Software Agents, Conversations, and Context for Web Services Composition

Z. Maamar[α], S. Kouadri Mostéfaoui[β], H. Yahyaoui[γ], and W. J. van den Heuvel[δ]

[α]Zayed University, [β]University of Fribourg, [γ]Laval University, and [δ]Tilburg University

Abstract

This paper presents a software agent-based and context-oriented approach for Web services composition. A Web service is an accessible application that other applications and humans can discover and trigger to satisfy various needs such as travel booking. Composition operations are outsourced to software agents, which engage in conversations with their peers to agree on the Web services that will take part in composition. In these conversations, the context surrounding the Web services is considered as well.

Web services are nowadays becoming a major technology for deploying automated business interactions between distributed and heterogeneous applications. A Web service is a component that can be assembled and re-used in a distributed, Internet-based environment. Benatallah et al. associate the following properties with a Web service [4]: (i) independent as much as possible from specific platforms and computing paradigms; (ii) developed mainly for inter-organizational situations; and (iii) easily composable so that developing complex adapters for the needs of composition are not required.

Composing Web services (also called services in the rest of this paper) rather than accessing a single service is essential and provides better 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 [5]. 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 [8].

Entrusting the composition of Web services to software agents sheds the light on the presence of several issues such as 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 actions to avoid conflicts, and what back-up strategies handle the execution exceptions of Web services? To address some of these issues, software agents need to be aware of the context in which the composition and execution of the Web services are happening. Context is the information that characterizes the interaction between humans, applications, and the surrounding environment [6].

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. Ardisson et al. argue that current Web services standards support simple interactions and are mostly structured as question-answer pairs [2]. 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 [3], who noticed that the despite 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 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 [11]. 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 such as: (i) Web services act as passive components rather than active components that can be embedded with context-awareness mechanisms, (ii) existing approaches for Web services composition (e.g., BPEL4WS) typically facilitate orchestration only,

while neglecting information about the context of users and services, and (iii) lack of appropriate techniques for modeling and specifying conversations between Web services. This paper discusses how *software agents, conversations, and context constitute a cornerstone for Web services composition*. In the next section, the approach that agentifies the composition of Web services is presented. This is followed by a discussion on the value-added of contexts and conversations to the agentification approach. Then, the automation of conversation and context management is outlined. Finally, concluding remarks and some activities for future work are presented.

Agentifying Web Services Composition

Various types of input sources such as service and user can feed context with details. Roman and Campbell observe in [12] 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' personal preferences. This paper adopts a service-centric context to promote applications that (i) allow adaptability of services, (ii) deal with availability of services, and (iii) support dynamic composition of services. In the following, the focus is on the context of services.

The agentification of Web services composition supports determining the appropriate types and roles of agents, which ensure the deployment of the composition specification. Three types of agents are put forward. They are referred to as composite-service-agent, master-service-agent, and service-agent. Because a Web service is considered as a component, it is instantiated each time it is being called to take part in a composition. Before the instantiation occurs, several elements associated with the Web service are checked. These elements constitute a part of the context, denoted by $\mathcal{W}$-context, of the Web service, and they are as follows: (i) number of service instances currently running *vs.* maximum number of service instances that can be simultaneously run, (ii) execution status of each service instance deployed, and (iii) request time of the service instance *vs.* availability time of the service instance.

The role of the master-service-agent is to track the multiple Web service instances that are obtained from a Web service (i.e., object-class principle). Master-service-agents, Web services,

and $\mathcal{W}$-contexts are all stored in a pool (Fig. 1). A master-service-agent processes the instantiation demands that are submitted to its respective Web service. These demands originate from the composite-service-agents that identify the composite services to set-up. 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 for joining a composite service can be rejected because of a period of non-availability or an overloaded status of a Web service.

To be aware of the running instances of a Web service so its $\mathcal{W}$-context is updated, the master-service-agent associates each service instance it creates with a service-agent and $\mathcal{I}$-context. The service-agent manages the specification of the service instance (specification based on service chart diagram [10]) and its $\mathcal{I}$-context. Using this specification, the service-agent knows the states that the service instance takes, 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 interaction. The content of $\mathcal{I}$-contexts feeds the content of $\mathcal{W}$-contexts with various details such as (i) what is the execution status of a service instance, (ii) when is the execution of a service instance supposed to resume in case it was suspended, (iii) what are the reasons of suspending the execution, and (iv) when is the execution of a service instance expected to complete?

With regard to composite-service-agents, their role is to trigger the specification of the composite services and monitor the deployment of this specification [10]. A composite-service-agent ensures that the appropriate component services are involved and collaborating according to a specific specification. When a composite-service-agent downloads the specification of a composite service, it (i) establishes a context, denoted by $\mathcal{C}$-context, for the composite service, and (ii) identifies the first Web services to be triggered. When the first Web services of the composite service are known, the composite-service-agent asks their respective master-service-agents for service instantiation. If a master-service-agent agrees on the instantiation after checking the $\mathcal{W}$-context, a service-agent and $\mathcal{I}$-context are set up. Afterwards, the specification of the new service-instance is transmitted to the service-agent. This service-agent initiates the execution

of the service instance and keeps the master-service-agent informed about the execution status. Because of the regular notifications occurring between service-agents and master-service-agents, exceptions are immediately handled and corrective actions are carried-out on time. In addition, whilst the Web service instance is being performed, the service-agent identifies the Web services that are due for execution after this service instance. In case there are Web services due for execution, the service-agent requests from the composite-service-agent to engage in conversations with their respective master-service-agents.

Fig. 1 represents an execution session of the composite service CS_1, which has 4 primitive component-services (i.e., 4 service instances): $service_{\{1,2,3,4\}}$. 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 service-agent has already been discussed in the previous paragraphs. 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 agents receive extra details from service-agents including (i) the next services to be called for execution, and (ii) the type of these services whether mandatory or optional.

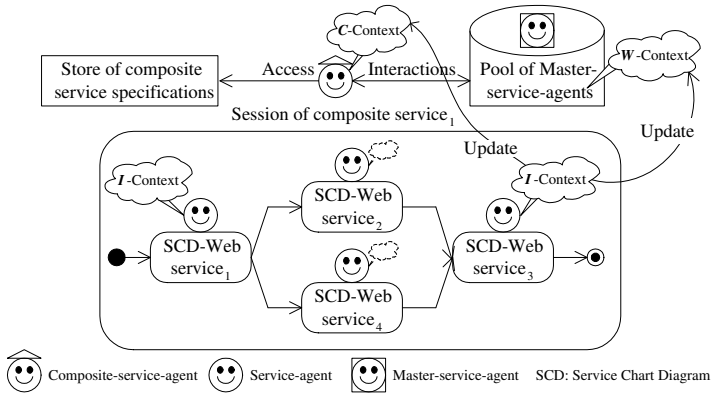

Figure 1: Software agents and contexts for Web services composition

Modeling $\mathcal{I}/\mathcal{W}/\mathcal{C}$-contexts

Besides the three types of agents that Fig. 1 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 most-grained content, whereas $\mathcal{C}$-context has the least-grained content. The $\mathcal{W}$-context is in between the two contexts. Details on $\mathcal{I}$-context update $\mathcal{W}$-context, and details on $\mathcal{W}$-context update $\mathcal{C}$-context. We are using Tuple Spaces to implement the update operations between contexts [1], but this outside this paper's scope[1]. To keep the paper self-contained, only the arguments of $\mathcal{I}$-context are detailed.

The $\mathcal{I}$-context of a Web service instance consists of the following parameters (Tab. 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 on the $\mathcal{I}$-contexts of its respective Web service instances and consists of the following parameters: label, master-service-agent label, number of instances allowed, number of instances currently running, next service instance availability, status per service instance, and date.

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

It was pointed out that a service-centric context is adopted in this paper. This has been shown with the $\mathcal{W}$-context of a Web service, which is 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 Web services composition of Fig. 1 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 has been illustrated with the maximum

[1] A sample of a control tuple illustrating an update between contexts: **modified($\mathcal{I}$-context i, WebService-Instance 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.

Table 1: Parameters of an $\mathcal{I}$-context and their description

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 are behind the failure of the execution of the service instance.
Corrective actions: illustrates the actions the service instance performs because the execution has failed.
Date: identifies the time of updating the parameters above.

number of services instances that can be created *vs.* 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 dynamic composition is supported.

Because of the aforementioned requirements, $\mathcal{W}$-context is seen along three interconnected axes (Fig. 2):

1. Instance axis: 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 axis: is about meeting the computing resource requirements of service instances, tracking their execution status, and avoiding conflicts on these computing resources.

3. Time axis: 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. 2, instance, execution, and time axes are connected to each other. First, deployment

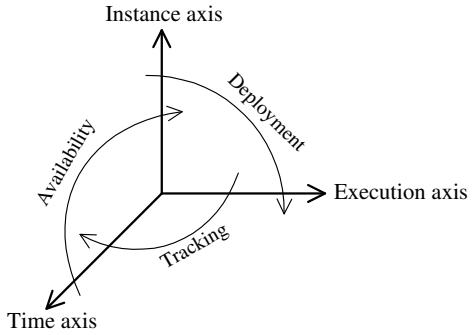

Figure 2: Axes of context

denotes the connection between instance and execution axes and reflects the Web services that are instantiated for execution. Second, tracking denotes the connection between execution and time axes 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 axes and reflects the continuous verification that the Web services perform before they commit additional service instances over a certain period of time.

Managing Conversations

In a reactive Web services composition [7] such as the one that features the agentification approach of Fig. 1, the selection of the component services of a composite service is carried out on-the-fly. We outsource the selection operations to composite-service-agents that engage in conversations with the respective master-service-agents of the Web services. In these conversations, master-service-agents decide if their respective Web services will join the composition after assessing the $\mathcal{W}$-contexts. If yes, Web service instances, service-agents, and $\mathcal{I}$-contexts are deployed.

When a Web service instance is under execution, its service-agent checks the specification of the service instance. The objective is to verify if extra 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 the extra Web services. These conversations have two aims: (i) invite master-service-agents and their Web service to participate in the composition; and (ii) ensure that the Web services are ready for instantiation following their invitation acceptance.

Fig. 3 depicts a conversation diagram between a service-agent, a composite-service-agent, and a master-service-agent. The composite-service-agent identifies a composite service of n component Web services$_{(1, \cdots, i, j, \cdots, n)}$. In this figure, rounded rectangles correspond to states (states with underlined labels belong to Web service instances, and other states belong to agents), italic sentences correspond to conversations, and numbers correspond to 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, actions to request the participation of the next Web services (i.e., Web service$_j$) are performed.

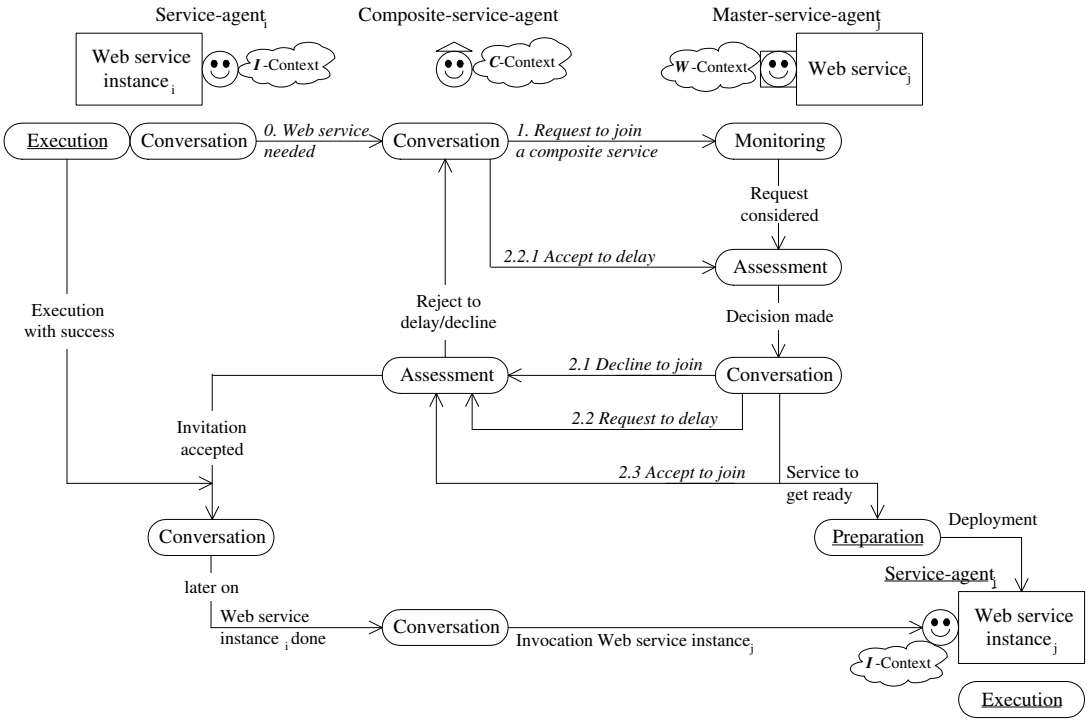

Figure 3: Conversation-based interaction diagram between agents

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). Web service$_j$ (0) is an element of the composite service that is under preparation. It should be noted that a composite service is decomposed into three parts. The first part corresponds to the Web service instances that have already completed their execution with success (Web services$_{1,\ ...,\ i-1}$). The second part corresponds to the Web service instance(s) that is now under execution (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$_{j,\ ...,\ n}$). Initially, master-service-agent$_j$ is in a monitoring mode in which it tracks the instances of Web service$_j$ that currently participates 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 the decision making, 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 notification (2.1). Because it is assumed that a component service can be either mandatory or optional in a composite service, the composite-service-agent needs 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).

- Otherwise (i.e., Web service$_j$ is mandatory), the composite-service-agent engages in further conversations with master-service-agent$_j$ checking 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 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 for extending 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 per master-service-agent$_j$'s request. 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 per master-service-agent$_j$'s request. This means that master-service-agent$_j$ will get 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. A master-service-agent may request an extension of deadline since additional instances of a Web service cannot be committed before other running 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 [9]. 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 finished, 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$_j$.

Conversation and Context Management Automation

We are in the process of implementing a prototype that supports conversation and context management automation for Web services composition through the use of software agents. Developing an infrastructure to support such automation is clearly a complex endeavor that will likely require further progress in both research and standardization. As a first step, the prototype's architectural overview is depicted in this paper (Fig. 4). In what follows we outline the functionality of the major classes:

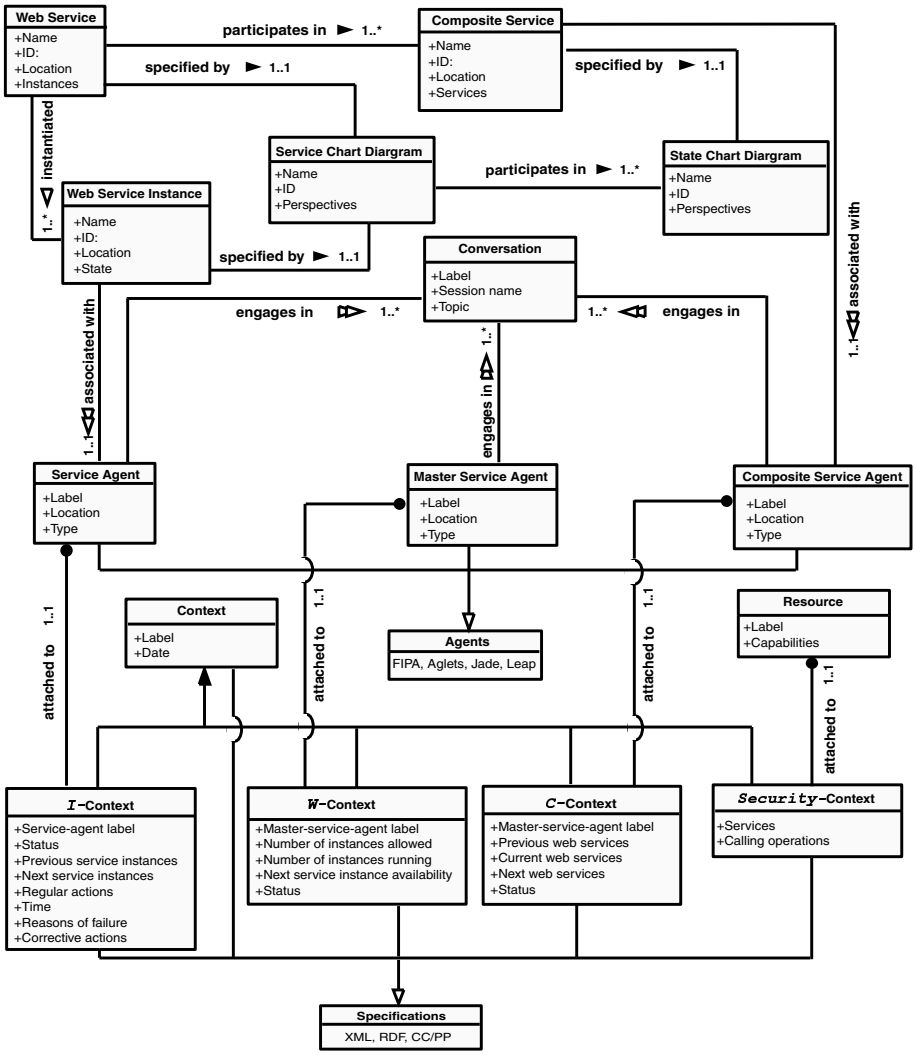

Figure 4: Prototypical class model

- Service chart diagram: is used to specify Web services and Web services instances.
- State chart diagram: is used to specify composite services. A state chart diagram is an aggregation of the service chart diagrams of the component Web services of a composite service.
- Agents: is used to specify the agents (service-agent, master-service-agent, and composite-service-agent) of the prototype. Each agent has a label, type, and location that is the 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 the appropriate agent (service-agent, master-service-agent, and composite-service-agent).
- Conversation: is the central unit for managing and maintaining conversations between the different agents of the system. Conversation are represented as XML documents.

Conclusions

Our approach builds upon software agent, conversation, and context concepts is one step towards the dynamic management of Web services composition. Several types of agents are put forward namely composite-service-agents associated with composite services, master-service-agents associated with Web services, and service-agents associated with Web service instances. The different agents are aware of the context of their respective services in the objective to devise composite services on-the-fly. Conversations between agents have also featured the composition of Web services. Before Web service instances are deployed, agents engage in conversations to decide if service instances will be annexed to a composite service. Such decisions are based on several factors such as the maximum number of service instances that can deployed at the same time and the availability of these instances over a certain period of time.

Our future work is about adaptability of Web services 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.

References

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

[2] L. Ardissono, A. Goy, and G. Petrone. Enabling Conversations with Web Services. In *Proceedings of the Second International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS'2003)*, Melbourne, Australia, 2003.

[3] B. Benatallah, F. Casati, and F. Toumani. Web Service Conversation Modeling, A Cornerstone for E-Business Automation. *IEEE Internet Computing*, 8(1), January/February 2004.

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

[5] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. 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, 2003.

[6] P. Brézillon. Focusing on Context in Human-Centered Computing. *IEEE Intelligent Systems*, 18(3), May/June 2003.

[7] D. Chakraborty and A. Joshi. 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, 2001.

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

[9] H. Ludwig, A. Keller, A. Dah, and R. King. 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, 2002.

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

[11] M. Papazoglou and D. Georgakopoulos. Introduction to the Special Issue on Service-Oriented Computing . *Communications of the ACM*, 46(10), October 2003.

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