Service Chart Diagrams - Description & Application

Zakaria Maamar
College of Information Systems
Zayed University
P.O. Box 19282, Dubai, U.A.E
zakaria.maamar@zu.ac.ae

Boualem Benatallah
School of Computer Science & Engineering
The University of New South Wales
Sydney NSW 2052, Australia
boualem@cse.unsw.edu.au

Wathiq Mansoor
College of Information Systems
Zayed University
P.O. Box 19282, Dubai, U.A.E
wathiq.mansoor@zu.ac.ae

ABSTRACT

This paper presents an approach for the design and development of service-driven applications. These applications rely on the collaboration of multiple services that businesses offer to the external community. To ensure that the collaboration of services takes place effectively, service chart diagrams are proposed as a specification technique. These diagrams leverage the traditional state chart diagrams of UML. Furthermore, in service chart diagrams it is advocated that services do not invoke each other. However, they engage conversations before committing themselves to a composition process of services.

Categories and Subject Descriptors

H.4.5 [**Information Systems**]: service-oriented applications.

General Terms

Modeling, Diagrams.

Keywords

state, service, diagram, conversation.

1. INTRODUCTION

With the rapid development of information and communication technologies, users are becoming more and more demanding on businesses to provide them with relevant and up-to-date information. Furthermore, needs of users continue to grow and change becoming overtime more complex to satisfy. Needs vary from basic ones such as weather forecast of city X, to complex ones such as stock quotation of business Y and its direct competitors since last fall. This situation appeals for advanced approaches and tools to support software designers and developers in their work. Service-driven applications are deemed appropriate to deal with the aforementioned situations [4, 15].

The following example motivates the importance of service-driven applications. Paul is planning for his vacation; (i) he wants to book a domestic flight and an accommodation; (ii) he, also, wants to find some attractions for visit; (iii) he would like to rent a car if the location of the major attraction is far from the location of the booked accommodation. To handle Paul's request, the collaboration of multiple services[1] is required. These services are: flight reservation, hotel booking, attraction search, and car rental. All the services have to be connected according to a specific flow of control. First, flight reservation is completed. Then, hotel booking and attraction searching are triggered concurrently. Each business being involved in the vacation scenario provides its services that may have to collaborate with other services if needed and *vice-versa*. Rather than just being invoked through their application programming interfaces, we advocate that services should be given the opportunity to engage conversations if they wish. There is an increasing trend towards run-time composition, where services choose dynamically with whom they would like to trade. The composition of services from multiple origins calls for new design approaches and representation formalisms. This is motivated by the following elements:

- Distribution: businesses that provide services are most of the time spread across multiple locations. These businesses have to get together in order to be aware of their respective capabilities and constraints.
- Heterogeneity: services are developed independently from each other with diverse technologies. It is agreed that programmers who implement services are unlikely to collaborate with each other during development.
- Autonomy: services carry out operations without considering the operations of other services. Services are considered as self-contained components.

To undertake work on service composition, we suggest as a part of the solution the use of service chart diagrams as an extension to the state chart diagrams of UML [9]. A service chart diagram identifies the context surrounding the execution of a service in terms of who provides a service (organization), with whom a service engages conversations (flow), with what a service contributes (information), and where a service contributes (location of execution). We strengthen the fact that services have to be able to decide with whom to collaborate, what kind of support they offer/request to/from other services, and what "visible" parts of behavior (from private to public, and *vice-versa*) to exhibit. Processes that implement services illustrate that behavior.

Section *2* presents briefly UML state chart diagrams. Section *3* provides basic definitions about services. Section *4* introduces service chart diagrams in terms of rationale and basics. For illustration purposes, Section *5* applies service chart diagrams to a running example. Section *6* presents related work. Finally, Section *7* concludes the paper.

[1] Services are also known as Web services.

WWW2003, May 20–24, 2003, Budapest, Hungary.
ISBN 963-311-355-5.

2. STATE CHART DIAGRAM

A state chart diagram is one of the several diagrams that UML integrates [9]. It is a graphical representation of a state machine that visualizes how and under what circumstances a modelled element (e.g., a class, a system, or a business process) changes its states. Furthermore, a state chart diagram is used for showing which activities are executed as a result of the occurrence of events. Mainly, a state chart diagram displays the states that an object takes during its life in response to received stimuli. Responses correspond to the execution of activities.

3. WEB SERVICES

Regardless of its type (E-service or M-service), a Web service is a set of ordered operations to perform according to certain inputs. The order can be sequential or concurrent. Samples of Web services are currency conversion and cinema ticket purchasing. Potential users have to know how to request a service for execution. However, users do not have to know neither how to operate the service nor how the service operates or is operated. [13] distinguishes between two types of services: E-services and M-services (M for Mobile).

In this paper, a composite services consists of several component services whether composite services or services. Multiple technologies are associated with the success of Web services: WSDL, UDDI, and SOAP. These technologies aim at supporting the definition of Web services, their advertisement, and their binding for triggering purposes [10].

a. E-services

An E-service is a component that an organization provides in order to be assembled and re-used in a distributed, Internet-based environment. A component is an E-service if it is [5]: 1) independent as much as possible from specific platforms and computing paradigms; 2) developed mainly for inter-organizational situations rather than for intra-organizational situations; and 3) easily composable, its assembling with other E-services does not require the development of complex adapters.

b. M-services

Two definitions are associated with an M-service. The ”weak” definition is to trigger remotely an E-service from a mobile device for execution. In that case, the E-service is an M-service. The ”strong” definition is to transfer wirelessly an E-service from its hosting site to a mobile device where its execution takes place. In that case, the E-service is an M-service that meets the following requirements [14]: 1) transportable through wireless networks; 2) composable with other M-services; 3) adaptable according to the computing features of mobile devices; and finally 4) runnable on mobile devices. In this paper, we focus on the M-services that comply with the ”strong” definition.

Figure 1 illustrates a snapshot of a mobile service running on a cell-phone. The service provides information to tourists visiting Dubai. Upon request of tourists, the service is downloaded to their mobile devices.

c. E-services vs. *M-services*

The difference between an E-service and M-service occurs at two levels. The first level concerns the communication channel, i.e., wired *vs.* wireless. The second level concerns the location of where the processing of the service occurs: server side on a fixed platform for an E-service *vs.* user side on a mobile platform for an M-service.

Figure 1: Mockup of tourist mobile-book

4. DESCRIPTION OF SERVICE CHART DIAGRAMS

4.1 Rationale

The current requirements of designing applications call for new representation formalisms and design approaches. Designers are faced with multiple obstacles that need to be dealt with quickly and efficiently. An example of these challenges consists of maintaining the coherence of the content of a database of an e-commerce site. Thousands of customers from over the world may initiate at the same time multiple purchase requests for the same product but with different selection criteria. This scenario puts forward new demands not only on support and delivery information technology, but also on the way business processes have to be designed, developed, and maintained. Another example of challenges that face designers consists of dealing with the issues and obstacles of mobile applications. Designers and programmers are put on the front line of satisfying the promise of businesses and service providers of delivering Internet content to users of mobile devices. Service-driven applications seem to be one of the relevant technologies that could help in addressing the aforementioned challenges and obstacles. Services, rather than code, are emerging as the key artifacts of software design and development, raising therefore the level of abstraction. Service chart diagrams are among the pillars on top of which the trend of service-driven applications can be built. Services in such diagrams are not only invoked for their operations but rather asked to establish conversations before joining any service composition process.

4.2 Concepts & Formalisms

A service-driven application is a process that connects multiple services. The connection is an outcome of the conversations that take place between services. Different parameters are included in those conversations, e.g., workload and location of a service. Having several services enables the consideration of multiple businesses that offer those services. Quality and execution cost of a service are among the selection criteria that affect the shape of any composite service in terms of number of services to be considered and execution chronology of these services.

Service chart diagrams are based on UML state chart diagrams. This time, the emphasize is on the context surrounding the execution of a service rather than on the states that a service takes.

Services are represented from five perspectives. Besides the state perspective that includes the states of a service (see Section 2), the flow perspective corresponds to the execution chronology of the connected services. Here, the flow is conversation-based. The organization perspective identifies the business that supplies a service. The information perspective identifies the data that are exchanged between services. These data are identified during conversations and packaged into XML documents. A service that completes its execution may have to leave certain data to the next services that are due for execution, so they could resume pending operations. Finally, the location perspective identifies the current site of a service. A service can be in one of the two sites: 1) business site waiting to be selected and inserted into a composite service; or 2) execution site under performance. According to Section 3, an execution site corresponds to a business site for an E-service or client site for an M-service.

A service chart diagram enhances a state chart diagram with details obtained from the various perspectives of Figure 2. Therefore, the service chart diagram of a composite service consists of connecting the service chart diagrams of all the services that constitute that composite service. Table 1 summarizes the three layers that constitute a service chart diagram. Interesting is layer *2* which contains the states that a service takes. These states constitute themselves a state chart diagram that is wrapped in the different perspectives. It should be noted that the states of layer *2* integrate both normal and ad-hoc operating of a service. Ad-hoc operating corresponds to the exceptional cases that may occur, e.g., execution failure. Thus, back-up states and also, extra services can be requested to deal with the exceptional cases.

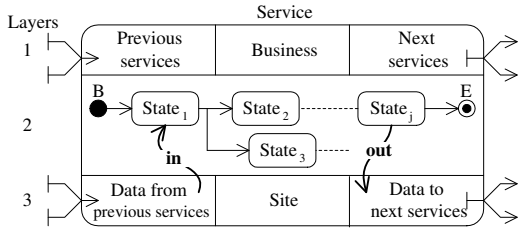

Figure 2: Representation of a service chart diagram

In Figure 2, the three-layer representation of a service chart diagram offers two major advantages. First, the layers allow a clear distinction between the components that contribute to the specification of a service. If a component has to be modified, the modification impact on that specification will be limited. Second, the layers offer a connection between the services at three levels of abstraction. These levels are data, state, and service.

Table 1: Layers of a service chart diagram

Layer	Field	Perspective
1	Previous services Next services	Flow
	Business	Organization
2	States	State
3	Data from previous services Data for next services	Information
	Site	Location

In Table 1, next services field represents the list of services that are due for execution after a service completes its execution. This list is an expression that combines services over logical operators (AND, OR). For instance, services that are connected with an AND operator have to be triggered in a concurrent way. A similar description applies to OR operator. Each service that appears in next services field is also annotated with the following elements:

1. The protocol that enables the invocation of the service. SOAP over HTTP is among the protocols that can be used [1].
2. The conditions to check before the service is invoked. The elements of a condition are obtained from the states of the service that is under execution.

5. APPLICATION OF SERVICE CHART DIAGRAMS

In Section 1, the vacation of Paul motivated our discussions on the importance of new design approaches. We pointed out that 4 services were required to handle Paul's request. Figure 3 is a sample of travel planning composite service that will be used in the rest of this paper. In addition to these services, 2 new services are added: driving time calculation that checks the distance between the location of the hotel and the location of the main attraction, and user notification that provides responses to user.

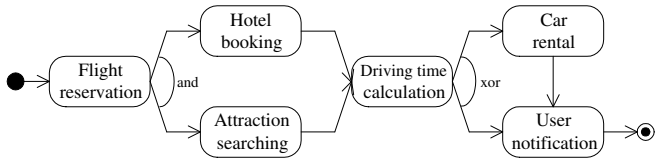

Figure 3: Travel planning composite service

In an open economy market, competition between businesses is a natural practice. To set up a composite service brokering mechanisms such as UDDI have to be made available. The role of such mechanisms is first, to facilitate the search of businesses that offer services and second, to match these services to the submitted requests of users. Despite their importance, brokering and service selection mechanisms do not fall within the scope of this paper.

5.1 Service chart diagrams vs. Sites

According to the location perspective, a service (i.e., an instance) can be in one of the following two sites: business site or execution site. Both types of site influence the content (in term of states) and shape (in term of chronology) of a service chart diagram. In what follows, the conceptual description of the service chart diagram of Figure 2 is applied according to the features of each site. Flight reservation service of Figure 3 is used for illustration purposes.

service takes stand by state waiting, first to be selected among multiple services by the composition process and then, connected to other services. Figure 4 is the service chart diagram of the service in business site. In this diagram, certain fields of Table 2 (e.g., business, next services, and site) are filled with values. In addition, stand by, preparation, and transfer are the states that the service takes. Different activities are undertaken within each state. Transfer state only applies to M-services. Indeed, the execution of M-services takes place in a different site to the business site. Flight reservation service is followed by two services: hotel booking and attraction searching. Both services are triggered in case a flight reservation is confirmed.

service is due for execution. This execution occurs either in the business site for an E-service or client site for an M-service. Figure 5 is the service chart diagram of flight reservation service in execution site. Preparation state only applies to M-services; they need to be checked and installed upon arrival from the business site to the site of user which is a mobile device. Table 3 illustrates how the fields of Table 1 are instantiated according to execution site.

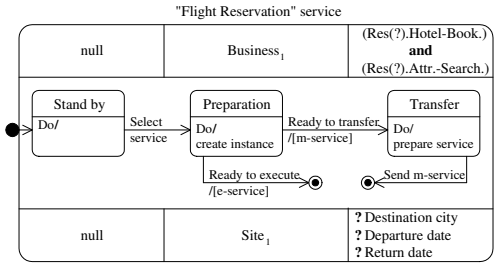

Figure 4: Service chart diagram - Business site

Table 2: State chart diagram details - Business site

Field	Value
Previous services	null
Next services	(Reservation(?).Hotel Booking) **and** (Reservation(?).Attraction Searching)
Business	$Business_1$ (offers the service)
States	Stand by, Preparation, /Transfer
Data from previous services	null
Data for next services	?Destination_city, ?Departure_date, ?Return_date
Site	$Site_1$ (where the service is located now)

After finishing the execution of flight reservation service, the relevant information such as date of departure and date of return are obtained and afterwards, submitted to the next services.

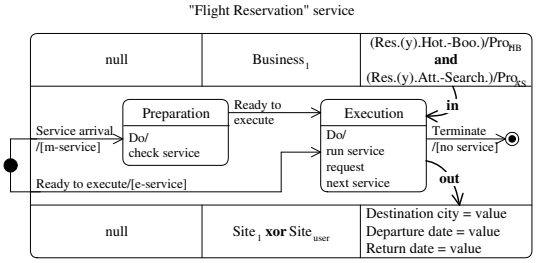

Figure 5: Service chart diagram - Execution site

Note: In Table 3, Protocol$_{HB}$ and Protocol$_{AS}$ correspond respectively to the protocols that trigger Hotel Booking service and Attraction Searching service.

5.2 Conversation-driven composition

In Section 5.1, we pointed out that a service is initially in a selection stage (i.e., business site) and afterwards, enters an execution stage (i.e., execution site). We advocate that services must be able to talk to each other before they decide if to join a composition process, what states they take according to the outcome of conversations, and what activities they perform within these states. Conversations are based on a Conversation Language (CL) and are of different types, e.g., representatives, directives, commissives, and permissives [8]. When services engage conversations, they need a-priori to agree upon the exchange protocol to communicate with each other.

In our research, the use of conversations aims at raising the level of services to the level of autonomous components that are able to make independent decisions [11]. This aids in building composite services at run-time instead of design-time. What is interesting to point out is the concurrency that exists between the selection and

Table 3: State chart diagram details - Execution site

Field	Value
Previous services	null
Next services	(Res.(y).Hotel Booking)/Pro.$_{HB}$ **and** (Res.(y).Attraction Searching)/Pro.$_{AS}$
Business	$Business_1$ (offers the service)
States	/Preparation, Execution
Data from previous services	null
Data for next services	Value(Des._city,Dep._date,Ret._date)
Place	$Site_1 \oplus Site_{user}$

execution stages of a service in an execution site. When a service is under execution, it has at the same time to initiate conversations with the services that are due for execution (see next services field). The purpose of these conversations is twofold: invite the services to join the composition process and make sure that the services are ready for execution after they agreed on joining the process. Since service chart diagrams of Figure 4 and Figure 5 do not contain any conversation state, we deemed appropriate to complete these diagrams with the missing states.

a. Business Site

Figure 6 illustrates a light version (i.e., with no perspectives) of the new service chart diagram in business site after introducing the conversation state. The main difference with the service chart diagram of Figure 4 is that now a service can either accept or reject joining a composition process. Without conversations, it was granted that a service will take part to the composition process. A service can turn down an invitation to join a process of composing services for various reasons; e.g., the maximum number of the instances that can be deployed at the same time of that service has been reached.

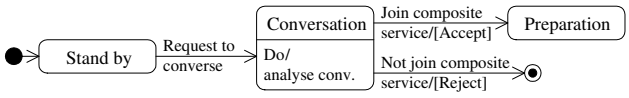

Figure 6: Updated service chart diagram - Business site

b. Execution Site

Figure 7 illustrates a light version of the new state chart diagram in execution site after introducing the conversation state. While a service is being executed, it engages conversations with the next services that are due for execution. It should be noted that execution and conversation states are concurrent.

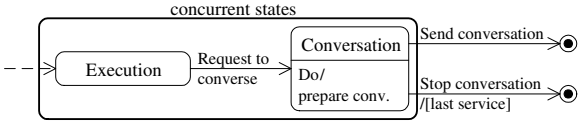

Figure 7: Updated service chart diagram - Execution site

Figure 8 represents a conversation-based interaction diagram between two services of a composite service CS. It includes n component services (service$_{1,\ \cdots,\ i,\ j,\ \cdots,\ n}$). For the sake of simplicity, the services are executed sequentially. In this figure, rounded rectangles correspond to states, italic sentences correspond to conversations, and numbers correspond to the chronology of these con-

versations. Initially, service$_i$ takes two concurrent states; execution state where certain activities are carried out and conversation state where certain activities to select the next services, namely service$_j$, are carried out, too. In what follows, We focus on the conversation state of service$_i$.

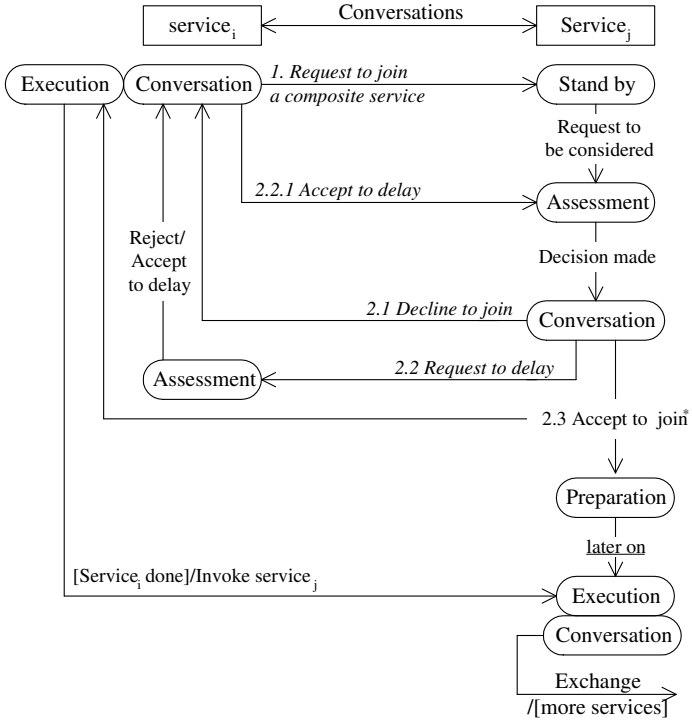

Figure 8: Conversation-based interaction diagram between services

With conversations, our aim is to enable services to make decisions regarding their intention to join a composite service. In Figure 8, the first established conversation consists of sending a request from service$_i$ to service$_j$ to join the composite service (1). This composite service is decomposed into three segments. The first segment corresponds to the services that have completed their execution (service$_{1,\ \cdots,\ i-1}$). The second segment corresponds to the service that is currently under execution (service$_i$). Finally, the third segment corresponds to the composite service that is under preparation (service$_{j,\ \cdots,\ n}$). Service$_j$ is in stand by mode waiting to receive invitations of joining a composition process. When it receives an invitation, service$_j$ enters the assessment state. Within that state, service$_j$ considers its constraints and makes a decision whether to decline the invitation, to delay its making decision, or to accept the invitation. Samples of constraints could be the number of active requests invoking a service simultaneously and the period of no-availability of a service for some maintenance work. Table 4 illustrates a conversation message that has several attributes among them the identifier and subject of conversation.

Case a. - In case service$_j$ declines the invitation, a conversation message is sent back from service$_j$ to service$_i$ for notification (2.1). Thus, service$_i$ enters again the conversation state, asking another service$_{k,\ (k \neq j)}$ to join the composite service (1). It could be assumed that there is always one service that returns a positive response to the invitation of joining a composite service.

Table 4: Sample of a conversation message

<Conversation
Identifier: conversation$_1$
In-reply-to: null
From: Service$_i$
To: Service$_j$
Content:
Subject: request-to-join-composite-service
Deadline-to-respond: time & date
. . .
/>

Case b. - In case service$_j$ cannot make a decision before the deadline of response that service$_i$ has fixed, service$_j$ requests from service$_i$ to extend this deadline (2.2). Service$_i$ has two alternatives: a) refuse the request of service$_j$ which means that service$_i$ has to look again for another service (Case a.), or b) accept the request of service$_j$ which means service$_j$ will get notified about the acceptance of service$_i$ (2.2.1). In alternative b), service$_j$ enters the assessment state again in order to make a decision. Service$_j$ may request an extension of the deadline for several reasons. For example, it cannot commit additional instances of service$_j$ while other instances have not yet completed their execution. Indeed, it is argued that for service composition it is desirable to dynamically choose service providers, and service instances based on current network and servers loads.

Case c. - In case service$_j$ accepts to join the composite service it notifies its acceptance to service$_i$ (2.3), so a Service Level Agreement (SLA) can be established [12]. At the same time, service$_j$ enters the preparation state to get itself ready for execution. It should be noted in Figure 8 that Accept-to-join link between conversation and preparation states of service$_j$ plays two roles: a transition to enter the preparation state and a trigger for a conversation message to notify service$_i$.

When service$_i$ finishes its execution, it invokes service$_j$ according to the agreement that was established in Case c. Therefore, service$_j$ enters the execution state and at the same time, initiates conversations with the next services. Service$_j$ adopts the aforementioned approach.

6. RELATED WORK

The Web Service Conversation Language (WSCL) of [6] describes the structures (types) of documents a service expects to receive and produce, as well as the order in which the interchange of documents takes place. In fact, the conversation component of a service is seen as a way to describe the kinds of operations the service supports (e.g., clients to log in first and then request catalog). In our work, we see conversations as a means for services to discuss the establishment of a composite service at different levels: if to join, when to join, and with what to join. A service enters different states depending on the outcome of conversations. The interactions that a service supports are part of the activities undertaken within the states.

In [7], the authors discussed the way DAML-S organizes a Web service description into three conceptual areas. The *profile* area describes what the service does in terms of advertising, discovery, and matching. This is the kind of information service-seeking agents require in their work. The *process model* area tells how the service

works, including information about the service's inputs, outputs, pre-conditions, and effects. The process model is also important in composing and monitoring processes. Finally, the *grounding* area tells how an agent can access a service. Typically, it specifies a communication protocol and provides details such as port numbers used in contacting the service. The conceptual areas that DAML-S puts forwards have a lot of similarities with the perspectives that embed a service chart diagram. First, the profile can be associated with the organization perspective. Indeed, an organization that provides a service decides what functionalities and capabilities to put into a service. Second, the process model corresponds to the flow perspective at the composite service level and to the state chart diagram at the service level. Finally, the grounding corresponds to the next services field of the flow perspective. A service that is listed in that field is annotated with the protocol that enables its invocation.

Conversations between Web services have attracted the attention of Ardissono et al. [3]. Ardissono et al. worked on a conversational model that aims at supporting complex interactions between clients and Web services, where several messages have to be exchanged before the service is completed. Conversation may evolve in different ways, depending on the state and the needs of the participants. While we view the conversations of [3] as application-domain dependent and execution-driven, our suggested conversations are application-domain independent and composition-driven. It should be noted that both types of conversations complement each other. Composition-driven conversations are part of the initial exchange of messages that takes place during the preparation of a composite service (e.g., does a Web service have an interest in joining a composition process?). While execution-driven conversations illustrate the exchange of messages that occur during the deployment of a composite service (e.g., how to submit a user's request to a Web service?). Therefore, the chronology of conversations starts with composition-driven conversations and continues with execution-driven conversations.

7. CONCLUSION

In this paper, we presented an approach for designing service-driven applications. Service chart diagrams constitute the backbone of the approach; they leverage the traditional state chart diagrams of UML. Additional elements are added to state diagrams, such as the organization that offers a service and the place of where the execution of the service takes place. The specification of a composite service consists of connecting the service chart diagrams of all the services that are involved in that composite service. Before connecting them, contributing services engage conversations to decide if they join the composite service or not. Conversations aim at raising raise the services to the level of autonomous components. As stated in [2], the services that are capable of engaging intelligent interactions would be able to discover and negotiate with each other, mediate on behalf of their users, and compose themselves into more complex services.

One of the main issues that needs to be dealt with during conversations is scalability. If a service is requested by a great number of services, a bottleneck situation may happen. Indeed, the requested service has to engage conversations with each service which could definitely take time and require computing resources.

Acknowledgments

The authors would like to thank Aysha Alsayed Mohamed Ismail Almarzouqi, a fourth year student at ZU for her tourist mobile-book service.

8. REFERENCES

[1] Simple Object Access Protocol (SOAP). http://www.w3.org/TR/SOAP/, Visited July 2002.

[2] S. Akhil, M. Vijay, S. Mehmet, L. Li Jie, and C. Fabio. Automated SLA Monitoring for Web Services. Technical Report HPL-2002-191, HP Laboratories, Palo Alto, California, USA, 2002.

[3] 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 (forthcoming).

[4] B. Benatallah and F. Casati (*Editors*). Special Issue on Web Services. Distributed and Parallel Databases, An International Journal, Kluwer publishers, 12(2-3) September 2002.

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

[6] 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. UDDI Working Draft Best Pratices Document, Hewlett-Packard Company.

[7] J. J. Bryson, D. L. Martin, S. A. Mcllraith, and L. A. Stein. Toward Behavioral Intelligence in the Semantic Web. *IEEE Computer*, 35(11), November 2002.

[8] B. Chaib-draa and F. Dignum. Trends in Agent Communication Language. *Computational Intelligence*, 2002.

[9] L. L. Constantine. *Fundamentals of Object-Oriented Design in UML*. Addison-Wesley, 2000.

[10] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2), March/April 2002.

[11] M. Huhns. Agents as Web Services. *IEEE Internet Computing*, 6(4), July/August 2002.

[12] 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.

[13] Z. Maamar, B. Benatallah, and Q. Sheng. Towards a Composition Framework of E-/M-Services. In *Proceedings of The 1st International Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Devices held in conjunction with the 1st International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS'2002)*, Bologna, Italy, 2002.

[14] Z. Maamar, W. Mansoor, and Q. H. Mahmoud. Software Agents to Support Mobile Services. In *Proceedings of the First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS'2002) (Poster Session)*, Bologna, Italy, 2002.

[15] J. Roy and A. Ramanujan. Understanding Web Services. *IEEE IT Professional*, November/December, 2001.