Software & System Modeling (2006) 00:
DOI 10.1007/s10270-006-0006-z

SPECIAL ISSUE PAPER

Zakaria Maamar · Djamal Benslimane · Michael Mrissa · Chirine Ghedira

$\mathcal{C}$oo$\mathcal{PS}$ – Towards a method for coordinating personalized services

Received: 27 April 2004 / Revised: 20 September 2005 / Accepted: 20 September 2005

Abstract This paper presents $\mathcal{C}$oo$\mathcal{PS}$, which is a method for $\mathcal{C}$oordinating $\mathcal{P}$ersonalized $\mathcal{S}$ervices. These services are primarily offered to mobile users. The concept of services is the object of intense investigations from both academia and industry. However, very little has been accomplished so far regarding first, personalizing services for the benefit of mobile users, and second, providing the appropriate methodological support for those (i.e., designers) who will be specifying the operations of personalization. Various obstacles still exist such as lack of techniques for modeling and specifying the integration of personalization into services, and existing approaches for service composition typically facilitate orchestration only, while neglecting contexts of users and services. $\mathcal{C}$oo$\mathcal{PS}$ consists of several steps ranging from service definition and personalization to service deployment. Each step has some representation techniques, which aim at facilitating the specification and validation of the operations of coordinating personalized services.

Keywords Service · Coordination · Personalization · Method

1 Introduction and motivation

Nowadays, Internet technologies are supporting businesses in the deployment of new strategies of interaction with their peers and customers. Services, particularly *Web services* are among the core components of these strategies [12]. Business-to-Customer (B2C) cases identified the first generation of Web services. More recently, businesses have started connecting their processes with other peers through what is commonly referred to as Business-to-Business (B2B) Web services. The advantages of Web services highlight primarily their capacity to be composed into high-level business processes, which could span over multiple distributed and heterogeneous applications. The widespread adoption of Web services is due to several standards including WSDL, UDDI, and SOAP [21].

Parallel to the new role of the Internet as a vehicle of delivering services, a major growth in the field of wireless and mobile technologies is witnessed. Evidences of this growth are multiple including surfing the Web from mobile devices and exchanging pictures between mobile devices. Because users are relying more and more on mobile devices to conduct their operations, enacting services from such devices, and possibly downloading these services from their respective hosting sites to such devices for execution constitute research avenues that are worthwhile pursuing [30]. *M-services* (M for Mobile) denote the services that are intended to be deployable in a wireless environment [19].

Composing services (Web services or M-services) rather than accessing a single service is essential and offers better benefits to users. Composition addresses the situation of a user's request that cannot be satisfied by any available component service, whereas a *composite service* obtained by combining the available services (Web services, M-services, or composite services) might be used for satisfying the request. Discovering and selecting the component services according to the requirements of users, integrating the selected component services into a composite service, triggering the composite service for execution, and finally monitoring the execution of the composite service are among the operations that users will have to be responsible for. The aforementioned operations are so complex that is deemed appropriate considering *software agents* to assist users in performing them [9]. A software agent is an autonomous entity that acts on behalf of user, makes decisions, interacts with other agents, and migrates to distant hosts if needed. Besides the importance of assisting users, it is also deemed appropriate supporting those (i.e., designers) who will be responsible for designing, specifying, validating, and implementing the operations related to service composition.

Communicated by: Bernhard Schätz and Ingolf Krüger

Z. Maamar (✉)
Zayed University, Dubai, United Arab Emirates
E-mail: zakaria.maamar@zu.ac.ae

D. Benslimane · M. Mrissa · C. Ghedira
Claude Bernard Lyon 1 University, Lyon, France

Research on services has concentrated on three aspects: (i) service description and modeling, (ii) service discovery, and (iii) service composition. One of the composition operations that will definitely benefit from a methodological support is the coordination of personalized services in an environment of mobile users. On the one hand, coordination means the mechanisms that specify the orchestration of the component services of a composite service. The orchestration is about among other things the execution chronology of the component services, the data that the component services exchange, the states that the component services take according to this exchange, and last but not least the actions that the component services execute. On the other hand, personalization means the integration of the preferences of users into the specification of the orchestration of the component services. In this paper users' preferences are of type location and time, where each type is organized along two perspectives (i.e., execution perspective – where and when a mobile user would like to have services performed; delivery perspective – when and where a user would like to have the outcome of performing services delivered). Through appropriate mechanisms, users will have the opportunity to adjust the specification of the coordination of the component services according first, to their personal preferences and second, to the features and constraints of the environment in which they will probably be located (e.g., is there any network coverage in this building, is there any printer close to user?). Keeping track of the progress of the user-adjusted specification requires some awareness mechanisms, which detect for instance the changes in the environment. These mechanisms are installed on top of a structure, which we refer to as *context*. Context is the information that characterizes the interaction between users, applications, and the surrounding environment [10]. In addition, to ensure much better interactions between humans, applications, and the environment, it is recommended to proceed with leveraging these interactions to the level of *conversations* [16]. A conversation is a consistent exchange of messages between participants who are involved in joint operations and thus, have common interests.

Composition of services is a very active area of R&D. However, *very little* has been accomplished so far regarding first, personalizing services for mobile users, and second, providing the appropriate support through methodologies for those who will undertake the specification of personalization operations. In particular, several obstacles still exist such as (i) lack of techniques for modeling and specifying the integration of personalization into services, (ii) services are dealt with as passive components rather than active components that can be embedded with context-awareness mechanisms, (iii) existing approaches for service composition (e.g., WSFL and BPEL) typically facilitate orchestration only, while neglecting information about the context of users and services, and (iv) existing coordination approaches (e.g., WS-Coordination) are not specifically intended to support the deployment of personalized services. In this paper, $\mathcal{C}$oo$\mathcal{PS}$, which is a method for $\mathcal{C}$oordinating $\mathcal{P}$ersonalized $\mathcal{S}$ervices in an environment of mobile users is presented. Section 2 overviews some concepts like Web services, software agents, and context. Section 3 illustrates the steps that $\mathcal{C}$oo$\mathcal{PS}$ encompasses. Section 4 discusses the role of context in $\mathcal{C}$oo$\mathcal{PS}$. Section 5 is about related work and Sect. 6 concludes the paper.

2 Preliminaries

What is a Service? In this paper, two types of services are considered: Web service and M-service. Benatallah et al. provide the following properties for a Web service [6]: independent as much as possible from specific platforms and computing paradigms; mainly developed for inter-organizational situations; and easily composable so that developing complex adapters for the needs of composition is not required. With regard to M-services, Maamar and Mansoor propose two definitions [19]. The weak definition is to remotely trigger a service for processing from a mobile device. In that case, the service acts as an M-service. The strong definition is to transfer a service from its hosting site to a mobile device where its processing occurs. In that case, the service acts as an M-service that has the following properties: transportable through wireless networks; composable with other M-services; adaptable according to the computing features of mobile devices; and runnable on mobile devices.

What is a Software agent? A software agent is a piece of software that autonomously acts to carry out tasks on users' behalf [9]. The design of many software agents is based on the approach that the user only has to specify a high-level goal instead of issuing explicit instructions, leaving the how and when decisions to the agent.

What is Context? Dey defines context as any information that is relevant to the interactions between a user and an environment [14]. This information can be about the circumstances, objects, or conditions by which the user is surrounded. Many researchers have attempted defining context among them Schilit et al. who propose computing, user, and physical context [29].

What is Personalization? For Bonett, *personalization involves a process of gathering user-information during interaction with the user, which is then used to deliver appropriate content and services, tailor-made to the user's needs. The aim is to improve the user's experience of a service* [8]. Personalization is mainly about integrating users' preferences into the process of delivering any information-related content or outcome of service computing.

What is Conversation? Conversations have been the object of multiple investigations in various research fields. In this paper we discuss conversations from a service perspective. A conversation is a consistent exchange of messages between participants that are engaged in joint operations and hence, have common interests.

What is Coordination? According to Papazoglou and Georgakopoulos, coordination of Web services is to "*control the*

execution of the component services, and manage dataflow among them and to the output of the component service" [25]. In addition to the communication language component, coordination is another core component on which collaboration is built upon. The purpose of coordination is to prevent conflicts on various aspects by specifying who is in charge of doing what and when and where this is supposed to be done [20]. Coordination can be either implicit or explicit. Implicit coordination assumes that the participants are aware of the existence of certain rules to adopt. The rules are predefined, made available, and related to a particular application-domain (e.g., car traffic). Explicit coordination requires a clear specification of the responsibilities and the mechanisms that are used for conflict resolution.

3 Presentation of $\mathcal{C}oo\mathcal{PS}$

3.1 Service specification step

The first step of $\mathcal{C}oo\mathcal{PS}$ is to specify the component services of a composite service. We recall that, in this paper, it is assumed that these component services are already identified and integrated into the composite service. The service specification step relies on the concept of *service chart diagram* [17], which is a means for modeling and defining services. This diagram enhances a state chart diagram, putting emphasis on the context surrounding the execution of a service rather than only on the states that a service takes (Fig. 1). To this end, the states of a service are wrapped into five perspectives, each perspective has a set of parameters. The state perspective corresponds to the state chart diagram of the service. The flow perspective corresponds to the execution chronology of the composite service in which the service participates. The business perspective identifies the organizations that make the service available. The information perspective identifies the data that are exchanged between the service and the rest of the services of the composite service. Finally, the performance perspective illustrates the ways the service is invoked for execution.

A service chart diagram is primarily intended to be used by designers who are familiar with the concepts and approaches of design and development. When it comes to service personalization (Sect. 3.5), which is the responsibility of users, these users have to indicate when and where they would like to have the component services performed according to particular periods of time and particular locations.

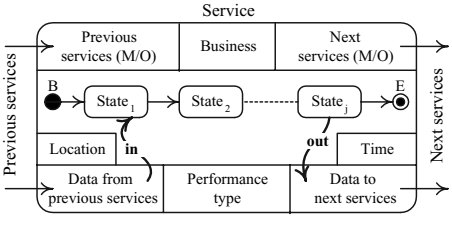

Fig. 1 Service chart diagram of a component service

Users may also indicate when and where they would like to have the outcome of performing the component services delivered. To meet the requirements of personalization, two extra perspectives denoted by *location* and *time* are anchored to a service chart diagram (Fig. 1).[1] Location means particular places such as `classroom`, and time means particular moments such as `in the afternoon`.

Because a composite service is made up of several component services, the process model that underlies the composite service is specified as a state chart diagram.[2] In this diagram, states are associated with the service chart diagrams of the component services (Fig. 1), and transitions are labelled with events, conditions, and variable assignment operations. For illustration purposes, Fig. 2 presents *travel assistant composite-service* as a state chart diagram. The composite service consists of four dependent services, associated each with a service chart diagram: *flight booking*, *hotel booking*, *attraction search*, and *car rental*.

In this paper, it is assumed that all the information needed to instantiate the perspectives' parameters of a service chart diagram are known in-advance to designers (instantiation means assigning values to the parameters of the perspectives). Therefore, a designer knows the pre- and post-data that are exchanged between the component services, the businesses that provide the component services, and the possible ways of invoking the component services. For instance, *flight booking* is the first service to be triggered and is followed by *hotel-booking* and *attraction-search* services. These services need the following inputs: *departure date*, *return date*, and *destination city*. *Flight booking* takes *stand by* and *execution* states. For personalization purposes, the instantiation of the parameters of time and location perspectives of *flight-booking* service is left for users (Sect. 3.5).

3.2 Service agentification step

The second step of $\mathcal{C}oo\mathcal{PS}$ consists of agentifying the participants that are involved in coordinating personalized services. Agentification means the identification of the relevant types of software agent, which will be responsible for deploying the specifications of composite services. A sample of specification is given in Fig. 2. The participants are categorized into user, component service, and composite service. The designer assigns a software agent to each category of participant, which results in the following types of software agent: *user-agent*, *service-agent*, and *composite-agent* (Fig. 3).

[1] The semantics of time and location perspectives is based on interval relationships [1] and qualitative spatial representations [24], respectively.

[2] Several languages for specifying Web services composition exist such as Business Process Execution Language [13] and Web Services Flow Language [15]. In our work, we adopt state chart diagrams and the value-added of these diagrams to Web services composition is discussed in [5].

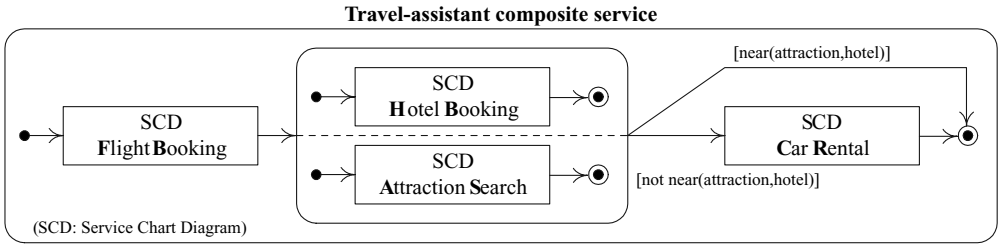

Fig. 2 Travel assistant composite-service as a state chart diagram

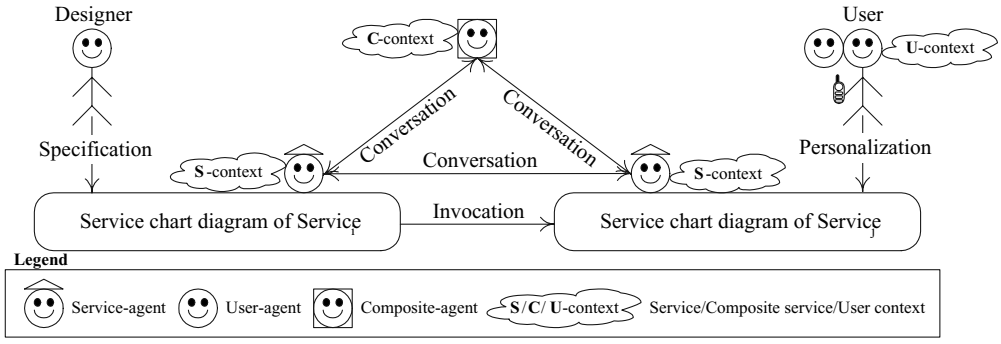

Fig. 3 Deployment of software agents on top of service chart diagrams

From a design perspective, the development of a composite service requires working on how to connect the component services together (Fig. 3). In $\mathcal{C}oo\mathcal{PS}$, the connections are identified at two levels. The highest level of connection, referred to as conversation, defines the messages that are exchanged between the service-agents of the component services. The messages are on various matters including (i) what is the preparation status of the next component services, (ii) where do the next component services need to be executed by finding out the current location *vs*. the location that the user has indicated in his preferences, and (iii) when do the next component services need to be executed by finding out the current time *vs*. the time that the user has indicated in his preferences? Acting as a conversation controller, the composite-agent monitors the conversations that service-agents initiate. A composite-agent ensures that the right component services are selected, added, triggered, and executed with respect to the specification of the composite service that this composite-agent represents (Fig. 2). To this purpose, the composite-agent takes various actions, which include for example comparing the current execution location of a component service to the execution location as indicated in the specification (i.e, requested by user). Because service-agents and composite-agents will engage in conversations, they need to comply with multiple types of policy for regulating the exchange of these conversations. The specification of conversation policies is the responsibility of designers and is carried out in the conversation specification step (Sect. 3.4).

The lowest level of connection, referred to as invocation in Fig. 3, identifies the messages that implement the conversation level. Samples of this implementation include data transfer, service invocation, and request submission. Because of the technical obstacles that may face the implementation of conversations (e.g., network connection interrupted), specific actions for exception handling are needed. These actions are included in the conversation policies and assigned to service-agents. In case of any exception occurrence, a service-agent has first, to notify the composite-agent and second, make its respective component service takes an appropriate state. In this state, the component service will have to carry out some corrective actions. For instance, if a component service does not acknowledge a data reception, the sender component-service has to enter a new state so this lack of acknowledgment is handled in proper and timely manners. The specification of the corrective actions to take during conversations is part of the conversation specification step (Sect. 3.4).

The designer of a composite service can trace the deployment progress of the specification he has devised using the information that composite-agents, service-agents, and user-agents manage on their respective components. This information is stored in a structure that is denoted by context. The designer specializes context into $\mathcal{C}$-context (context of $\mathcal{C}$omposite service), $\mathcal{S}$-context (context of $\mathcal{S}$ervice), and $\mathcal{U}$-context (context of $\mathcal{U}$ser). The specification of the internal structure of each type of context is the responsibility of designers (Sect. 3.3).

In $\mathcal{C}oo\mathcal{PS}$, two features support the coordination of the execution of a composite service's specification. These features are referred to as *one-step ahead* and *backward conversation* (Fig. 4). By one-step ahead, we mean that while a set of component services are under execution, a monitoring of the specification of the next component services is

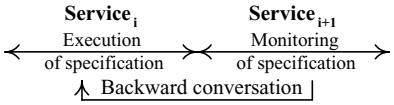

Fig. 4 Concurrent processing of execution and monitoring of services

concurrently performed. The monitoring consists of checking (i) when and where the next component services need to be executed and (ii) what data need to be received from the previous component services. In case the service-agent of a component service to be executed detects a potential delay in receiving data or that the appropriate location is different from the location that the user has indicated as part of his preferences, the service-agent initiates a backward conversation with the appropriate predecessor service-agents. It should be noted that the information with whom a service-agent converses is available in the service chart diagram of the service that the service-agent represents (Fig. 1). The objective of the backward conversation is to make aware the predecessor service-agents of the possibility of violating the specification of the composite service, which means the obligation of taking corrective actions. A note about the backward conversation is also sent to the composite-agent for notification and context update.

3.3 Context specification step

The third step of $\mathcal{C}$oo$\mathcal{PS}$ consists of specifying the contexts of the multiple participants (user, component service, composite service) that were considered in the service agentification step (Sect. 3.2). In this third step, specification means the definition of the parameters that constitute the internal structure of a context.

In Fig. 3, $\mathcal{C}$-context, $\mathcal{S}$-context, and $\mathcal{U}$-context are highlighted. $\mathcal{U}$-context is fed by data from two independent sources: user's mobile device for time-related matter and user (i.e., himself) for location-related matter.[3] $\mathcal{S}$-context is fed by data from $\mathcal{U}$-context and the perspectives of a service chart diagram. A service-agent continuously checks the $\mathcal{S}$-context so it takes actions and engages in conversations with a composite-agent and other service-agents. Finally, $\mathcal{C}$-context is fed by data from $\mathcal{S}$-contexts and $\mathcal{U}$-context. A composite-agent continuously checks the $\mathcal{C}$-context so it takes actions and engage in conversations with service-agents. $\mathcal{C}$oo$\mathcal{PS}$ adopts Tuple Spaces [11] for implementing the update operations between contexts. However, to keep the paper self-contained tuple spaces are not discussed.

The $\mathcal{S}$-context of a component service consists of the following parameters (Table 1): label, service-agent label, status, previous component services, next component services, regular actions, start time (requested and effective), location (requested and effective), reasons of failure, corrective actions, and date. It should be noted that *time-effective* and *location-effective* parameters are execution-dependent (i.e., when and where the execution has effectively happened), whereas *time-requested* and *location-requested* parameters are user-dependent. Concepts presenting some similarities to effective and requested parameters are discussed in Sect. 5.

To ensure that the preferences of a user are considered during the deployment of the specification of a component service (Fig. 2), the values of *time-requested* and *location-requested* parameters should respectively be *equal* to the values of *time-effective* and *location-effective* parameters (a minimal difference is also accepted between requested and effective parameters). Any discrepancy between the parameters of type *requested* and the parameters of type *effective* indicates that the user's adjustment with regard to *execution location* or *execution time* of a component service has not been properly handled (e.g., due to some unforeseen obstacles, the *effective-execution* location of a service is location$_1$. However, the execution location that the user has requested is location$_2$).

The $\mathcal{C}$-context of a composite service is built upon the $\mathcal{S}$-contexts of its respective component services and consists of the following parameters (Table 2): label, composite-agent label, previous component services, current component services, next component services, status per component service, time, and date. It should be noted that *status per component service* parameter is service-dependent since this parameter's value is obtained from *status* parameter of $\mathcal{S}$-context (Table 1).

The $\mathcal{U}$-context of user consists of the following parameters (Table 3): user-agent label, previous locations per component services, current location per component services, next locations per component services, previous periods of time/component services, current period of time per component services, next periods of time per component services, and date.

3.4 Conversation specification step

The fourth step of $\mathcal{C}$oo$\mathcal{PS}$ consists of specifying the conversations that occur, first, between the component services of a composite service, and, second, between the component services and a composite service (Fig. 3). Specification means the definition of the communication protocols and policies that regulate the exchange of conversation messages.

In Fig. 3, the conversations have the following participants: component services and composite services. $\mathcal{C}$oo$\mathcal{PS}$ does not recommend leveraging the interactions between users and services to the level of conversations. In fact, these interactions are simply limited to a single round of exchange (e.g., question/answer interaction pattern). Conversations are complex because several rounds of interaction are required (e.g., propose / counter-propose / accept⊕reject⊕counter-propose/$\cdots$) before the outcome

[3] Automatic detection of user's location can be used [26]. In this paper, a manual detection of the user's current location is promoted and argued as follows. This type of detection allows a better handling of the privacy issue as users only reveal the locations that they wish to be revealed to external systems.

Table 1 Description of $\mathcal{S}$-context parameters

Parameter & Description
Label: corresponds to the identifier of the component service.
Service-agent label: corresponds to the identifier of the service-agent of the component service.
Status: informs about the current status of the component service (in-progress, suspended, aborted, or terminated).
Previous component services: indicates if there are component services before the component service (null if there are no predecessors).
Next component services: indicates if there are component services after the component service (null if there are no successors).
Regular actions: illustrates the actions that the component service normally performs; these actions are described in the state perspective of the service chart diagram of the component service (Fig. 1).
Start time (requested and effective): informs when the execution of the component service should start (i.e., user-dependent) and has started (i.e., execution-dependent).
Location (requested and effective): informs where the execution of the component service should happen (i.e., user-dependent) and has happened (i.e., execution-dependent).
Reasons of failure: informs about the reasons that are behind the failure of the execution of the component service.
Corrective actions: illustrates the actions that the component service has to perform in case the execution fails; these actions are described in the state perspective of the service chart diagram of the component service (Fig. 1).
Date: identifies the time of updating the parameters above.

Table 2 Description of $\mathcal{C}$-context parameters

Parameter & Description
Label: corresponds to the identifier of the composite service.
Composite-agent label: corresponds to the identifier of the composite-agent of the composite service.
Previous component services: indicates which component services of the composite service have been executed with regard to the current component services (null if there are no predecessors).
Current component services: indicates which component services of the composite service are now under execution.
Next component services: indicates which component services of the composite service will be called for execution with regard to the current component services (null if there are no successors).
Status per component service: returns the status of each component service of the composite service that has been executed and under execution as well.
Time: informs when the execution of the composite service has started.
Date: identifies the time of updating the parameters above.

Table 3 Description of $\mathcal{U}$-context parameters

Parameter & Description
User-agent label: corresponds to the identifier of the user-agent.
Previous locations per component services: keeps track of all the locations, as requested by the user, that have featured the execution of component services (null if there are no previous locations).
Current location per component services: indicates the current location, as requested by the user, that should feature now the execution of component services.
Next locations per component services: indicates all the locations, as requested by the user, that will feature the execution of component services (null if there are no next locations).
Previous periods of time per component services: keeps track of all the periods of time, as requested by the user, that have featured the execution of component services (null if there are no successor periods of time).
Current period of time per component services: indicates the current time, as requested by the user, that should feature now the execution of component services.
Next periods of time per component services: keeps track of all the periods of time, as requested by the user, that will feature the execution of component services (null if there are no next periods of time).
Date: identifies the time of updating the parameters above.

expected out of these rounds of interaction is obtained. In addition, during conversations participants have to adjust their behavior with regard to the messages that participants receive and submit. This is not the case with the participants that decide adopting a question/answer interaction pattern. Backing the importance of integrating conversations into service composition as $\mathcal{C}$oo$\mathcal{PS}$ promotes, Ardissono et al. observe that current Web services standards such as WSDL are integrated into systems featured by simple interactions [2]. This is not enough as there are Web services-based situations where it is required to express complex interactions by using conversations. The same comment is made by Benatallah et al. in [4], who noticed that despite the growing interest in Web services, several issues remain to be addressed to provide Web services with benefits similar to traditional integration middleware. One

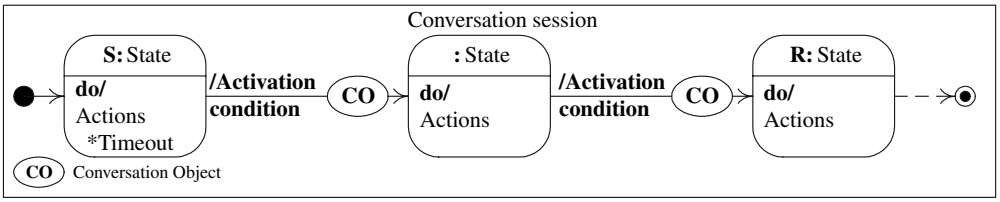

Fig. 5 Extended state diagram for specifying conversation sessions

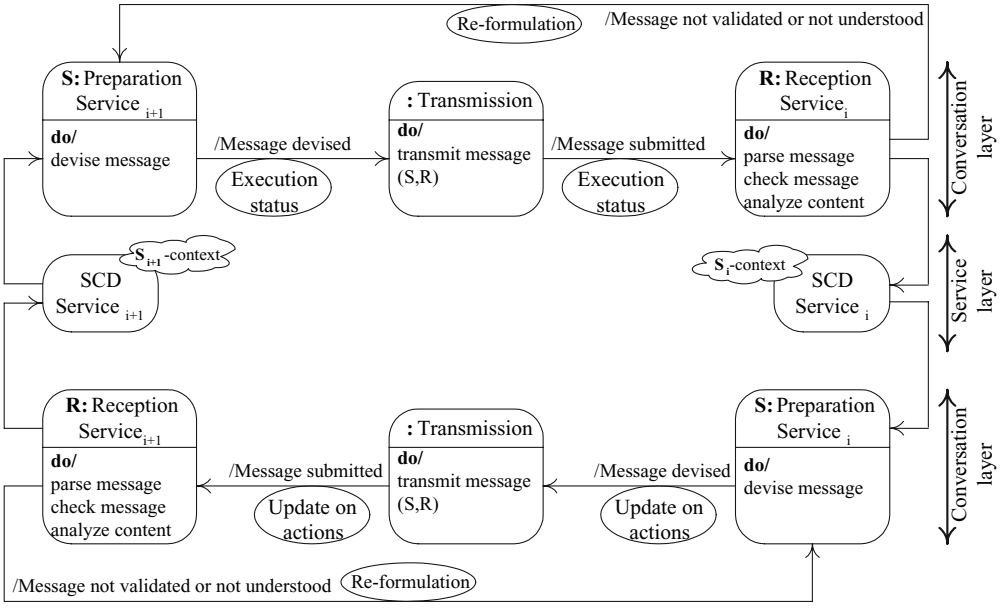

Fig. 6 Extended state diagram of execution and monitoring conversation session

of Benatallah et al.'s suggestions to enhance Web services is the development of a conversational metamodel.

The conversation specification step of $\mathcal{C}oo\mathcal{PS}$ relies on the concept of *extended state chart diagram* (Fig. 5), which is thoroughly discussed in [16]. This diagram is a means for modeling and defining conversations.

In the coordination of personalized services, two types of conversation session are identified: *invitation of services to participate in a composition*, and *execution and monitoring of component services*. In the following, we illustrate how a designer specifies the second conversation session. This session is primarily selected since it refines the description that is associated with Fig. 4. In the execution and monitoring conversation-session, the objective is to establish a backward conversation from the component services that are due for execution to the component services that are currently under execution. In Fig. 6, two types of layer exist. The first type of layer, called conversation, corresponds to the states that the conversation takes in this session. The second type of layer, called service, corresponds to the states that the services participating in this session take. We recall that the specification of services is based on service chart diagrams (Fig. 1).

Because services are associated with $\mathcal{S}$-contexts, their respective service-agents compare the current content of the $\mathcal{S}$-contexts (Table 1) to the designer's specification (Fig. 2). In case a service (i.e., through its respective service-agent) detects a discrepancy such as wrong location with regard to the user's preference, the service immediately alerts the previous service and the composite service about this discrepancy. As a first step, the service initializes a conversation message through the state (**S**:preparation – service$_{i+1}$). Once the message is devised, it is sent to the previous service as the state (**R**:reception – service$_i$) illustrates. Before that, the conversation takes the state (:transmission), which consists of transferring the message from the sender (i.e., service$_{i+1}$) to the receiver (i.e., service$_i$). Once the message is received, the receiver service parses and analyzes the content of the message. The message is returned to the sender in case it is not valid. Otherwise the content of the message (i.e., conversation object) makes the receiver service takes the appropriate actions such as identifying the right execution location of the service. Afterwards, the receiver service, which acts now as a sender, returns information to the sender service on the actions that the receiver service has taken because of the discrepancy that the sender service has

reported. This feedback is also conducted through conversation as the following states depict (**S**:preparation – service$_i$), (:transmission), and (**R**:reception - service$_{i+1}$).

3.5 Service personalization step

The fifth step of $\mathcal{C}oo\mathcal{PS}$ consists of personalizing the component services that constitute a composite service. In this step, personalization means the instantiation of the parameters of time and location perspectives while considering the fact that users are mobile.

Personalization from a mobile perspective is complex since many issues are to be addressed. Some of the issues as reported in [23] include (i) what content to present to user, (ii) how to show the content to user, (iii) how to ensure user's privacy, and (iv) how to create a global personalization scheme? While these issues are coupled to a Web-content provisioning perspective, $\mathcal{C}oo\mathcal{PS}$ considers additional issues since personalization is coupled to a Web-service provisioning perspective. These issues are (i) at what level can a service be personalized, (ii) does service personalization occur before or after composition, (iii) to what extent can a user personalize a service, and (iv) does service personalization have to comply with specific policies?

Once the parameters of the perspectives of a service chart diagram (excluding location and time perspectives) are instantiated (Sect. 3.1) and the relevant types of agents are identified (Sect. 3.2), the designer notifies potential users about the availability of a new composite service (e.g., using SMS). If a user has interest in the composite service, he requests from the designer to send him the light version of the specification of this composite service. By light, we mean the following details on the specification: (i) list of component services (e.g., hotel booking, attraction search), (ii) execution chronology of the component service (e.g., hotel booking then attraction search), and (iii) non-instantiated location and time parameters of the component services. Upon reception, the specification is stored in the user's mobile device. Afterwards, the user selects the component services that he wishes adjusting their location and time of execution parameters. For instance, the user can indicate that *hotel-booking* service of travel assistant composite-service will be executed after 9 am once he is at work.

When the user returns back the specification of the composite service, which includes now his preferences, the designer carries out a consistency check of this specification. In case of any inconsistency, the designer invites the user either to review his adjustments or to relax some of the temporal or location constraints. The exchange between the designer and user keeps going until the specification of the composite service after adjustment is declared free of conflicts. The following is an example of conflict.

Designer specification	$Service_i$: data $\mapsto$ $Service_j$
User adjustment	$(\cdots, Service_i, ?location, time_i)$, $(\cdots, Service_j, ?location, time_j)$, before$(time_j, time_i)$

In this example, $Service_i$ will be executed before $Service_j$ due to the data dependency that is indicated in the designer's specification ($Service_i$: data $\mapsto$ $Service_j$). However, the user suggests a temporal order for these two services where the execution time of $Service_j$ (i.e., $time_j$) is *before* the execution time of $Service_i$ (i.e., $time_i$), which violates the designer specification. As a result, the temporal relationship before$(time_j, time_i)$ needs to be relaxed.

3.6 Service deployment step

The sixth and final step of $\mathcal{C}oo\mathcal{PS}$ consists of deploying the component services of a personalized composite service. In this step, deployment means the execution of the specification of the composite service as obtained in Sect. 3.1 and personalized in Sect. 3.5. The different agents that were set up in Sect. 3.2 are responsible for the execution

The execution order of the component services depends on two factors: (i) execution chronology as defined by designers, and (ii) time and location values as defined by users. Besides the three factors, information that $\mathcal{C}$-context, $\mathcal{S}$-context, and $\mathcal{U}$-context contain is used. Indeed, the current location of a user is obtained from $\mathcal{U}$-context. The current state of a service is obtained from $\mathcal{S}$-context. Last but not least, the list of component services that have (i) completed their execution, (ii) are under execution, and (iii) will be called for execution are obtained from the $\mathcal{C}$-context.

Because time and location parameters are user-dependent, a tracking of both user's current location and current time needs to be performed. When the designer declares the consistency of the specification of a personalized composite service (Sect. 3.5), the light version of this specification is kept stored in the user's mobile device. The user-agent, which also resides in the user's mobile device, compares on a regular basis the time and location parameters as indicated in the specification to the current time and current location of the user. If there is a match, the user-agent wirelessly notifies the composite-agent about the matching (Fig. 6). Once it gets notified, the composite-agent refers to the copy of the specification of the composite service it has so that it can start now executing actions among them sending invitations of participation to new component services and initiating the execution of certain component services. Details on mechanisms for inviting services are given in [18].

4 Context in $\mathcal{C}oo\mathcal{PS}$

Section 3.3 has pointed out the kind of collaboration that occurs between the three types of context. For instance, $\mathcal{S}$-context submits some context details to $\mathcal{C}$-context. When a user personalizes a composite service, *time-requested* and *location-requested* parameters of the $\mathcal{S}$-context of some (or all) of the component services are instantiated (Table 1). Because the user-agent continuously monitors the light version of the specification of a composite service (Sect. 3.6),

the user-agent identifies the component services that are due for execution with regard to a specific location or a specific time. In case the user-agent identifies candidate component services for execution, it notifies then the composite-agent, which proceeds with contacting the respective service-agents of these component services for the needs of execution. If there are no obstacles preventing the execution of the component services, *location-effective* or *time-effective* parameters receive the values of their counter-part parameters namely *location-requested* and *time-requested*. This means that the user's preferences are properly handled. Otherwise (i.e., existence of obstacles such as component service overloaded or lack of input data for a component service), the execution of the component services is postponed until these obstacles are dealt with. This means that *location-effective* or *time-effective* parameters will have values that differ from the values of their counter-part parameters namely *location-requested* and *time-requested*.

Roman and Campbell observe in [28] that a user-centric context promotes applications that (i) move with users, (ii) adapt to the changes in the available resources, and (iii) provide configuration mechanisms based on users' preferences. Parallel to the user-centric context, $\mathcal{C}oo\mathcal{PS}$ adopts a service-centric context in order to promote applications that (i) permit service adaptability, (ii) track service execution, and (iii) support on-the-fly service composition. A user-centric context is associated with $\mathcal{U}$-context, whereas a service-centric context is associated with $\mathcal{S}$-context and $\mathcal{C}$-context. Because services are the core components of a composition process, $\mathcal{C}oo\mathcal{PS}$ organizes $\mathcal{S}$-context along three perspectives: *participation*, *execution*, and *location*/*time*.

1. Participation perspective: ensures that the component services of a composite service are properly specified and got ready for composition and execution.
2. Execution perspective: ensures that the requirements in terms of computing resources of the component services are met and that the tracking of the execution of these component services is happening.
3. Location/Time perspective: ensures that location- and time-related preferences are integrated into the specification of a composite service and properly considered during deployment.

Three connections exist between participation, execution, and location/time perspectives. First, deployment denotes the connection between participation and execution perspectives and reflects the component services that are got ready for execution. Second, tracking denotes the connection between execution and location/time perspectives and reflects the monitoring of the component services that occurs according to specific locations and specific times. Finally, configuration denotes the connection between location/time and participation perspectives and reflects the component services that could be subject to adjustment at the levels of location and time.

5 Related work

Composition of services is a very active area of R&D. However, to our knowledge few projects have aimed at personalizing services for the benefit of mobile users and at the same, providing the appropriate support through methodologies to those who will be carrying out the specification of personalization. We present in the following some of the works that have backed shaping $\mathcal{C}oo\mathcal{PS}$ and its respective steps and representation formalisms.

The Web Services Conversation Language (WSCL) is an initiative on the integration of conversations into Web services. The WSCL describes the structure of documents that a Web service is supposed to receive and produce, as well as the order in which the exchange of these documents is supposed to occur. In fact, the conversation component of a Web service is a means for describing the operations that a Web service supports (e.g., clients have to log in first, before they can check the catalog). While the WSCL focusses on specifying the operations that Web services support, our study of conversations focusses on the mechanisms of devising composite services. Indeed, we classify these mechanisms into four categories: domain-dependent, domain-independent, composition-driven, and execution-driven. Domain-dependent conversation mechanisms deal with the features of the application domain when it comes for example to structuring the format and content of the conversation messages to exchange. Indeed, conversations for car rental are different from those for hotel booking. The opposite occurs for domain-independent conversation mechanisms when it comes for example to triggering a service for execution. The domain does not affect the way a service is triggered. Instead, the implementation technology on which the services are running affects the type of triggering. With regard to composition-driven conversations, they represent the conversations that are needed to devise a composite service such as how to look for the services and how to exchange agreements between these services. Execution-driven conversation mechanisms represent the messages that are needed to deploy a composite service. Therefore, the chronology of conversations starts with composition-driven conversations and proceeds with execution-driven conversations.

A conversation usually consists of a static and dynamic part. The static part is about the parameters that constitute the structure of a conversation message. The dynamic part is about the progress of a conversation according to a specific chronology. This chronology is specified with policies that specify for example how to react in a response of receiving a message, when to submit a message, and what to expect from an exchange. One of the initiatives dealing with policies is referred to as Conversation-Policy XML (cpXML). In cpXML, a conversation policy is a set of constraints on schemas of messages that may be sent, and on the sequencing of messages, in a conversation between two or more applications. $\mathcal{C}oo\mathcal{PS}$ uses extended state chart diagrams to specify the policies that regulate the conversation exchange (Sect. 3.4).

Barkhuus and Dey have identified three levels of interactivity for context-aware applications [3]: personalization, active context-awareness, and passive context-awareness. According to both authors, personalization, also referred to as customization and tailoring, is motivated by the diversity and dynamics featuring nowadays applications. For active context-awareness, it concerns applications that, on the basis of sensor data, change their content autonomously. In passive context-awareness applications merely present the updated context to the user and let the user specify how the application should change. $\mathcal{C}oo\mathcal{PS}$ adopts a passive context-awareness style with the manual feeding of the user's current location. While we mentioned that this type of feeding presents some limitations *vs*. an automatic feeding, it enables however an efficient handling of the privacy concern of users. According to Bisdikian et al. in [7], there are several challenges that hinder the deployment of location-based applications. From a user perspective, many people consider location monitoring as intrusive and thus, privacy-related questions arise. In $\mathcal{C}oo\mathcal{PS}$, a manual detection of the location parameter is considered. This gives users more confidence in releasing information on their location as the decision to inform about their location goes now back to users. Finally and from a personalization perspective, Reiff-Marganiec and Turner have both suggested policies as a means for managing the customization process that users carry out over telecommunication systems [27].

In Table 1, parameters of type requested and parameters of type effective have a major overlapping with the QoS of type advertised (or "promised") and QoS of type delivered. Ouzzani and Bouguettaya report that a key feature in distinguishing between competing Web services is their QoS, which encompasses several qualitative and quantitative parameters that measure how well the Web service delivers its functionalities [22]. A Web service may not always fulfill its advertised QoS parameters, due to various fluctuations related for example to the network status or resource availability. Therefore, some differences between QoS advertised and QoS delivered values occur. However, large differences indicate that the Web service is suffering a performance degradation in delivering its functionalities. The same comment is made on parameters of type requested *vs*. parameters of type effective when it comes to service personalization (Table 1). A major difference between the values of these parameters indicates the lack of considering a user's personal preferences in terms of execution time or execution location.

6 Conclusion

In this paper, we presented $\mathcal{C}oo\mathcal{PS}$, which is a method for $\mathcal{C}$oordinating $\mathcal{P}$ersonalized $\mathcal{S}$ervices in an environment of mobile users. Personalization aims at integrating users' preferences into the specification of the component services of composite services. Composition addresses the situation of a user's request that cannot be satisfied by any available service, whereas a composite service obtained by combining a set of available services might be used. The core concepts of $\mathcal{C}oo\mathcal{PS}$ are software agent, service chart diagram, state chart diagram, context, and last but not least conversation.

A case-tool that supports $\mathcal{C}oo\mathcal{PS}$-based development life-cycle is under construction using `Java NetBeans 3.6`. This case-tool has three modules: `service`, `agent`, and `context`. The service module allows designers the selection and composition of services. For instance, the module includes tools for describing and managing component services and composite services, and for monitoring their execution. The agent module is responsible for the creation of agents and support of their conversations. Finally, the context module defines and stores the structure of each context type as defined in Sect. 3.3. This module uses control tuples to manage the interactions between contexts.

$\mathcal{C}oo\mathcal{PS}$ is one step towards supporting those who will be responsible for personalizing the specification of composite services. $\mathcal{C}oo\mathcal{PS}$ consists of six steps namely service specification, service agentification, context specification, conversation specification, service personalization, and service deployment. Our future work examines the support for exception handling during the execution of personalized services. It might happen that a preference change affects the execution of the whole composite service.

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Communica tions of the ACM **26**(11) (1983)
2. Ardissono, L., Goy, A., Petrone, G.: Enabling conversations with Web Services. In: Proceedings of The Second International Joint Conference on Autonomous Agents and Multi-Agent Systems, (AAMAS'2003). Melbourne, Australia (2003)
3. Barkhuus, L., Dey, A.: Is context-aware computing taking control away from the user? Three levels of Interactivity examined. In: Proceedings of The Fifth International Conference on Ubiquitous Computing (UbiComp'2003), Seattle, Washington, USA (2003)
4. Benatallah, B., Casati, F., Toumani, F.: Web Service Conversation Modeling, a cornerstone for e-business automation. IEEE Internet Computing **8**(1) (2004)
5. Benatallah, B., Dumas, M., Sheng, Q. Z., Ngu. A.: Declarative composition and peer-to-peer provisioning of dynamic Web services. In: Proceedings of The 18th International Conference on Data Engineering (ICDE'2002). San Jose, CA, USA (2002)
6. Benatallah, B., Sheng, Q.Z., Dumas, M.: The Self-Serv Environ ment for Web services composition. IEEE Internet Computing **7**(1) (2003)
7. Bisdikian, C., Christensen, J., Davis, J., Ebling, M.E., Hunt, G., Jerome, W., Lei, H., Maes, S., Sow, D.: Enabling Location-based Applications. In: Proceedings of The 1st ACM International Workshop on Mobile Commerce (WMC'2001) held in conjunction with The Seventh Annual Interna tional Conference on Mobile Computing and Networking(MobiCom'2001). Rome, Italy (2001)
8. Bonett, M.: Personalization of Web Services: Opportunities and Challenges. ARIADNE, (2001). http://www.ariadne.ac.uk/, ISSN: 1361-3200.
9. Boudriga, N., Obaidat, M.S.: Intelligent Agents on the web: A Review. Computing in Science Engineering **6**(4) (2004)
10. Brezillon B.: Focusing on context in human-centered computing. IEEE Intelligent Systems **18**(3) (2003)

11. Cabri, G., Leonardi, L., Zambonelli, F.: Reactive tuple spaces for mobile agent coordination. In: Proceedings of The 2nd International Workshop on Mobile Agents (MA'1998), Stuttgart, Germany (1998)
12. Chung, J.Y., Lin, K.J., Mathieu, R.G.: Web services computing: advancing Software interoperability. IEEE Computer **36**(10) (2003)
13. Curbera, F., Khalaf, R., Mukhi, N., Tai, S., Weerawarana, S.: The next step in web services. Communications of the ACM **46**(10) (2003)
14. Dey, A.K., Abowd, G.D., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applica tions. Human-Computer Interaction Journal, Special Issue on Context-Aware Computing **16**(1) (2001)
15. Leymann, F.: Web Services Flow Language (WSFL 1.0). Technical report, IBM Corporation (2001)
16. Maamar, Z., Kouadri Mostefaoui, S., Benslimane, D.: Conversations for web services composition. In: Proceedings of The 7th Asia Pacific Web Conference (APWeB'2005). Shanghai, China (2005)
17. Maamar, Z., Benatallah, B., Mansoor, W.: Service chart diagrams – description and application. In: Proceedings of The Alternate Tracks of The 12th International World Wide Web Conference (WWW'2003). Budapest, Hungary (2003)
18. Maamar, Z., Kouadri Mostefaoui, S., Yahyaoui, H.: Towards an Agent- based and context-oriented approach for Web Services Composition. IEEE Transactions on Knowledge and Data Engineering **17**(5) (2005)
19. Maamar, Z., Mansoor, W.: Design and development of a software agent-based and mobile service-oriented environment. e-Service Journal, Indiana University Press **2**(3) (2003)
20. Malone, T., Crowston, K.: The interdisciplinary study of coordination. ACM Computing Surveys **26**(1) (1994)
21. Milanovic, N., Malek, M.: Current solutions for web service composition. IEEE Internet Computing **8**(6) (2004)
22. Ouzzani, M., Bouguettaya, A.: Efficient access to web services. IEEE Internet Computing **8**(2) (2004)
23. Panayiotou, C., Samaras, G.: mPERSONA: Personalized Portals for the wireless user: an agent approach. Journal of ACM/Baltzer Mobile Net working and Applications, Special Issue on Mobile Commerce (forthcoming) (2003)
24. Papadis, D., Sellis, T.: On the qualitative representation of spatial knowledge in 2D space. The Very Large Data Bases Journal, Springer Verlag **3**(4) (1994)
25. Papazoglou, M., Georgakopoulos, D.: Introduction to the special issue on service-oriented computing . Communications of the ACM **46**(10) (2003)
26. Ratsimor, O., Korolev, V., Joshi, A., Finin, T.: Agents2Go: an infras tructure for location-dependent service discovery in the mobile electronic commerce environment. In: Proceedings of The 1st ACM International Work shop on Mobile Commerce (WMC'2001) Held in Conjunction with The Sev enth Annual International Conference on Mobile Computing and Networking(MobiCom'2001). Rome, Italy (2001)
27. Reiff-Marganiec, S., Turner, K.J.: Feature Interactions in Telecommunications and software systems VII, chapter a policy architecture for enhancing and controlling features. Amyot D. and Logrippo L., IOS Press (Amsterdam) (2003)
28. R.oman, M., Campbell, R.H.: 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)
29. Schilit, B., Adams, N., Want, R.: Context-aware computing applications. In: Proceedings of The IEEE Workshop on Mobile Computing Systems and Applications. Santa Cruz, California, USA (1994)
30. Yunos, H.M., Gao, J.Z., Shim, S.: Wireless advertising's challenges and opportunities. IEEE Computer **26**(5) (2003)

Zakaria Maamar is an associate professor in computer sciences at Zayed University, Dubai, United Arab Emirates. His research interests include Web services, software agents, and context-aware computing. Maamar has a PhD in computer sciences from Laval University.

Djamal Benslimane is a full professor in computer sciences at Claude Bernard Lyon 1 University and a member of the Laboratoire d'InfoRmatique en Images et Systèmes d'information- Centre National De la Recherche Scientifique (LIRIS-CNRS), both in Lyon, France. His research interests include interoperability, Web services, and ontologies. Benslimane has a PhD in computer sciences from Blaise Pascal University.

Michael Mrissa is a Ph.D. candidate in computer sciences at Claude Bernard Lyon 1 University and a member of the Laboratoire d'InfoRmatique en Images et Systèmes d'information - Centre National De la Recherche Scientifique (LIRIS-CNRS), both in Lyon, France. His research interests include semantic Web services, interoperability and peer-to-peer networks.

Chirine Ghedira is an associate professor in computer sciences at Claude Bernard Lyon 1 University and a member of the Laboratoire d'InfoRmatique en Images et Systèmes d'information- Centre National De la Recherche Scientifique (LIRIS-CNRS), both in Lyon, France. Her research interests include Web services and context-aware computing. Ghedira has a PhD in computer sciences from the National Institute for Applied Sciences (INSA).