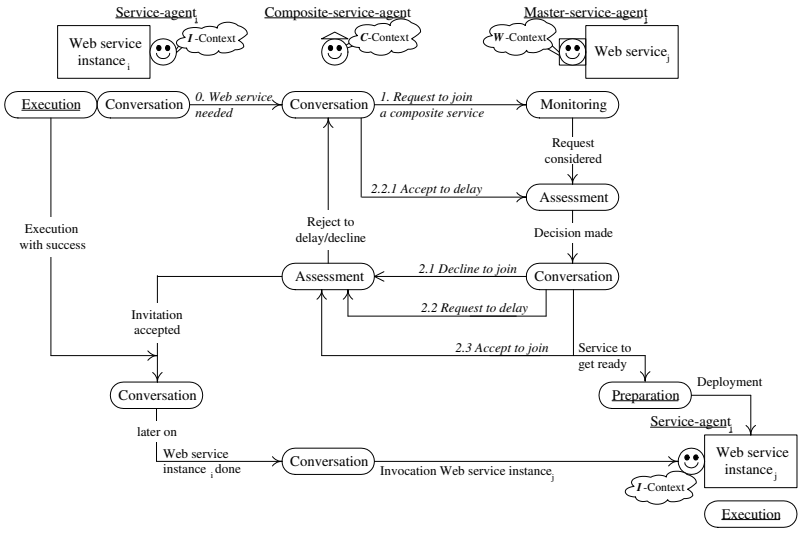

Figure 2: Conversation diagram between agents

rest of the component services that have to get ready for execution (Web services$_{j,\cdots,n}$). Initially, master-service-agent$_j$ is in a monitoring mode in which it tracks the instances of Web service$_j$ that are currently participating in different composite services. When it receives a request to create an additional instance, master-service-agent$_j$ enters the assessment state. Based on the $\mathcal{W}$-context of Web service$_j$, master-service-agent$_j$ evaluates the request of the composite-service-agent and makes a decision on one of the following options: (a) decline the request, (b) delay its making decision, or (c) accept the request. Due to lack of space, only (a) and (c) options are presented.

Option a. Master-service-agent$_j$ of Web service$_j$ declines the request of the composite-service-agent. A conversation message is sent from master-service-agent$_j$ to the composite-service-agent for information (2.1). Because a component service can be either mandatory or optional in a composite service, the composite-service-agent has to decide whether it has to pursue with master-service-agent$_j$. To this end, the composite-service-agent relies on the specification of Web service$_i$ and the $\mathcal{C}$-context of the composite service. Two exclusive cases are offered to the composite-service-agent:

- If Web service$_j$ is optional, the composite-service-agent enters again the conversation state, asking the master-service-agent of another Web service$_{k,\,(k \neq j)}$ to join the composite service (1).

- If Web service$_j$ is mandatory, the composite-service-agent engages further conversations with master-service-agent$_j$ asking for example for the reasons of rejection or the availability of the next instance of Web service$_j$.

Option c. Master-service-agent$_j$ of Web service$_j$ accepts to join the composite service. Consequently, it informs its acceptance to the composite-service-agent (2.3). This is followed by a Web Service Level Agreement (WSLA) between the two agents [6]. At the same time, master-service-agent$_j$ ensures that Web service$_j$ is getting ready for execution through the preparation state.

When the execution of Web service instance$_i$ is completed, service-agent$_i$ informs the composite-service-agent about that. According to the agreement of **Option c**, the composite-service-agent interacts with service-agent$_j$ so the newly created instance of Web service$_j$ is triggered. Therefore, Web service instance$_j$ enters the execution state. At the same time, the composite-service-agent initiates conversations with the master-service-agents of the next Web services that follow Web service$_j$.

4. SECURITY OF WEB SERVICES

Because Web services require the computing *resources* of hosts on which they are executed, we ensure that neither the services *misuse* these resources nor the hosts *alter* the integrity of the services. We highlight our security strategy that prevents services from misusing the resources of hosts.

4.1 Service-based access control

The Role-Based Access Control (RBAC) is a well-know approach for managing access rights of users in businesses [9]. In a business, it is common that users play roles based on role's requirements *vs.* user's capabilities. In an open service-oriented environment, the RBAC strategy is inappropriate. Because new services may be offered and existing services may be changed or withdrawn, a continuous adaptation of the access privileges that are assigned to roles and thus, to users is deemed mandatory. Managing roles and their access privileges becomes a real burden. The execution of the Web services constitutes a serious threat to computing hosts. A Web service can use the local services of a host (e.g., calendar) to request some sensitive information.

To handle the aforementioned security situations, we are in the process of designing a *service-based access control architecture*. Fig. 3 represents this architecture where numbers correspond to the steps that grant a Web service the right to use the resources of a host. The rationale of monitoring module and security context is explained below. We note

that a security context is totally different from the contexts of Section 3.2.

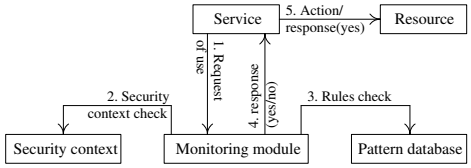

Figure 3: Service-based access control architecture

First of all, a service submits a request of use to the monitoring module (1). Once received, the monitoring module checks the security context of the resource that this service is about to use (2). Details on the security context are in Section 4.2. Next, the monitoring module browses the pattern database (3). The objective is to identify any *malicious pattern* that might exist in the database and present similarities with the execution trace of the service requesting the use of the resources. Details on the pattern database are in Section 4.3. A response to grant or deny the request of use is sent back to the service (4). Finally, the request is implemented (5) in case of a positive response from the monitoring module.

4.2 Security context specification

In Fig. 3, the monitoring module decides if a service has the right to use a resource. To this end, the monitoring module relies on a security context that has the following format: $(S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_n) \bullet R$ where S_i is a service, R is a resource, $\rightarrow$ is a calling operation between services, and $\bullet$ is a use request of a resource. Each resource R has a security context that needs to be checked each time a request of use of that resource is initiated. The final decision to grant/deny the use of a resource R is based on the algorithm of Fig. 4.

`Input:`	$S_1, S_2, \cdots, S_n, R$
`Get`	`Security Context` (SC) `of` R
`If`	`any` S_i `in` SC `not have the right to use` R
`Then`	`Deny request to use` R
`Else`	`Call check_rules()`

Figure 4: Algorithm for a resource request of use

`Check_rules()` is a function that uses a database of malicious patterns. It guarantees that the actions to be performed before granting the right of using a resource R do not constitute a malicious attack. Otherwise, the request of use is denied.

4.3 Pattern database

The pattern database stores all the *potential* threats on the resources of an environment of Web services. A pattern identifies each threat. A pattern is based on the execution trace of a service with regard to the sequence of primitive actions (i.e., no services called) that are implemented. Relying on the database, the monitoring module controls both the security context of any use request of a resource and the primitive actions that a service will perform. For example, let us assume a and b as primitive actions. When a service plans to perform action a then action b, the monitoring module checks if $a \cdot b$ does not constitute a malicious pattern ($\cdot$ represents a sequence of actions). To this end, the monitoring module consults the pattern database.

5. CONCLUSION

In this paper, we presented our approach for composing Web services using software agent and context. Several types of agents are suggested: composite-service-agents associated with composite services, master-service-agents associated with Web services, and service-agents associated with Web service instances. The different agents have been aware of the context of their respective services in the objective to devise composite services on-the-fly. Three types of context are used: $\mathcal{I}$-context, $\mathcal{W}$-context, and $\mathcal{C}$-context. Conversations between agents have also featured the composition of Web services. Before Web service instances are created, agents engage conversations to decide if service instances are created and annexed to a composite service. Such a decision is based on several factors such as the maximum number of service instances that can deployed at the same time.

6. REFERENCES

[1] S. Ahuja, N. Carriero, and D. Gelernter. Linda and Friends. *Computer*, 19(8), August 1986.

[2] L. Ardissono, A. Goy, and G. Petrone. Enabling Conversations with Web Services. In *Proc. of AAMAS'2003*, Melbourne, Australia, 2003.

[3] B. Benatallah, Q. Z. Sheng, and M. Dumas. The Self-Serv Environment for Web Services Composition. *IEEE Internet Computing*, 7(1), January/February 2003.

[4] A. K. Dey, G. D. Abowd, and D. Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction Journal, Special Issue on Context-Aware Computing*, 16(1), 2001.

[5] N. Jennings, K. Sycara, and M. Wooldridge. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers*, 1(1), 1998.

[6] H. Ludwig, A. Keller, A. Dah, and R. King. A Service Level Agreement Language for Dynamic Electronic Services. In *Proc. of WECWIS'2002*, Newport Beach, California, USA, 2002.

[7] Z. Maamar, B. Benatallah, and W. Mansoor. Service Chart Diagrams - Description & Application. In *Proc. of WWW'2003*, Budapest, Hungary, 2003.

[8] M. Roman and R. H. Campbell. A User-Centric, Resource-Aware, Context-Sensitive, Multi-Device Application Framework for Ubiquitous Computing Environments. Technical report, UIUCDCS-R-2002-2282 UILU-ENG-2002-1728, Departement of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA, 2002.

[9] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based Access Control Models. *IEEE Computer*, 20(2), February 1996.

[10] I. A. Smith, P. R. Cohen, J. M. Bradshaw, M. Greaves, and H. Holmback. Designing Conversation Policies using Joint Intention Theory. In *Proc. ICMAS'1998*, Paris, France, 1998.