# TOWARDS AN AGENT-BASED AND CONTEXT-ORIENTED APPROACH TO COMPOSE WEB SERVICES

Zakaria Maamar$^{\alpha}$, Soraya Kouadri Mostéfaoui$^{\beta}$, Hamdi Yahyaoui$^{\gamma}$, and Willem Jan van den Heuvel$^{\delta}$

$^{\alpha}$*Zayed University, Dubai, U.A.E*

$^{\beta}$*University of Fribourg, Fribourg, Switzerland*

$^{\gamma}$*Laval University, Quebec, Canada*

$^{\delta}$*Tilburg University, Tilburg, The Netherlands*

Keywords: Web service, software agent, composition, context, conversation.

Abstract: We present an agent-based and context-oriented approach for the composition of Web services. A Web service is an accessible application that other applications and humans can discover and trigger to satisfy multiple needs (e.g., hotel booking). Because of the complexity that characterizes the composition of Web services, two concepts in this paper are put forward to reduce this complexity namely software agent and context. A software agent is an autonomous entity that acts on behalf of users, whereas context is any relevant information that characterizes a situation. During the composition process, software agents engage conversations with their peers to agree on which Web services will participate in this process. In these conversations, agents take into account the execution context of the Web services.

## 1 INTRODUCTION

Web services are nowadays emerging as a major technology for deploying automated interactions between distributed and heterogeneous applications. Various technologies back this deployment such as WSDL, UDDI, and SOAP (Curbera et al., 2003). The technologies cited support the definition of Web services, their advertisement to the community of users, and finally their binding for triggering purposes.

A Web service is an accessible application that other applications and humans can automatically discover and invoke (Benatallah et al., 2003). In general, composing multiple Web services (also called services in the rest of the paper) rather than accessing a single service is essential and provides more benefits to users (Berardi et al., 2003). Discovering the component services, inserting the services into a *composite service*, triggering the composite service for execution, and monitoring this execution for the needs of exception handling are among the operations that users will have to be responsible. 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. A SA is an autonomous entity that acts on user's behalf, makes decisions, interacts with other agents, and migrates to distant hosts if needed (Jennings et al., 1998).

Entrusting the composition of Web services to software agents is challenging. Various challenges need to be tackled 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 are coordinated so conflicts are avoided, and 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* in which the composition and execution of the Web services happen. Context is the information that characterizes the interaction between humans, applications, and the surrounding environment (Brézillon, 2003). For instance, before provisioning a Web service for execution the computing capabilities of the resources *vs.* 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. In this paper, we present our *agent-based and context-oriented approach for Web services composition*. Existing approaches for service composition, such as WSFL and BPEL4WS, typically facilitate orchestration only, while neglecting information about the context surrounding users and services.

In our agent-based and context-oriented approach, we leverage the interactions that occur between agents during the composition of Web services to the level of *conversations*. A conversation is a consistent exchange of messages between participants involved in joint operations and thus, have common interests. Agents engage conversations with their peers when it comes for example to search for the component ser-

Maamar Z., Kouadri Mostéfaoui S., Yahyaoui H. and Jan van den Heuvel W. (2004).
TOWARDS AN AGENT-BASED AND CONTEXT-ORIENTED APPROACH TO COMPOSE WEB SERVICES.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 214-221
DOI: 10.5220/0002600102140221
Copyright © SciTePress

vices, to check the availability of these services, and to trigger these services.

Section 2 defines Web services. Section 3 presents the agentification of Web services composition. Section 4 discusses the implementation of the discussed approach. Section 5 overviews related work. Finally, Section 6 draws our conclusions and summarizes our future work. We note 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.

## 2 BACKGROUND - WEB SERVICES

A Web service is an accessible application that other applications and humans as well can discover and trigger. A Web service is (Benatallah et al., 2003): (i) independent as much as possible from specific platforms and computing paradigms; (ii) developed mainly for inter-organizational situations rather than for intra-organizational situations; and (iii) easily composable (i.e., its composition with other Web services does not require complex adapters).

For the needs of our research on Web services such as the one we conducted in (Maamar and Mansoor, 2003), we developed *Service Chart Diagrams* (SCDs) as a specification means (Maamar et al., 2003). A SCD leverages the state chart diagram of UML (Harel and Naamad, 1996), putting the emphasize on the context surrounding the execution of a Web service rather than only on the states of a Web service (Fig. 1).

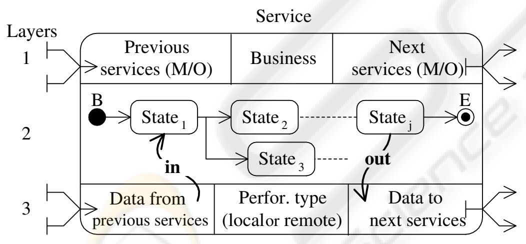

Figure 1: Service chart diagram

A SCD wraps the states of a Web service into five perspectives, each perspective having a set of attributes. The *state* perspective corresponds by default to the state chart of the service. The *flow* perspective corresponds to the execution chronology of the composite service in which the service appears (Previous services/Next services attributes - M/O respectively stands for Mandatory and Optional). The *business* perspective identifies the organizations that are ready for providing the service (Business attribute). The *information* perspective identifies the data that are exchanged between the services of the composite service (Data from previous services/Data for next services attributes). Because these services can be either mandatory or optional, the information perspective is tightly coupled to the flow perspective with regard to the mandatory data and optional data. Finally, the *performance* perspective illustrates the ways (remotely or locally) the service is invoked for execution (Performance type attribute).

Because a composite service is made of up of several component services, the process model underlying this composite service is specified as a state chart diagram whose (i) states are associated with the service chart diagrams of the component services (Fig. 1) and (ii) transitions are labelled with events, conditions, and variable assignment operations.

## 3 AGENTIFICATION OF WEB SERVICES COMPOSITION

Three types of input sources can contribute to the development of a context: service, user, or both. In (Roman and Campbell, 2002), the authors observe that a user-centric context promotes applications that (i) move with users, (ii) adapt according to changes in the available resources, and (iii) provide configuration mechanisms based on users' preferences. We advocate that a service-centric context promotes applications that (i) allow service adaptability, (ii) deal with service availability, and (iii) support on-the-fly service composition. In this paper, the focus is on the context of services.

### 3.1 Agent-based deployment

The objective of agentifying Web services composition is to determine the appropriate types and roles of agents that will deploy this composition. Currently, we are considering 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 participates 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:

- The number of service instances currently running *vs.* the maximum number of service instances that simultaneously can be run.
- The execution status and location of each service instance deployed.
- And, the time of request of the service instance *vs.* the time of availability of the service instance.

The role of the master-service-agent is to track the Web service instances that are obtained from a Web service. Master-service-agents, Web services, and

$\mathcal{W}$-contexts are all stored in a pool (Fig. 2). A master-service-agent processes the requests of instantiation that are submitted to a Web service. These requests originate from composite-service-agents of 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 because for example of overloaded status.

To be informed about the running instances of a Web service so the $\mathcal{W}$-context is 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 (Fig. 1) and the $\mathcal{I}$-context of the service instance. 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 constant interactions. Indeed, the $\mathcal{I}$-contexts feed the $\mathcal{W}$-contexts with various details including:

1. What is the execution status (in-progress, suspended, aborted, ...) of a service instance?
2. When is the execution of a service instance supposed to resume in case it has been suspended?
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 specifications of the composite services and monitor the deployment of these specifications. A composite-service-agent ensures that the appropriate component services are involved and collaborate according to a certain specification. When a composite-service-agent downloads the specification of a composite service, it (i) establishes a context denoted by $\mathcal{C}$-context for the 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. It is needless to say that our description of service deployment is applicable to any type of chronology. 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, a service-agent and an $\mathcal{I}$-context are set up. Afterwards, the service chart diagram of the new service-instance is transferred 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 occurring between service-agents and master-service-agents, exceptions are immediately handled. In addition, while the Web service instance is being performed, the service-agent defines the next Web services after this service instance (Fig. 1). In case there are Web services, the service-agent requests from the composite-service-agent to engage conversations with their respective master-service-agents.

The aforementioned description of the agentification approach presents potential advantages for managing services all along their life-cycle, from preparation to deployment and recomposition if needed. The most prominent advantage is the concurrency existing between the composition and execution of the services. While service-agents are in charge of executing the Web service instances, composite-service-agents engage at the same time conversations with master-service-agents to ensure that the next Web services are got ready for execution.

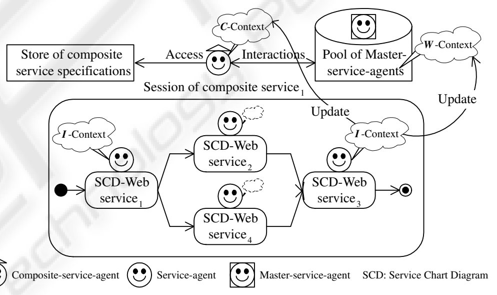

Figure 2: Agents deployment for Web services composition

Fig. 2 represents an execution session of the composite service $CS_1$. It has $4$ primitive component services. Each service instance is associated with a service chart diagram. The clouds in this 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 information occurring between a master-service-agent and a service-agent has already been discussed. In addition to a copy (or a 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 the following 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 Value-added of and 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. 2), three types of services were considered: composite service, Web service, and Web service instance. Each service has been attached to a 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. Furthermore, details on a $\mathcal{W}$-context are used for updating a $\mathcal{C}$-context. We are using *Tuple Spaces* to implement the update operations between contexts (Ahuja et al., 1986). However, to keep the paper self-contained these operations are not discussed[1]. In what follows, we highlight the structure of each context and discuss the value-added of contexts to the composition of Web services.

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.

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

In the beginning of Section 3 we pointed out that our focus is on service-centric context. This is shown with the $\mathcal{W}$-context of a Web service that connects $\mathcal{I}$-contexts and $\mathcal{C}$-contexts. A service-centric context promotes service adaptability, availability, and on-the-fly composition. These requirements are met in our agentification approach. A composite service may have to adapt its list of component Web services because of the availability of certain components. Availability has been illustrated with two cases: (i) a service is either mandatory or optional, and (ii) the maximum number of services instances that can be created with regard to the current number of service instances that are running. Since Web services are instantiated on a request-basis, this means that a composition of type on-the-fly is supported.

Since a service-centric context is adopted, we see a $\mathcal{W}$-context of a Web service along three interconnected perspectives (Fig. 3): instance, execution, and time. Rationale of each perspective is as follows.

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

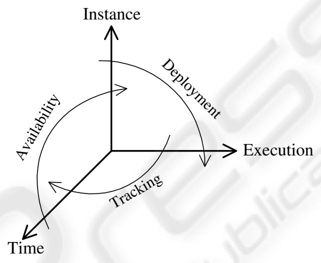

Figure 3: Perspective-based representation of context

Fig. 3 shows that instance, execution, and time perspectives are all interconnected. First, deployment denotes the connection between instance and execution perspectives, which reflects the instantiation of Web services for the needs of execution. Second, tracking denotes the connection between execution and time perspectives, which 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, which reflects the continuous verification the Web services perform before these Web services commit additional service instances for a certain period of time.

The use of context ensures that the requirements and constraints of the services to participate in a composition are considered. While current Web services composition approaches rely on different selection criteria such as execution cost, and reliability, we deem appropriate to integrate the context of services into this composition. Doulkeridis et al. also back the importance of including context during Web services composition (Doulkeridis et al., 2003). We have shown that services take part in a composition based on their availability 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 at any time to know what happened, what is happening with a Web service and all its respective instances. Predicting what will happen to a service could also be feasible in case the previous contexts (i.e., what happened with a service) are stored.

[1]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, 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.

In (Kouadri Mostéfaoui, 2003), Kouadri Mostéfaoui makes a reference to different types of context: user, computing, time, physical, 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.

## 3.3 Conversations and composition

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 (Chakraborty and Joshi, 2001), the selection of the component services to constitute 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 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 the service chart diagram of this instance. The purpose is to verify if extra Web services are needed. If yes, 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. 4 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 Fig. 4, 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. Further, service-agent$_i$ and the composite-service-agent take each a conversation state. In these conversation states, activities to ask 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 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 corresponds to the Web service instances that have already completed their execution (Web services$_{1,\, \ldots,\, i-1}$). The second part corresponds to the Web service instance that is now being executed (Web service instance$_i$). 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$_{j,\, \ldots,\, n}$). Initially, master-service-agent$_j$ is in a monitoring mode; 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 options: decline the request, delay its making decision, or accept the request.

**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$_{k,\,(k \neq j)}$ (i.e., a substitute) to join the composite service (1).
- Otherwise (i.e., 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 b.** Master-service-agent$_j$ of Web service$_j$ cannot make a decision before the response deadline that the composite-service-agent has fixed. Thus, master-service-agent$_j$ requests from the composite-service-agent to extend the deadline (2.2). The composite-service-agent has two alternatives based on 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 master-service-agent$_j$ has requested; this means that the composite-service-agent has to start again conversing with another master-service-agent$_{k,\,(k \neq j)}$ (**Option a**).
- Accept to extend the deadline as master-service-agent$_j$ has requested; this means that master-service-agent$_j$ will get notified about the acceptance of the composite-service-agent (2.2.1).

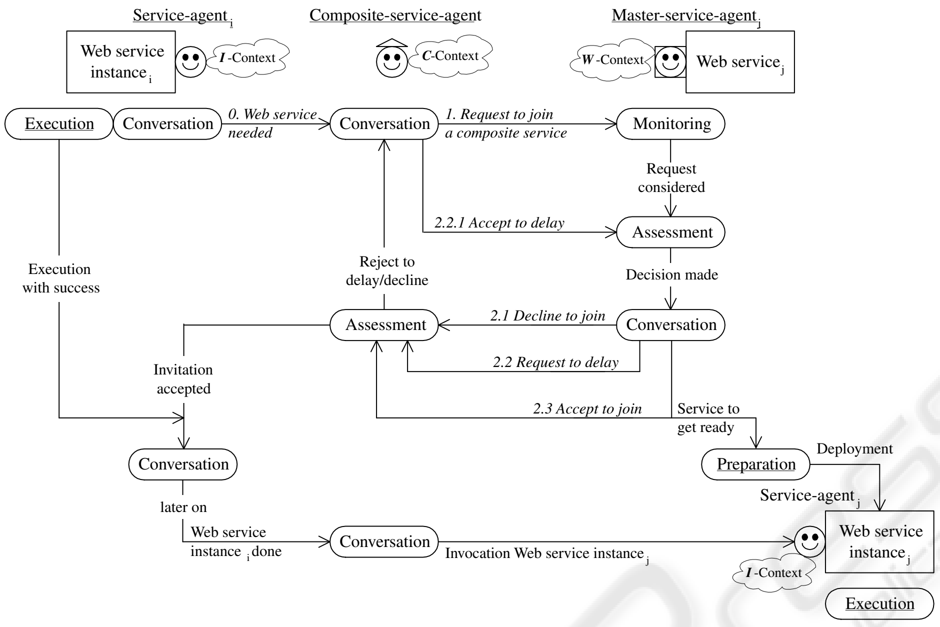

Figure 4: Conversation diagram between agents

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 such as additional instances of Web service$_j$ cannot be committed before other instances complete their execution.

**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 (Ludwig et al., 2002). 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 terminated, service-agent$_i$ informs the composite-service-agent about that. According to the agreement that was established in **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$_j$. It should be noted that the content of agreements is beyond this paper's scope.

## 4 PROTOTYPE IMPLEMENTATION

Based on the approach introduced in Section 3, we are in the process of implementing a prototype. Its architectural overview is depicted in Fig. 5. In what follows we outline the functionality of the major classes of the prototype:

- Service chart diagram: is used to specify Web services and Web Services instances.
- State chart diagram: is used to specify the composite services. It is an aggregation of the different service chart diagrams that correspond to the Web services participating in the composition.
- Agents: the agents set of classes (service agent, master service agent, and composite service agent) is used to specify the agents of the prototype. The service agent and the composite service agent classes are associated with the Web service instance and the composite service classes respectively. Each agent has a label, type, and location (platform where it resides).
- Context: the context set of classes ($\mathcal{I}$-context, $\mathcal{W}$-context, $\mathcal{C}$-context) is responsible for gathering, processing and parsing contextual information. It translates the information parsed into XML documents and attaches each context to its respective agent (service agent, master service agent, and composite service agent).
- Conversation: is the central unit for managing and maintaining conversations between the differ-

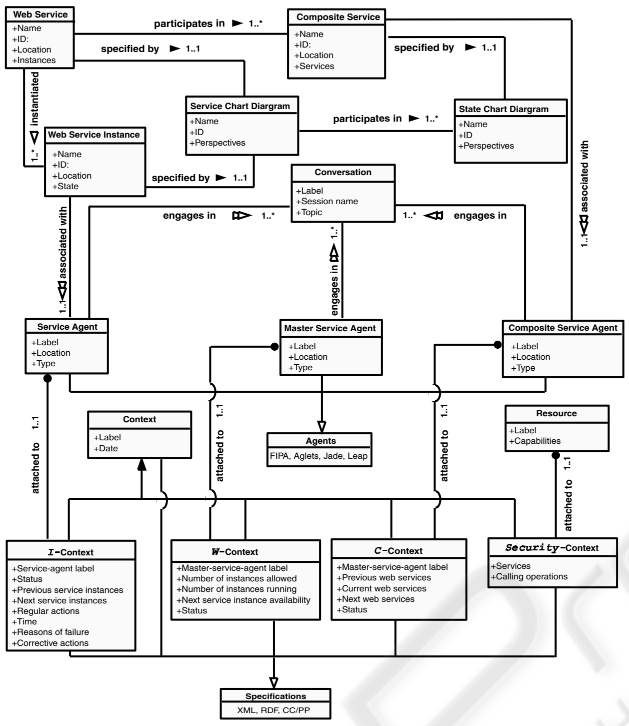

Figure 5: Class model of agent-based deployment

ent agents of the system. Conversation sessions are represented as XML documents.

## 5 RELATED WORK

There are several research initiatives in the field of Web services composition. Besides the traditional selection criteria that are used for composition (e.g., execution cost and 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 created from a Web service definitely delays the development of a composite service. Furthermore, being aware of when these instances will become available helps in adjusting 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 service chart diagram of Fig. 1, the flow perspective illustrates the pre- and post-component services of a composite service. These component services can be either optional or mandatory. These types of service are similar to what Berfield et al. call in (Berfield et al., 2002) by vital and nonvital services for a workflow application. Another use of previous and next services of the flow perspective is related to pre- and post-conditions. 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.

The Web Services Conversation Language (WSCL) is a sample of the initiatives on conversations in the field of Web services (Beringer et al., 2001). This language describes the structures of documents that a Web service expects to receive and produce, as well as the order in which the interchange of these documents is supposed to occur. While the WSCL focuses on the specification of the operations that Web services support, our conversations focus on the mechanisms of establishing composite services.

Finally, another use of conversations for Web services enables us dealing with the composability issue of Web services as Medjahed et al. discuss in (Medjahed et al., 2003). 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. Agents engage conversations for agreeing on what to exchange, how to exchange, when to exchange, and what to expect from an exchange.

## 6 CONCLUSION AND FUTURE WORK

In this paper, we presented our approach for composing Web services using software agent and context. Several types of agents have been considered namely composite-service-agents for composite services, master-service-agents for Web services, and service-agents for 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. 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 conversations to decide if service instances can be created and annexed to a composite service. Such a decision is based on several factors among them 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 ongoing work is decomposed into several thrusts. A security is one them as it is important to make sure that services do not misuse the computing resources on which they will be performed. Another thrust consists of composite service adaptability by allowing composite-service-agents to carry out some run-time modifications by for instance adding new component services, removing certain component services, or replacing certain component services. Therefore, the assessment of the effects of adaptation is deemed appropriate. Another thrust consists of refining the update operations that occur between contexts. We already pointed out the support of Tuple Spaces to such operations.

## REFERENCES

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

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

Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., and Mecella, M. (2003). A Foundational Vision for e-Services. In *Proceedings of The Workshop on Web Services, e-Business, and the Semantic Web (WES'2003) held in conjunction with The 15th Conference On Advanced Information Systems Engineering (CAiSE'2003)*, Klagenfurt/Velden, Austria.

Berfield, A., Chrysanthis, P. K., Tsamardinos, I., Pollack, M. E., and Banerjee, S. (2002). A Scheme for Integration E-Services in Establishing Virtual Enterprises. In *Proceedings of The Twelfth International Workshop on Research Issues in Data Engineering: Engineering e-Commerce/e-Business Systems (RIDE'02)*, San Jose, USA.

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

Brézillon, P. (2003). Focusing on Context in Human-Centered Computing. *IEEE Intelligent Systems*, 18(3).

Chakraborty, D. and Joshi, A. (2001). Dynamic Service Composition: State-of-the-Art and Research Directions. Technical report, TR-CS-01-19, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Maryland, USA.

Curbera, F., Khalaf, R., Mukhi, N., Tai, S., and Weerawarana, S. (2003). The Next Step in Web Services. *Communications of the ACM*, 46(10).

Doulkeridis, C., Valavanis, E., and Vazirgiannis, M. (2003). Towards a Context-Aware Service Directory. In *Proceedings of The 4th Workshop on Technologies for E-Services (TES'03) held in conjunction with The 29th International Conference on Very Large Data Bases (VLDB'2003)*, Berlin, Germany.

Harel, D. and Naamad, A. (1996). The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4).

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

Kouadri Mostéfaoui, S. (2003). Towards a Context-Oriented Services Discovery and Composition Framework. In *Proceedings of AI Moves to IA: Workshop on Artificial Intelligence, Information Access, and Mobile Computing held in conjonction with the 18th International Joint Conference on Artificial Intelligence (IJCAI'2003)*, Acapulco, Mexico.

Ludwig, H., Keller, A., Dah, A., and King, R. (2002). A Service Level Agreement Language for Dynamic Electronic Services. In *Proceedings of the 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information System (WECWIS'2002)*, Newport Beach, California, USA.

Maamar, Z., Benatallah, B., and Mansoor, W. (2003). Service Chart Diagrams - Description & Application. In *Proceedings of The Twelfth International World Wide Web Conference (WWW'2003)*, Budapest, Hungary.

Maamar, Z. and Mansoor, W. (2003). Design and Development of a Software Agent-based and Mobile Service-oriented Environment. *e-Service Journal, Indiana University Press*, 2(3).

Medjahed, B., Rezgui, A., Bouguettaya, A., and Ouzzani, M. (2003). Infrastructure for E-Government Web Services. *IEEE Internet Computing*, 7(1).

Roman, M. and Campbell, R. H. (2002). 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.