Toward an Agent-Based and Context-Oriented Approach for Web Services Composition

Zakaria Maamar, Soraya Kouadri Mostéfaoui, and Hamdi Yahyaoui

Abstract—This paper presents an agent-based and context-oriented approach that supports the composition of Web services. A Web service is an accessible application that other applications and humans can discover and invoke to satisfy multiple needs. To reduce the complexity featuring the composition of Web services, two concepts are put forward, namely, software agent and context. A software agent is an autonomous entity that acts on behalf of users and the context is any relevant information that characterizes a situation. During the composition process, software agents engage in conversations with their peers to agree on the Web services that participate in this process. Conversations between agents take into account the execution context of the Web services. The security of the computing resources on which the Web services are executed constitutes another core component of the agent-based and context-oriented approach presented in this paper.

Index Terms—Web service, composition, software agent, context, conversation, security.

◆

1 INTRODUCTION

Web services are nowadays emerging as a major technology for deploying automated interactions between distributed and heterogeneous applications [7], [16]. Various standards back this deployment, including WSDL, UDDI, and SOAP [13]. These standards respectively support the definition of Web services, their advertisement to the community of potential users, and, finally, their binding for invocation purposes. Aissi et al. argue that Web services are meant to be used as bridges between applications that otherwise would have required extensive integration and development efforts [2].

The increasing demand of users for high quality and timely information is putting businesses under the pressure of adjusting their know-how and collaborating with other peers for various reasons, e.g., cost-effectiveness and expertise-availability. A strategy that implements this collaboration is to merge business processes despite well-known obstacles (e.g., lack of a common ontology [32]). 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 [7]. 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. Composition primarily addresses the situation of a user's request that cannot be satisfied by any available service, whereas a composite service obtained by combining available services might be used [8]. Discovering the component services, adding the component services to a composite service, triggering the composite service for execution, and, last but not least, monitoring the execution in case of exception handling are among the operations that users will have to be responsible for. Because of the complexity of most of these operations, software agents are deemed appropriate candidates to assist users. A software agent is an autonomous entity that acts on behalf of user, makes decisions, collaborates with its peers, and migrates to distant hosts if needed [18].

Entrusting the composition of Web services to software agents is not straightforward. Different questions arise, 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 that conflicts are avoided, and, last but not least, what back-up strategies handle execution exceptions of Web services. To address a part of these issues, agents need to be aware of the *context* [11], [14] in which the composition and execution of the Web services are expected to happen. Context is the information that characterizes the interaction between humans, applications, and the surrounding environment [11]. For instance, before provisioning a Web service for execution, the current computing capabilities of the resources versus the computing requirements of the Web service needs to be assessed. In addition, before deploying any back-up strategy, an assessment of the type of exception is required.

To make Web services composition more efficient, the interactions between agents of Web services are leveraged to the level of conversations. A conversation is a consistent exchange of messages between participants involved in joint operations and, consequently, have common interests. Ardissono et al. argue that current Web services standards support simple interactions and are mostly structured as question-answer pairs [3]. This lack of standardization hinders the possibility of expressing complex situations that call for more than two turns of interaction. The same comment is made by Benatallah et al. in [5], who noticed

---

- *Z. Maamar is with the College of Information Systems, Zayed University, PO Box 19282, Dubai, United Arab Emirates. E-mail: zakaria.maamar@zu.ac.ae.*
- *S.K. Mostéfaoui is with the Computer Science Department, University of Fribourg, Switzerland. E-mail: soraya.kouadrimostefaoui@unifr.ch.*
- *Y. Hamdi is with the Computer Science Department, Laval University, Canada. E-mail: hamdi.yahyaoui@ift.ulaval.ca.*

Manuscript received 18 Aug. 2003; revised 1 June 2004; accepted 20 Sept. 2004; published online 17 Mar. 2005.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0152-0803.

that despite the growing interest in Web services, several issues remain to be addressed to provide Web services with benefits similar to traditional integration middleware. One of Benatallah et al.'s suggestions to enhance Web services is the development of a conversational metamodel. In this paper, it will be shown how software agents, acting on behalf of services, could engage in conversations with their peers to, for example, search for the component services, check their availabilities, and trigger them once they agree on participating in a composition.

Web services composition is a very active area of research and development [31]. However, *very little* has been accomplished to date regarding the integration of conversations and context into agent-based composition approaches of Web services. In particular, several obstacles still hinder this integration; for instance, 1) web services act as passive components rather than active components that can be embedded with context-awareness mechanisms, 2) existing approaches for Web services composition (e.g., WSFL, BPEL) typically facilitate choreography only, while neglecting information about the context of users and services, and 3) lack of appropriate techniques for modeling and specifying conversations between Web services. This paper presents our *agent-based and context-oriented approach for Web services composition*. Section 2 overviews the concepts of software agent, context, and conversation. Section 3 defines Web services and the process of composing them. Section 4 presents the approach that agentifies the composition process of Web services. In addition, the value added of contexts and conversations to this approach is discussed in this section. Section 5 talks about the security of the computing resources on which the Web services are to be executed. Section 6 discusses the implementation of the approach proposed for agentifying and contextualizing Web services composition. Section 7 overviews related work. Finally, Section 8 draws conclusions and highlights future work. It should be noted at that level that the mechanisms for discovering the component Web services of a composite service, while important, do not fall within the scope of this paper. Mechanisms such as UDDI registries could be used.

2 PRELIMINARIES

2.1 Software Agent

A software agent is a piece of software that autonomously acts to carry out tasks on behalf of users [18]. The design of many software agents is based on the approach that the user only has to specify high-level goals instead of issuing explicit instructions, leaving the how and when decisions to the agent. A software agent exhibits a number of features that make it different from other traditional components: autonomy, goal-orientation, collaboration, flexibility, self-starting, temporal continuity, character, communication, adaptation, and, last but not least, mobility (it should be noted that not all these features need to embody a software agent).

2.2 Context

Composed of con and text, context refers to the meaning that can be inferred from an adjacent text. Dey et al. define context as any information that is relevant to the interactions between a user and an environment [14]. This information can be about the circumstances, objects, or conditions by which the user is surrounded. Many researchers have attempted defining context, among them Schilit et al., who propose three categories of context [36]: 1) computing category (e.g., network connectivity, communication cost), 2) user category (e.g., profile, location), and 3) physical category (e.g., lighting, temperature).

To enhance systems with context-aware capabilities, many issues have to be addressed. Satyanarayanan has listed some of them, including [35]: How is context internally represented; where is context stored; does context reside locally, in the network, or in both; how frequently does context information have to be consulted; and, what is the overhead of taking context into account?

2.3 Conversation

A conversation is a sequence of messages that involve two or more participants who intend to achieve a particular purpose [37]. In addition, a conversation is goal-directed, has a task-driven pattern, and is specified with conversation policies that regulate the progress of the information exchange [20]. A conversation either succeeds or fails. On the one side, a conversation succeeds because the outcome expected out of the conversation has been achieved (e.g., action implemented, feedback received). On the other side, a conversation fails because the conversation faced some technical difficulties (e.g., communication-medium disconnected) or the expected outcome has not been achieved (e.g., action requested not-implemented).

The Web Services Conversation Language (WSCL) is one of the initiatives on Web services conversations [10]. WSCL describes the structure of documents that a Web service expects to receive and produce, as well as the order in which the exchange of these documents is to happen. In fact, the conversation component of a service is a means for describing the operations that the service supports (e.g., clients to log in first, then they can request catalog).

3 WEB SERVICES

3.1 Definitions

A Web service is an accessible application that other applications and humans can discover and invoke. Benatallah et al. suggest the following properties for a Web service [7]: 1) independent as much as possible from specific platforms and computing paradigms, 2) mainly developed for interorganizational situations, and 3) easily composable so that developing complex adapters for the needs of composition is not required.

For the needs of our research on Web services [25], [27], [28], we developed *service chart diagrams* as a means for modeling and defining services [24]. A service chart diagram enhances a state chart diagram[1] [17], putting this time the emphasis on the context surrounding the execution of a service rather than only on the states that a service takes (Fig. 1). To this end, the states of a service are wrapped into five perspectives, each perspective having a set of parameters. The state perspective corresponds to the state chart diagram of the service. The flow perspective corresponds to

---

[1] A state chart diagram is a graphical representation of a state machine that visualizes how and under what circumstances a modeled element changes its states. In addition, a state chart diagram depicts the actions that are executed because of the occurrence of specific events.

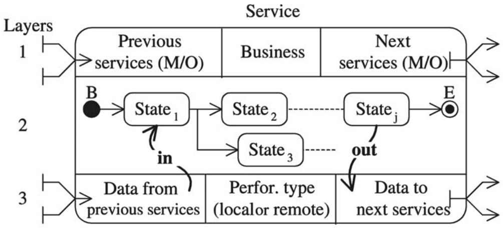


Fig. 1. Service chart diagram of a component service.

the execution chronology of the composite service in which the service participates (Previous services/Next services parameters—M/O for Mandatory/Optional). The business perspective identifies the organizations (i.e., providers) that offer the service (Business parameter). The information perspective identifies the data that are exchanged between the services of the composite service (Data from previous services/Data for next services parameters). Because the services that participate in a composition can be either mandatory or optional, the information perspective is tightly coupled to the flow perspective with regard to mandatory and optional data. Finally, the performance perspective illustrates the ways by which the service is invoked for execution (Performance type parameter, more details on invocation types are given in [23]).

3.2 Composite Services

A composition approach connects Web services together in order to devise composite services. The connection of Web services implements a business logic, which depends on the application domain and control flow of the business case for which the composite service is being devised. Examples of business cases are various, such as travel planning and journal-paper review. It is accepted that the efficiency and reliability of a composite service strongly depend on the commitments, performance, and delivery capabilities of each of the component services. In what follows, an overview of the approaches for developing composite services is presented [12].

3.2.1 Proactive Composition versus Reactive Composition

A proactive composition is an offline process that gathers in-advance available component services to constitute a composite service. The composite service is precompiled and ready to be triggered upon users' requests. In a proactive composition, the component services are usually stable and may possibly be running on resource-rich platforms. A reactive composition is the process of creating composite services on-the-fly. A composite service is devised on a request-basis from users. Because of the on-the-fly property, a component manager is required and ensures the identification and collaboration of the component services. Despite a "certain" complexity of the reactive composition, it has several advantages over the proactive composition, for instance the possibility of tracking the status of the composition process so that corrective actions can be promptly taken and the possibility of optimizing runtime arguments, such as bandwidth use, data transfer routes, and execution charges. Discussions on the pros and cons of each type of composition is available in [21].

3.2.2 Mandatory Composite Service versus Optional Composite Service

A mandatory composite service corresponds to the compulsory participation of all the component services in the execution process. Because it is expected that the component services will be spread over the network, the reliability of the execution process of each component service affects the whole reliability of the composite service. An optional composite service does not necessarily involve all of the component services. Some component services can be skipped during execution due to various reasons, such as the possibility of substitution or nonavailability.

Because a composite service is made up of several component services, the process model underlying the composite service is specified as a state chart diagram (the value-added of state charts to Web services composition is discussed in [6]). In this diagram, states are associated with the service chart diagrams of the component services (Fig. 1) and transitions are labeled with events, conditions, and variable assignment operations. For illustration purposes, Fig. 2 presents *travel assistant composite-service* (`TAS`) as a state chart diagram. It is composed of several dependent services, each service having its service chart diagram: *flight booking* (FB), *hotel booking* (HB), *attraction search* (AS), and *car rental* (CR).

4 AGENTIFICATION OF WEB SERVICES COMPOSITION

Three types of input sources can contribute to a context development: service, user, or both. In [33], Roman and Campbell observe that a user-centric context promotes applications that 1) move with users, 2) adapt according to changes in the available resources, and 3) provide configuration mechanisms based on users' personal preferences. From our side, we advocate that a service-centric context promotes applications that 1) allow service adaptability, 2) deal with service availability, and 3) support on-the-fly service composition. In this paper, the emphasis is on the context of services.

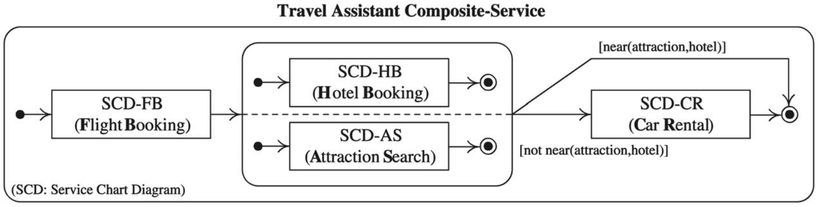

Travel Assistant Composite-Service
SCD-FB (Flight Booking)
SCD-HB (Hotel Booking)
SCD-AS (Attraction Search)
SCD-CR (Car Rental)
[near(attraction,hotel)]
[not near(attraction,hotel)]
(SCD: Service Chart Diagram)

Fig. 2. Travel assistant composite-service as a state chart diagram.

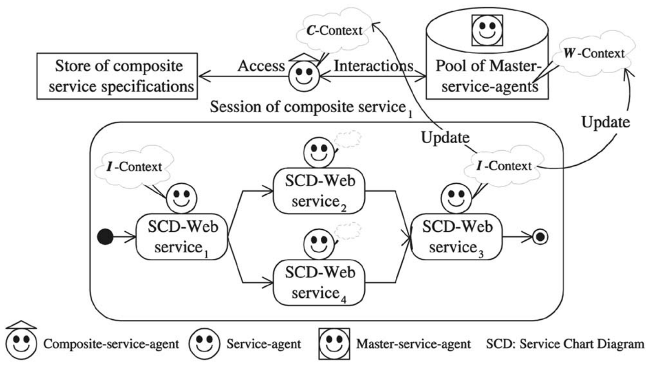


Fig. 3. Agents deployment for Web services composition.

4.1 Deploying Composition Using Agents

The aim of agentifying Web services composition is to determine the appropriate types and roles of agents that deploy the specification of this composition, as Fig. 2 illustrates. Currently, three types of agents are devised: *composite-service-agent*, *master-service-agent*, and *service-agent*. The rationale of each agent is given below.

We consider a Web service as a component that is instantiated each time it participates in a new composition. Prior to any instantiation, several elements related to the Web service are checked. These elements constitute a part of the context, denoted by $\mathcal{W}$-context, of the Web service and are as follows: 1) number of service instances currently running versus maximum number of service instances that can be simultaneously run, 2) execution status of each service instance deployed, and 3) request time of the service instance versus availability time of the service instance.

The role of the master-service-agent is to track the multiple Web services of type instance, which are obtained from a Web service of type root (i.e., similar to class-object principle). Master-service-agents, Web services, and $\mathcal{W}$-contexts are all stored in a pool (Fig. 3). 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. Upon approval, a service instance along a context, denoted by $\mathcal{I}$-context, is created. An authorization of joining a composite service can be rejected because of multiple reasons: period of nonavailability, overloaded status, or exception situation.

To be aware of the running instances of a Web service so that its $\mathcal{W}$-context is updated, the master-service-agent associates each instance it creates with two components (Fig. 3): service-agent and $\mathcal{I}$-context. The service-agent manages the service chart diagram of the service instance and its respective $\mathcal{I}$-context. For example, the service-agent knows the states that the service instance will 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 a constant interaction. Indeed, the content of $\mathcal{I}$-contexts feeds the content of $\mathcal{W}$-contexts with various details including:

1. What is the execution status (in-progress, suspended, aborted, or terminated) of a service instance?
2. When is the execution of a service instance supposed to resume in case it has been suspended and what are the reasons of suspending the execution?
3. When is the execution of a service instance expected to complete? What kind of completion is expected (success or failure)?
4. What are the corrective actions that are taken in case the execution of a service instance fails?

With regard to composite-service-agents, their role is to trigger the specification of the composite services (Fig. 2) and monitor the deployment of this specification. A composite-service-agent ensures that the appropriate component services are involved and collaborate according to a specific specification. When a composite-service-agent downloads the specification of a composite service from the store of specifications, it 1) establishes a context denoted by $\mathcal{C}$-context for the composite service and 2) identifies the first Web services to be triggered. For the sake of simplicity, it is assumed that the Web services constitute a sequence. It is needless to say that our description of the operations is applied to any type of service chronology. When the first Web service is identified, the composite-service-agent interacts with the master-service-agent of this Web service in the objective to ask for service instantiation. If the master-service-agent agrees on the instantiation after checking the $\mathcal{W}$-context (Section 2.3), a service-agent and $\mathcal{I}$-context are both created. Afterwards, the details on the service chart diagram of the new service-instance are 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 that corrective actions are 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 (*next services* of the flow perspective, Fig. 1). In case there are Web services due for execution, the service-agent requests from the composite-service-agent to engage in conversations with their respective master-service-agent (Section 2.3).

The aforementioned description of the agentification approach presents potential advantages for managing services all along their life-cycle, from (dynamic) preparation to actual deployment and recomposition, if needed. The most prominent advantage is the concurrency that exists between Web services composition and execution. While service-agents are in charge of executing the Web service instances, composite-service-agents engage, at the same time, in conversations with master-service-agents of the next Web services to ensure that these next Web services are getting ready for execution. The concurrent composition and execution of services are usually referred to as *interleaving* [26].

Fig. 3 represents an execution session of the composite service CS_1. CS_1 has four primitive component-services (in fact, four Web services of type instance). Each service instance is associated with 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 $\mathcal{C}$-context and $\mathcal{W}$-context of the respective composite-service-agent and master-service-agent. The exchange of information that occurs between master-service-agent and

TABLE 1
Description of the Parameters of $\mathcal{I}$-Context

Label: corresponds to the identifier of the service instance.
Service-agent label: corresponds to the identifier of the service-agent in charge of the service instance.
Status: informs about the current status of the service instance (in-progress, suspended, aborted, terminated).
Previous service instances: indicates whether there were service instances before the service instance (could be null).
Next service instances: indicates whether there will be service instances after the service instance (could be null).
Regular actions: illustrates the actions the service instance performs.
Begin time: informs when the execution of the service instance has started.
End time (expected and effective): informs when the execution of the service instance is expected to terminate and has effectively terminated.
Reasons of failure: informs about the arguments that caused the failure of the execution of the service instance.
Corrective actions: illustrates the actions that the service instance performs because the execution has failed.
Date: identifies the time of updating the parameters above.

service-agent has already been discussed in the previous paragraphs. In addition to a complete copy (or part based on the level of details that needs to be tracked) of that exchange that is sent to composite-service-agents, these ones receive extra details from service-agents including: 1) the next services to be called for execution and 2) the type of these services either mandatory or optional (*next services* parameter of the flow perspective, Fig. 2).

4.2 Modeling $\mathcal{I}/\mathcal{W}/\mathcal{C}$-Contexts

Besides the three types of agents that Fig. 3 encompasses, three types of services are considered, namely, composite service, Web service, and Web service instance. Each service is attached to a specific context. $\mathcal{I}$-context has the fine grained content, whereas $\mathcal{C}$-context has the coarse grained content. The $\mathcal{W}$-context is in between. Details on $\mathcal{I}$-context update $\mathcal{W}$-context and details on $\mathcal{W}$-context update $\mathcal{C}$-context. We use Tuple Spaces to support the update operations between contexts [1], but this is outside this paper's scope.[2]

The $\mathcal{I}$-context of a Web service instance consists of the following parameters (Table 1): label, service-agent label, status, previous service instances, next service instances, regular actions, begin time, end-time expected, end-time effective, reasons of failure, corrective actions, and date.

The $\mathcal{W}$-context of a Web service is built upon the $\mathcal{I}$-contexts of its respective component Web service instances and consists of the following parameters (Table 2): label, master-service-agent label, number of instances allowed, number of instances running, next service instance availability, status/service instance, and date.

The $\mathcal{C}$-context of a composite service is built upon the $\mathcal{W}$-contexts of its respective Web services and consists of the following parameters (Table 3): label, composite-service-agent label, previous Web services, current Web service, next Web services, begin time, and date.

It was pointed out in the beginning of Section 4 that a service-centric context is adopted in this paper. This is shown with the $\mathcal{W}$-context of a Web service, which constitutes the link between $\mathcal{I}$-contexts and $\mathcal{C}$-contexts. A service-centric context promotes service adaptability, availability, and dynamic composition. The agentification approach for the Web services composition of Fig. 3 meets these three requirements. A composite service might have to adapt its list of component Web services because of the availability of certain of these components. Availability is illustrated with the maximum number of service instances that can be created versus the current number of service instances that are running. Since Web services are instantiated on a request-basis and according to their context, this means that a dynamic composition is supported.

Because of the aforementioned requirements, $\mathcal{W}$-context is organized along three interconnected perspectives (Fig. 4):

1. *Instance perspective* is about creating service instances (e.g., number of instances allowed, number of instances running), assigning them to composite services, and getting the next service instances ready for execution.
2. *Execution perspective* is about meeting the computing resource requirements of service instances, tracking their execution status, and avoiding conflicts on these resources.
3. *Time perspective* is about time-related parameters of service instances and constraints on service instances (e.g., period of request, period of availability, end-time expected).

In Fig. 4, the three perspectives are connected to each other. First, deployment denotes the connection between instance and execution perspectives and reflects the Web services that are instantiated for execution. Second, tracking denotes the

---

[2] A sample of a control tuple illustrating an update between contexts: **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, then the respective $\mathcal{W}$-context w of the web service ws also needs to be updated after collecting information from this $\mathcal{I}$-context i.

TABLE 2
Description of the Parameters of $\mathcal{W}$-Context

Label: corresponds to the identifier of the Web service.
Master-service-agent label: corresponds to the identifier of the master-service-agent in charge of the Web service.
Number of instances allowed: corresponds to the maximum number of service instances that can be created from the Web service.
Number of instances running: corresponds to the number of service instances of the Web service that are currently running.
Next service instance availability: corresponds to when a new service instance of the Web service will be made available.
Status/Service instance: corresponds to the status of each service instance of the Web service that is deployed (based on status parameter of $\mathcal{I}$-context).
Date: identifies the time of updating the parameters above.

TABLE 3
Description of the Parameters of $\mathcal{C}$-Context

Label: corresponds to the identifier of the composite service.
Master-service-agent label: corresponds to the identifier of the composite-service-agent in charge of the composite service.
Previous Web services: indicates which Web services of the composite service have been executed with regard to the current Web services.
Current Web services: indicates which Web services of the composite service are currently under execution.
Next Web services: indicates which Web services of the composite service will be called for execution with regard to the current Web services..
Begin time: informs when the execution of the composite service has started.
Status/Web service: corresponds to the status of each Web service instance of the composite service that is deployed (based on status parameter of $\mathcal{I}$-context).
Date: identifies the time of updating the parameters above.

connection between execution and time perspectives and reflects the monitoring of the instances of Web services that occurs over a certain period of time. Finally, availability denotes the connection between time and instance perspectives and reflects the continuous verification that the Web services perform before they commit additional service instances for a certain period of time.

The use of context ensures that the requirements of and constraints on the services to participate in a composition process are taken into account. While current Web services composition approaches rely on different selection criteria, such as execution cost, execution time, and reliability [39], it is deemed appropriate, including context of services, in this composition, particularly when a reactive composition of Web services is adopted. Doulkeridis et al. also back the importance of including context during Web services composition [15]. We have shown that services take part in a composition based on their availabilities and the current needs of the composite services in component services. Moreover, the use of context is suitable for tracing the execution of Web services during exception handling. It would be possible to know at any time what happened and what is happening with a Web service and all its respective instances. Predicting what will happen to a Web service would also be feasible in case the previous contexts (i.e., what happened to a service) are stored. In [19], Kouadri Mostéfaoui makes a reference to different types of context, including user, computing, time, and history. History context stores details on other contexts for future and further use. Applications can make use of not just the current context, but also past contexts to adapt their behavior for better actions and interactions with users.

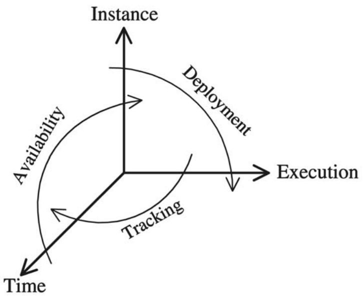


Fig. 4. Perspectives of a service-centric context.

4.3 Managing Conversations between Agents

In the rest of this paper 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 the agentification approach of Fig. 3, the selection of the

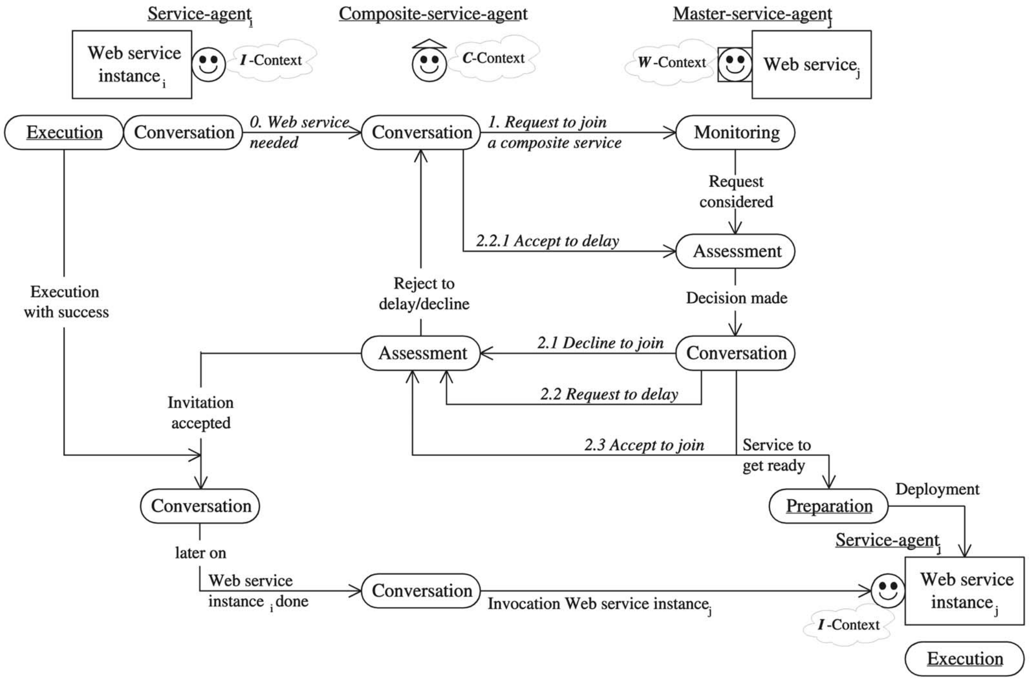


Fig. 5. Conversation diagram between agents.

component services to constitute a composite service is carried out on-the-fly. The selection operations are outsourced to composite-service-agents that engage in 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 $\mathcal{W}$-contexts. In case of a positive decision, Web service instances, service-agents, and $\mathcal{I}$-contexts are all deployed.

When a Web service instance is being executed, its service-agent checks the service chart diagram of this service instance (Fig. 1). The purpose is to check if additional Web services have to be executed. If yes, the service-agent requests from the composite-service-agent to engage in conversations with the master-service-agents of these Web services. These conversations have two aims: 1) invite master-service-agents and, thus, their Web service to participate in the composition process and 2) ensure that the Web services are ready for instantiation following their invitation acceptance.

Fig. 5 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$_{(1,\ldots,i,j,\ldots,n)}$. 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 to involve Web service$_j$ (0), the composite-service-agent engages in conversations with master-service-agent$_j$ (1). This service is an element of the composite service that is under preparation. A composite service is decomposed into three parts. The first part corresponds to the Web service instances that have successfully completed their execution (Web services$_{1,\ldots,i-1}$, *previous Web services* parameter of Table 3). The second part corresponds to the Web service instance that is now being executed (Web service instance$_i$, *current Web services* parameter of Table 3). Finally, the third part corresponds to the rest of the composite service that is due for execution and, hence, has to get ready for execution (Web services$_{j,\ldots,n}$, *next Web services* parameter of Table 3). 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: decline the request, delay its making decision, or accept the request.

4.3.1 Option A

Master-service-agent$_j$ of Web service$_j$ declines the request of the composite-service-agent. A conversation message is sent back 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).[3]
- Otherwise (i.e., Web service$_j$ is mandatory), the composite-service-agent engages in further conversations with master-service-agent$_j$, questioning about, for example, the reasons of rejection or the availability of the next instance of Web service$_j$.

4.3.2 *Option B*

Master-service-agent$_j$ of Web service$_j$ cannot make a decision before the deadline of response that the composite-service-agent has fixed. Thus, master-service-agent$_j$ requests from the composite-service-agent if there is a room to extend the deadline (2.2). The composite-service-agent has two alternatives taking into account the $\mathcal{C}$-context of the composite service and the fact that a component service can be either mandatory or optional:

- Refuse to extend the deadline as requested by master-service-agent$_j$. This means that the composite-service-agent has to start again engaging in conversations with another master-service-agent$_{k,\ (k \neq j)}$ (**Option A**).
- Accept to extend the deadline as requested by master-service-agent$_j$. This means that master-service-agent$_j$ will be notified about the acceptance of the composite-service-agent (2.2.1). After receiving the acceptance, master-service-agent$_j$ of Web service$_j$ enters again the assessment state and checks the $\mathcal{W}$-context in order to make a decision on whether to join the composite service (**Option A**). A master-service-agent may request a deadline extension for several reasons, e.g., additional instances of Web service$_j$ cannot be committed before other instances complete their execution (Table 2).

4.3.3 *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 [22]. At the same time, master-service-agent$_j$ ensures that Web service$_j$ is getting ready for execution through the preparation state (i.e., deploy $\mathcal{I}$-context and service-agent$_j$).

When the execution of Web service instance$_i$ is completed, service-agent$_i$ informs the composite-service-agent about that. According to the agreement that is established in **Option C**, the composite-service-agent interacts with service-agent$_j$ so that 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$_j$. It should be noted that the content of agreements is beyond the scope of this paper.

5 SECURITY OF WEB SERVICES

An exhaustive list of the security risks that can prevent the widespread adoption of Web services is reported in [4], [38]. These risks range from authentication of the requesters and providers of services to denial of service and data integrity of WSDL files. In addition, Aissi et al. claim in [2] that the biggest challenge for Web services is the fragmentation of the security requirements. The inability to describe where and how to apply security measures is a large gap in the description of Web services. In this paper, because Web services require the computing resources of hosts on which they are executed, it is important to ensure that neither the services *misuse* the resources of hosts nor that the hosts *alter* the integrity of the services. In the following, we highlight the first part of the security strategy, which consists of preventing services from misusing the computing resources of hosts. The second security strategy, which consists of protecting services from malicious hosts, is part of our future work. It should be noted here that the security of services is discussed at the instance level.

5.1 Service-Based Access Control

The Role-Based Access Control (RBAC) is a well-known strategy for managing access rights of users in businesses [34]. In a business, it is common that users fulfill various roles. For each role a set of rights to use some resources are granted. 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. Therefore, 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, word processor) to request some sensitive information, such as a banking-application secret code.

To handle the aforementioned security situations, we designed a *service-based access control architecture*. Fig. 6 represents this architecture where numbers correspond to the steps that grant a Web service the right to run on the resources of a host. The rationale of monitoring module, security context, and pattern database is explained in the following paragraphs. It should be mentioned the security context of Fig. 6 is totally different from the contexts of Section 4.2.

First of all, a service submits a request of use to the monitoring module (1). Upon receiving the request, the monitoring module checks the security context of the resource that this service is about to use (2). Details on the security context are given in Section 5.2. Afterwards, the monitoring module browses the pattern database (3). The objective is to identify any *malicious pattern* that might exist in the database and present some similarities with the execution trace of the service that requests the use of the resources. Details on the pattern database are given in Section 5.3. A response to grant or deny the request for use is sent back to the service (4). Finally, the request is implemented (5) in case of a positive response from the monitoring module.

---

[3] Web service$_k$ is a substitute for Web service$_j$. If a substitute service does not exist, the composite-service-agent bypasses Web service$_j$ and works on Web service$_{j+1}$.

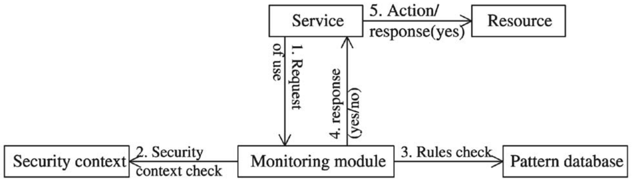


Fig. 6. Service-based access control architecture.

5.2 Security Context Specification

In Fig. 6, the monitoring module decides if a service has the right to use a resource. To make this decision, 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. The calling chain $(S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_n)$ of the service being currently monitored is determined upon the collaboration happening between the monitoring module and the service-agent of this service. A request, which originates from the monitoring module to the service-agent, allows collecting information on the calling chain. The service-agent uses *previous service instances* parameter of the service's $\mathcal{I}$-context (Table 1) in order to constitute the calling chain, namely, the list of services being called for execution. With regard to the resources in Fig. 6, 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 or deny the use of a resource R is based on the algorithm of Fig. 7.

5.3 Pattern Database

The pattern database aims at storing the *potential* threats on the resources, which are deployed in an environment of Web services. Each threat is identified with a pattern. A pattern is determined based on the execution trace of a service with regard to the sequence of primitive actions that the service implements. Threats are, first, parameterized with resource labels (e.g., file name, port number, etc.) and, second, associated with rules that specify under which circumstances a threat is malicious. It is planned to create and update the pattern database offline. In addition, only the system hosting the database has the right to use it so that malicious accesses are prevented.

Relying on the malicious pattern database, the monitoring module controls both the security context of any use request to a resource as well as the primitive actions that a service is going to be performed. For illustration purposes, we 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.

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

Fig. 7. Validation algorithm for a resource request of use. `Check_rules()` is a function that uses a database of malicious patterns. The function guarantees that the actions to be performed before granting the right to use the resource R do not constitute a malicious attack. Otherwise, the request of use is denied.

```
Read · Send is a malicious pattern iff
Read:  read x
Send:  send x
x:  secret
```

Fig. 8. Example of a specification rule of a malicious pattern.

Malicious patterns are detected using traces and data security levels. Data are labeled with security levels including public, shared, and secret. A malicious pattern is specified by one or several rules. Each rule specifies why a service is considered malicious. Fig. 8 illustrates a specification rule of a malicious pattern. It states that $Read \cdot Send$ is a malicious pattern if-and-only-if Read is an action of reading a secret data x and Send is an action of sending the same secret data x out through the network.

6 IMPLEMENTATION STATUS

We overview the progress of implementing the agent-based and context-oriented approach for Web services composition. After assessing our objectives and constraints, we decided to focus on a rapid and flexible prototyping of this approach rather than low-level implementation details. We have adopted Borland JBuilder Enterprise Edition 9.0.[4] JBuilder has a toolkit for building, testing, and deploying Web services. JBuilder includes as well a Web services explorer facility for publishing and searching for Web services.

The prototype architecture of the agent-based and context-oriented approach for Web services composition is depicted using a class model (Fig. 9). The following outlines the functionality of some classes:

- *Service chart diagram* is used to specify Web services and Web service instances.
- *State chart diagram* is used to specify composite services. A state chart diagram aggregates the service chart diagrams of the component Web services of a composite service.
- *Agents* is used to specify all agents (service-agent, master-service-agent, and composite-service-agent) of the prototype. Each agent has a label, type, and location, which is the platform where it resides.
- *Context*, with its types ($\mathcal{I}$-context, $\mathcal{W}$-context, $\mathcal{C}$-context), is responsible for gathering, processing, and parsing contextual information. It translates the information parsed into XML documents (Appendix A, available on the IEEE Computer Society's Digital Library at http://www.computer.org/publications/dlib) and attaches each context to the appropriate agent.
- *Conversation* is the central unit for managing and maintaining conversations between all agents of the prototype. Conversations are represented as XML documents.

The validation of the model of Fig. 9 was started with developing a `Web services composition manager`. The manager offers a set of tools that allow Web services'

---

[4] http://www.borland.com/jbuilder/enterprise/index.html.

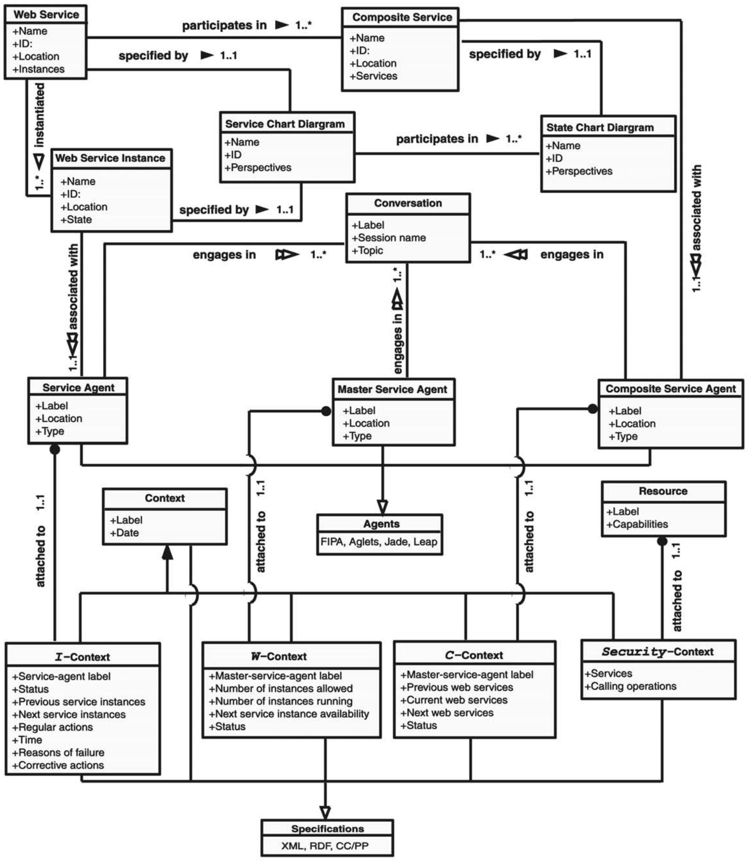


Fig. 9. Prototypical class model.

providers and users to create, compose, and execute services based on the different contexts. WSDL is used for Web services specification, whereas UDDI is used for Web services announcement and discovery. We recall that the details of each type of context are presented as XML files. For example, Appendix A (available on the IEEE Computer Society's Digital Library at http://www.computer.org/publications/dlib) illustrates the XML code associated with $\mathcal{I}$-context. A dedicated XML editor was implemented in order to develop and validate the different context files, which have all to comply with specific DTDs.

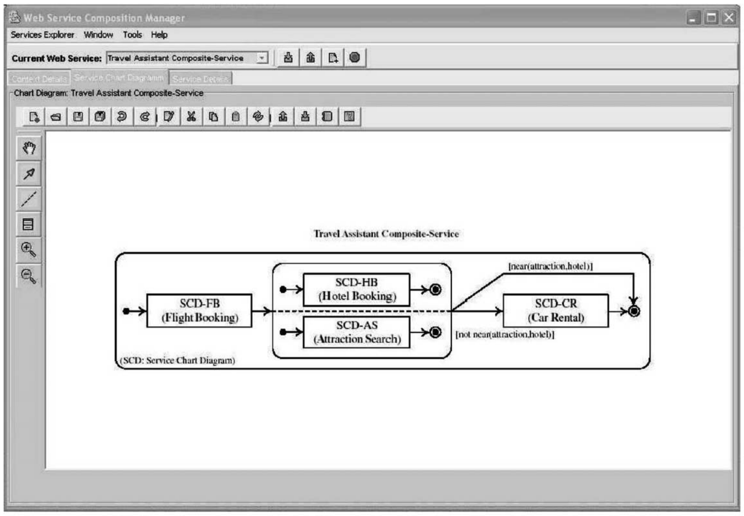
The Web services composition manager also offers an editor for describing service chart diagrams of a component service participating in composite services (Fig. 10). The editor provides tools for directly manipulating service chart diagrams, states, and transitions (add, remove, modify, etc.) graphically using simple drag and drop actions. All changes are reflected in the corresponding context details and WSDL description files.

7 RELATED WORK TO THE COMPOSITION APPROACH

There are several research initiatives in the field of Web services [31]. However, to our knowledge, none of these initiatives have attempted including context in the composition process of Web services. Besides the traditional selection criteria that are used in a similar process (e.g., execution cost and execution time), we have shown that context has a major effect on the Web services, in general, and their composition, in particular. For example, reaching the maximum number of instances that can be obtained from a Web service definitely delays the development of a composite service. Furthermore, being aware of when these instances will become available helps adjust the execution of the composite service. We have also shown that the context has been part of the conversations that agents of the Web services engage. In the rest of this section, we highlight some of the works that have supported our thoughts and inspired us shape our agent-based and context-oriented approach for Web services composition.

In the service chart diagram of Fig. 1, the flow perspective illustrates the pre and postcomponent services of a composite service. These component services can be either optional or mandatory. These types of services are similar to what Berfield et al. call vital and nonvital services for a workflow application in [9]. Another use of *previous* and *next services* of the flow perspective is related to pre and postconditions. These conditions specify what must be true before a component service can be executed and what will be true as a result of the component service execution.

In Section 2.3, we introduced the Web Services Conversation Language of [10]. While this language focuses on specifying the operations that Web services support, our conversations focus on the mechanisms of establishing composite services. We classify these mechanisms into four categories: domain-dependent, domain-independent, composition-driven, and execution-driven. Domain-dependent conversation mechanisms deal with the features of the application domain when it comes to structuring the format and content of the conversations to occur. For example, conversations for car rental are different from those for hotel booking. The opposite occurs for domain-independent



Current Web Service: Travel Assistant Composite-Service

Chart Diagram: Travel Assistant Composite-Service

Fig. 10. Editor for service-chart diagram definition.

conversation mechanisms when it comes, for example, to trigger a service for execution. The domain does not affect the way a service is triggered. Instead, the implementation technology on which the Web services are deployed affects the triggering of the service. With regard to composition-driven conversations, they represent the conversations that are needed to devise a composite service, such as how to look for the component Web services and how to exchange agreements between these component Web services. Execution-driven conversation mechanisms represent the messages that are needed to deploy a composite service. Therefore, the chronology of conversations starts with composition-driven conversations and continues with execution-driven conversations.

Another use of conversations in Web services enables dealing with the composability issue of Web services as Medjahed et al. report in [29]. For instance, mapping operations of the information exchanged between Web services may be required. Ensuring the composability of Web services can be achieved through conversations. For instance, agents engage in conversations for agreeing on what to exchange, how to exchange, when to exchange, and what to expect from an exchange.

In Table 1, parameters of type expected versus parameters of type effective have a major overlapping with the QoS of type advertised (or "promised") versus QoS of type delivered [30]. Ouzzani and Bouguettaya report that a key feature in distinguishing between competing Web services is their QoS, which encompasses several qualitative and quantitative parameters that measure how well the Web service delivers its functionalities [30]. A Web service may not always fulfill its advertised QoS parameters due to various fluctuations related for example to the network status or resource availability. Therefore, some differences between QoS advertised and QoS delivered values occur. However, large differences indicate that the Web service is suffering a performance degradation in delivering its functionalities. The same comment is made on parameters of type expected versus parameters of type effective of Table 1. A major difference between *end-time expected* and *end-time effective* indicates the obstacles that might have faced the execution of a Web service.

8 CONCLUSION AND FUTURE WORK

In this paper, we presented our approach for composing Web services using the concepts of software agent and context. Several types of agents have been put forward, namely, composite-service-agents associated with composite services, master-service-agents associated with Web services, and, finally, service-agents associated with Web service instances. The different agents are aware of the context of their respective services in the objective to devise composite services on-the-fly. To reach this objective, three types of context have been 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 in conversations to decide if service instances can be 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 be deployed at the same time and the availability of these instances for a certain period of time.

Our future work includes the completion of the proof-of-concept prototype. We also plan to examine the support for exception handling during conversation-based service composition. Software agents of composite services could perform some run-time changes of the specification of composition by adding new component services, removing certain component services, or replacing certain component services.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their comments and suggestions of improvements. Thanks are also due to Professor W.J. van den Heuvel from Tilburg University in the Netherlands for his feedback on earlier versions of this paper.

REFERENCES

[1] S. Ahuja, N. Carriero, and D. Gelernter, "Linda and Friends," *Computer*, vol. 19, no. 8, Aug. 1986.
[2] S. Aissi, P. Malu, and K. Srinivasan, "E-Business Process Modeling: The Next Big Step," *Computer*, vol. 35, no. 5, May 2002.

[3] L. Ardissono, A. Goy, and G. Petrone, "Enabling Conversations with Web Services," *Proc. Second Int'l Joint Conf. Autonomous Agents and Multi-Agent Systems (AAMAS '03),* 2003.

[4] A. Barbir, "Web Services Security: An Enabler of Semantic Web Services," *Proc. Business Agents and the Semantic Web,* held in conjunction with the 16th Canadian Conf. Artificial Intelligence (AI '03), 2003.

[5] B. Benatallah, F. Casati, and F. Toumani, "Web Service Conversation Modeling, A Cornerstone for E-Business Automation," *IEEE Internet Computing,* vol. 8, no. 1, Jan./Feb. 2004.

[6] B. Benatallah, M. Dumas, Q.Z. Sheng, and A. Ngu, "Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services," *Proc. 18th Int'l Conf. Data Eng. (ICDE '02),* 2002.

[7] B. Benatallah, Q.Z. Sheng, and M. Dumas, "The Self-Serve Environment for Web Services Composition," *IEEE Internet Computing,* vol. 7, no. 1, Jan./Feb. 2003.

[8] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella, "A Foundational Vision for e-Services," *Proc. Workshop Web Services, e-Business, and the Semantic Web (WES '03),* held in conjunction with the 15th Conf. Advanced Information Systems Eng. (CAiSE '03), 2003.

[9] A. Berfield, P.K. Chrysanthis, I. Tsamardinos, M.E. Pollack, and S. Banerjee, "A Scheme for Integration E-Services in Establishing Virtual Enterprises," *Proc. 12th Int'l Workshop Research Issues in Data Eng.: Eng. e-Commerce/e-Business Systems (RIDE '02),* 2002.

[10] D. Beringer, H. Kuno, and M. Lemon, "Using WSCL in a UDDI Registry 1.02," http://www.uddi.org/pubs/wsclBPforUDDI_5_16_011.doc, 2001.

[11] P. Brézillon, "Focusing on Context in Human-Centered Computing," *IEEE Intelligent Systems,* vol. 18. no. 3, May/June 2003.

[12] D. Chakraborty and A. Joshi, "Dynamic Service Composition: State-of-the-Art and Research Directions," Technical Report TR-CS-01-19, Dept. of Computer Science and Electrical Eng., Univ. of Maryland, 2001.

[13] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana, "The Next Step in Web Services," *Comm. ACM,* vol. 46, no. 10, Oct. 2003.

[14] 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 J.,* special issue on context-aware computing, vol. 16, no. 1, 2001.

[15] C. Doulkeridis, E. Valavanis, and M. Vazirgiannis, "Towards a Context-Aware Service Directory," *Proc. Fourth Workshop Technologies for E-Services (TES '03),* held in conjunction with the 29th Int'l Conf. Very Large Data Bases (VLDB '03), 2003.

[16] S. Evren, P. Bijan, and H. James, "Composition-Driven Filtering and Selection of Semantic Web Services," *Proc. 2004 AAAI Spring Symp. Semantic Web Services,* 2004.

[17] D. Harel and A. Naamad, "The STATEMATE Semantics of Statecharts," *ACM Trans. Software Eng. and Methodology,* vol. 5, no. 4, Oct. 1996.

[18] N. Jennings, K. Sycara, and M. Wooldridge, "A Roadmap of Agent Research and Development," *Autonomous Agents and Multi-Agent Systems,* vol. 1, no. 1, 1998.

[19] S. Kouadri Mostéfaoui, "Towards a Context-Oriented Services Discovery and Composition Framework," *Proc. AI Moves to IA: Workshop Artificial Intelligence, Information Access, and Mobile Computing,* held in conjunction with the 18th Int'l Joint Conf. Artificial Intelligence (IJCAI '03), 2003.

[20] F. Lin and D.H. Norrie, "Schema-Based Conversation Modeling for Agent-Oriented Manufacturing Systems," *Computers in Industry,* vol. 46, no. 3, Oct. 2001.

[21] S.W. Loke, "Proactive and Reactive Discovery, Composition, and Activation of Localized Services Accessed from Mobile Devices," *Proc. AI Moves to IA: Workshop Artificial Intelligence, Information Access, and Mobile Computing,* held in conjunction with the 18th Int'l Joint Conf. Artificial Intelligence (IJCAI '03), 2003.

[22] H. Ludwig, A. Keller, A. Dah, and R. King, "A Service Level Agreement Language for Dynamic Electronic Services," *Proc. 4th IEEE Int'l Workshop Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS '02),* 2002.

[23] Z. Maamar, "Moving Code (Servlet Strategy) versus Inviting Code (Applet Strategy)—Which Strategy to Suggest to Software Agents?" *Proc. Third Int'l Conf. Enterprise Information Systems (ICEIS '01),* 2001.

[24] Z. Maamar, B. Benatallah, and W. Mansoor, "Service Chart Diagrams—Description and Application," *Proc. Alternate Tracks of the 12th Int'l World Wide Web Conf. (WWW '03),* 2003.

[25] Z. Maamar and W. Mansoor, "Design and Development of a Software Agent-Based and Mobile Service-Oriented Environment," *e-Service J.,* vol. 2 no. 3, 2003.

[26] Z. Maamar, Q.Z. Sheng, and B. Benatallah, "Interleaving Web Services Composition and Execution Using Software Agents and Delegation," *Proc. First Int'l Workshop on Web Services and Agent-Based Eng. (WSABE '03),* held in conjunction with the Second Int'l Joint Conf. Autonomous Agents and Multi-Agent Systems (AAMAS '03), 2003.

[27] Z. Maamar, Q.Z. Sheng, and B. Benatallah, "On Composite Web Services Provisioning in an Environment of Fixed and Mobile Computing Resources," *Information Technology and Management J.,* special issue on workflow and e-business, vol. 5, no. 3, 2004.

[28] Z. Maamar, H. Yahyaoui, and W. Mansoor, "Design and Development of an M-Commerce Environment: The E-CWE Project," *J. Organizational Computing and Electronic Commerce,* vol. 14, no. 4, 2004.

[29] B. Medjahed, A. Rezgui, A. Bouguettaya, and M. Ouzzani, "Infrastructure for E-Government Web Services," *IEEE Internet Computing,* vol. 7 no. 1, Jan./Feb. 2003.

[30] M. Ouzzani and A. Bouguettaya, "Efficient Access to Web Services," *IEEE Internet Computing,* vol. 8, no. 2, Mar./Apr. 2004.

[31] M. Papazoglou and D. Georgakopoulos, "Introduction to the Special Issue on Service-Oriented Computing," *Comm. ACM,* vol. 46, no. 10, Oct. 2003.

[32] A. Ranganathan and R.H. Campbell, "A Middleware for Context-Aware Agents in Ubiquitous Computing Environments," *Proc. ACM/IFIP/USENIX Int'l Middleware Conf. (Middleware '03),* 2003.

[33] 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, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, 2002.

[34] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," *Computer,* vol. 20, no. 2, Feb. 1996.

[35] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *IEEE Personal Comm.,* vol. 8, no. 4, Aug. 2001.

[36] B. Schilit, N. Adams, and R. Want, "Context-Aware Computing Applications," *Proc. IEEE Workshop Mobile Computing Systems and Applications,* 1994.

[37] I.A. Smith, P.R. Cohen, J.M. Bradshaw, M. Greaves, and H. Holmback, "Designing Conversation Policies using Joint Intention Theory," *Proc. Third Int'l Conf. Multi-Agent Systems (ICMAS '98),* 1998.

[38] A. Yang, "Web Services Security," *eAI J.,* Sept. 2002.

[39] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q.Z. Sheng, "Quality Driven Web Services Composition," *Proc. 12th Int'l World Wide Web Conf. (WWW '03),* 2003.

Zakaria Maamar received the PhD degree in computer science from Laval University, Quebec, Canada in 1998. Currently, he is an associate professor in the College of Information Systems at Zayed University, Dubai, United Arab Emirates. His research interests lie in the areas of mobile computing, Web/mobile services, and software agents.

Soraya Kouadri Mostéfaoui is a PhD candidate in the Pervasive and Artificial Intelligence Research Group in the Computer Science Department of the University of Fribourg, Switzerland. Her research interests include ubiquitous computing, context-aware computing, Web services, and emergent behaviors.

Hamdi Yahyaoui is currently pursuing the PhD degree regarding the acceleration and semantic foundations of embedded Java Virtual Machines. He is a researcher in the Languages, Semantics, and Formal Methods Research Group at the Computer Sciences Department of Laval University, Quebec, Canada. His research interests lie in the areas of formal semantics, static analysis, Java acceleration, and Web services. He actively participated in the implementation of a Java optimizing compiler and a lightweight dynamic compiler for embedded Java Virtual Machines.