Service-Oriented Architectures and Mobile Services

Ivar Jørstad[1], Schahram Dustdar[2] and Do van Thanh[3]

[1] Norwegian University of Science and Technology, Dept. of Telematics, O.S. Bragstads plass 2E, N-7491 Trondheim, Norway
ivar@ongx.org
[2] Vienna University of Technology, Distributed Systems Group (DSG), Information Systems Institute A-1040 Wien, Argentinierstrasse 8/184-1, Austria
dustdar@infosys.tuwien.ac.at
http://www.infosys.tuwien.ac.at/Staff/sd/
[3] Telenor R&D, Snarøyveien 30 N-1331 Fornebu, Norway
thanh-van.do@telenor.com http://www.item.ntnu.no/~thanhvan

Abstract. Service-Oriented architectures and Service-Oriented Computing are the most recent approaches aiming at facilitating the design and development of applications on distributed systems. The primary goal of this paper is to investigate how the construction of mobile services can benefit from the Service-Oriented paradigm. The paper provides an elucidation of the Service-Oriented architecture. A general discussion of equivalence between service components is then undertaken, in order to enable an analysis of Service-Oriented architectures for mobile services. The paper proceeds with a mapping of existing mobile services on Service-Oriented architectures. The requirements of mobile services, which must be taken into consideration in the Service-Oriented architecture, are identified from a generic model of mobile services. A Service-Oriented architecture supporting mobile services is proposed.

1 Introduction

Lately, Service-Oriented Computing (SOC) [1] and Service-Oriented Architectures (SOA) [2] have gained a lot of momentum in software engineering. SOC and SOA are not completely new concepts; other distributed computing technologies like OMG's CORBA [3] and Microsoft's DCOM [4] have been based around similar principles. SOA and SOC are merely extensions of the existing concepts and new technologies, like XML Web Services, are being used to realize platform independent distributed systems.

Software engineering has moved from redundant reimplementation of similar code, through static reuse of code-segments, function-oriented programming, object-oriented programming and component-based software development. Eventually, it has ended to the concepts of distributed systems based on SOC and SOA. From developing static and tightly coupled monolithic systems, software engineering and systems development has moved to dynamic, flexible and loosely coupled platform independent distributed systems.

The provisioning and consumption of services in a Service-Oriented architecture is actually done in the same way as most everyday services. Services are published (through proper marketing) and consumed more or less as discrete units independently of other services. For example, the car washing service can be executed independently of the hair dressing service.

In a SOA, services are the primary building blocks, in contrast to earlier paradigms where the building blocks have been functions, objects or software components. Alone, or combined with others services, these building blocks can provide services to consumers. Services are an extension of the concept of components in component-based software engineering, and a service must fulfill a set of additional requirements to accommodate a SOA. Systems based on Service-Oriented architectures are flexible (primarily due to their loose coupling), and this can allow a composite service to dynamically change its internal structure (composition and distribution).

The Service-Oriented architecture appears to be an ideal paradigm for mobile services. However, it is currently focused only on enterprise and business services. One of the goals of this paper is to verify the feasibility of using Service Oriented in the design and implementation of mobile services. The paper starts with an overview of Service Oriented architectures. A brief study of current mobile services architectures presented earlier in [5] is then given. This paper is based on the composition and distribution principles as well as service continuity and service personalisation concepts published in [6][7][8][9]. A mapping of existing mobile services on Service-Oriented architectures will then be done. The requirements of mobile services, which must be taken into consideration in the Service-Oriented architecture, are identified from a generic model of mobile services. A Service-Oriented architecture supporting mobile services is finally derived.

2 Service-Oriented Architectures (SOA)

According to [10], a Service-Oriented architecture is a collection of services which communicate with each other. The question is how this is different from earlier concepts of distributed computing. With the SOA definitions of a service provided above, it is not clear how a service is different from an *application*. And if a service is not different from an application, a service logic component or a program, then it might seem like the term Service-Oriented architecture is just another new word describing the well-known concepts of distributed systems and distributed computing.

However, the key here is that a service should be *self-contained*. This means that it can always provide the same functionality, independently of other services. Distributed systems have previously been tightly interconnected, i.e., they have been subject to *tight coupling*. If one component failed, the entire service provided by the distributed system would fail. Also, previously distributed systems would be closed. Their interfaces were internal, not exposed and only known by the developers of the systems.

There are basically three functions that must be supported in a service-oriented architecture:

1. Describe and Publish service
2. Discover a service
3. Consume/interact with a service

The basic target requirements of a service-oriented architecture are:

Loose coupling – As in traditional object-oriented and component-based software engineering, one goal is to strive to have high cohesion and low coupling among classes and components. Loos coupling in a Service-Oriented architecture means that one piece of realizing software should be able to provide a service, independently of other software components. For example, a Web Service should be self-contained, and be able to provide whatever service it was intended to, without strong dependencies to other components. However, this does not restrict the Web Service from utilizing other Web Services. Loose coupling is a very important concept as it allows late binding between service consumers and the service realization. As will be discussed in a later section, this is particularly important for nomadic users.

Well-defined interfaces – The interfaces towards the service realization should be well-defined. In XML Web Services, this is done by specifying the interfaces in the Web Services Description Language (WSDL). By having well-defined interfaces, it is easy to develop new software that can access the service components.

Services in SOA are software components that have published interfaces, and these interfaces should be platform and language-independent. XML Web Services consist of technologies which allow development of platform independent software components, and are thus enabling technologies for SOA.

Stateless and autonomous service design – The standards that create a foundation for SOA (i.e., XML Web services standards) do not require stateful operation of services (each method invocation should be independent of previous method invocations). Nevertheless, services are self-contained and required to maintain *their own* state. According to [11], services should not require information to be kept between two requests. Services should not be dependent on the state or context of other services. For example, if two subsequent requests are performed to retrieve two pieces of information about the same user, both requests should include all information that is needed to decide how to retrieve this information, e.g. the complete SSN, and should not rely on programmatic artifacts like a session key.

However, for several reasons it will in many cases be necessary that some service components share state information. This will in particular be true about the communication between an end-user and a service, to simplify usage of services.

Discovery and service binding are also concepts of SOA. Consumers should be able to dynamically discover services. With regards to XML Web Services, this is enabled by UDDI.

3 Mobile Services

Let's first consider voice telephony in GSM [12] to see how it allows users to roam (i.e., user movement). Mobile services in GSM consist of components both located on the mobile device and in the network. Figure 1 shows the simplified architecture of mobile telephony service and the Short-Message Service (SMS). The Mobile Station (MS), or mobile phone, consists of two computing devices; the Mobile Equipment (ME), which is the phone itself and the Subscriber Identity Module (SIM), which is a Smart Card. For mobile telephony, the service components located on the Mobile Equipment and the SIM interact with the components on the Mobile services Switching Center (MSC), Home Location Register (HLR) and Visitor Location Register (VLR). While a mobile phone is allocated to a unique HLR, it is communicating with different MSCs and VLRs according to its location.

Figure 1 illustrates how the system allows a user to move between two service providers (two operator networks). The user physically moves, and thus needs to access another network which is reachable in the new location. For voice telephony in GSM, the user device now accesses a new radio network through different Base Transceiver Stations (BTS), controlled by other Base Station Controllers (BSC) and a different Mobile services Switching Center (MSC2). The user and device is also registered in another Visitor Location Register (VLR2). However, it still communicates with the Home Location Register (HLR) of the original network.

For the Short Message Service (SMS), all traffic goes through the Short Message Service Center (SMS-C) in the home network.

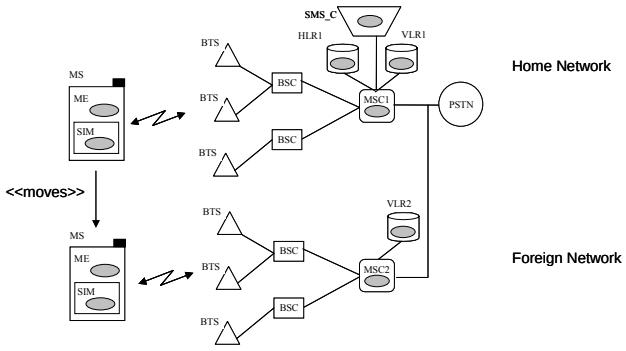

Figure 1: A user moves from one service provider to another in GSM

As shown in Figure 1, as the mobile phone moves, the components on the ME and the SIM will switch from interacting with MSC1 and VLR1 to interacting with MSC2 and VLR2. The functionality and interfaces of the components are standardised, but the instances, for examples MSC1 and MSC2, can be built by different manufacturers.

For the Short-message service the service components on the ME and the SIM always collaborate with the unique service component in the SMS-C (Short Message Service

Center) with the primary service provider (home network), no matter where in the world the mobile phone is.

For services in GSM it is possible to summarize as follows:

1. The composite service (GSM voice telephony) is realised by service components partly on a user device and partly in the network
2. The same composite service can be realised by different service components (different instances) for different users and different devices
3. The same service can be realised by service components developed by different manufacturers (different implementations) and located in different service provider locations
4. The composite service in GSM changes its internal composition to accommodate user movements

These observations will be used to illustrate how mobile services can be mapped into a service-oriented architecture.

4 Mobile Services and SOA

In a SOA, a mobile service is equivalent to an application realized by combining several SOA services. As seen in the previous section, a mobile service is composed of several standardized components, which can be interconnected at runtime to accommodate user movements.

However, mobile services pose additional requirements to the service architecture, which are not necessarily supported by service architectures for other services like enterprise and business services. For example, mobile services should tackle several types of movements:

- Movement of users between devices
- Movement of devices across networks
- Movement of services across domains

The current focus of SOA is mostly on enterprise and business services, while mobile services are not yet considered.

a. GSM and SOA

The GSM voice telecommunication service is realized by several interconnected components as seen in Figure 1. These components communicate with each other over standardized interfaces. GSM is not based on a Service-Oriented architecture, but it still shares some characteristics with SOA. The system is modularized into components (HLR, VLR, MSC etc.), which have their own specialized tasks (similar to services in a SOA). These components have well-defined interfaces which are used to access their functionality, and the specifications of these interfaces are open and

published through ETSI/3GPP. Thus, service providers can build a system with components from several different vendors, because they should be compatible. This allows users to move between service providers and still receive the same service offering.

b. Future Mobile Services and SOA

As in GSM, future mobile services will depend on the possibility to dynamically select appropriate service *instances*, based on the current location of the user. This could be as in GSM, discrete service instances, or it could be composite service instances. The current location of the user will be defined by a combination of the geographical location, current device, current network and possibly some domains:

L_{geo} = GeographicalLocation(user)
D = Device(user)
N = Network(D)
DO = Domain(user, N, D)

Location(user) = $L_{geo} \cap D \cap N \cap DO$

In contrast, with SOA the service environment is generally assumed to be one ubiquitous environment, i.e., the Internet. The notion of Location defined above is therefore not relevant for such services. Nomadic users, however, move across boundaries of these service environments (Locations), and access to services should be possible regardless of such movements.

Traditional mobile services (voice telecommunication and SMS) are personalized services; the identity of the user is strongly connected to the service, and personal adaptations can be performed (e.g. forwarding, answering machine etc.). One requirement of future mobile services is that personalization should also be possible. When a service is personalized, changes done to the service should always follow the user. For this to happen, the requirement for service continuity must also be fulfilled. These two requirements are therefore in one way strongly interconnected, and they will both depend on the composition of the services.

A generic mobile service can be said to contain some specific service components which are required to render the service; these are the components that supports the mobile characteristics of the service.

To see how a SOA can be adapted to support future mobile services and their specific requirements, it is therefore necessary to look at the architecture of a generic mobile service, i.e. what components it has and how are they interconnected. The internal composition of a generic mobile service is illustrated in Figure 2. The components are as follows:

MobileService – The overall representation of the service, consists of one or several service logic components. A Graphical User Interface (GUI) is one of these, exposing the service to the user. This component is most often a composite service, or an application in SOA terminology.

ServiceLogic – This is a piece of executable code, e.g. a program, or more precisely a service in SOA terminology.

ServiceState – These are state variables and other internal data used by a service logic component (e.g. register values). These have no equivalent in the common SOA terminology, but are assumed an integrated part of a service.

ServiceContent – These are other data that are used in conjunction with a service, e.g. documents, but which are not part of the state of the service logic component. Common SOA terminology does not include such a component.

ServiceContentMetaData – These are meta-information used together with ServiceContent. They can be data which provide additional value to the service content. Take an internet browser as an example; the *ServiceContent* are Web pages and the bookmark list is part of the *ServiceContentMetaData*. Common SOA terminology does not include an equivalent concept.

ServiceProfiles – These are profiles (*UserServiceProfile* and *DeviceServiceProfile*) containing information about how the service should behave towards specific users and devices. Common SOA terminology does not include this concept.

The *KeyStateVariables* component will be explained later.

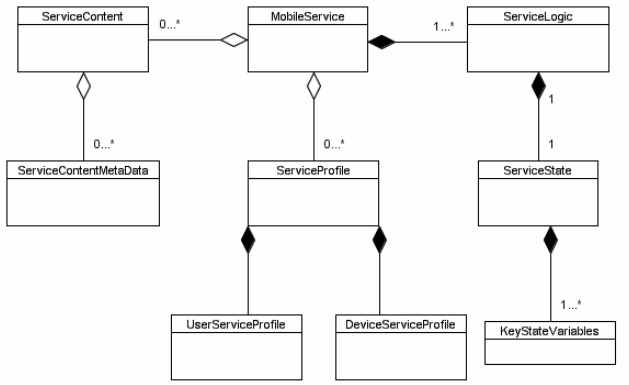

Figure 2: The composition of a generic mobile service

In summary, the support for service continuity and personalization in mobile services is dependent on the availability of instances (of the same or similar implementations) of the ServiceLogic component (considered in the next subsection) and the access to the *specific* ServiceState, the *specific* ServiceContent and the *specific* ServiceProfile.

c. Service Equivalence

As mentioned in the introduction, it could be possible to substitute one service for another, if they can be proven to be similar enough. In turn, this can be used to accommodate movements of a user and to realize mobile services.

Any movement might result in the original (composite) service no longer being available, e.g. because one of the discrete services is out of reach from the new location (see Figure 3). S_1 and S_2 represent composite services used directly by the user. S_3 and S_4 are discrete service components, used by S_1 to offer the composite service. When the user moves to a new service environment, the composite service is offered through S_2. However, this service has only access to S_4; S_3 is no longer accessible. A possible solution is to access S_5, which provide a service similar to S_3.

An analogy to GSM is that it would not be possible to send an SMS from the foreign network (Service Environment B), because the new network can not communicate with the home network and is thus not able to send messages through the SMS-C (S_3) in the home network of the user.

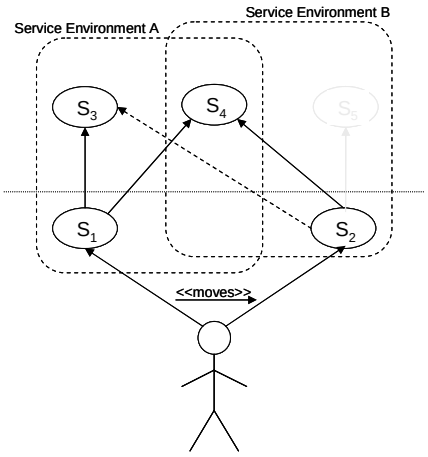

Figure 3: Composite services in different service environments

To allow various movements, it might be necessary to dynamically change the service architecture, e.g. by switching from one service provider to another or to switch between similar service components provided by the current service provider.

The goal of a SOA that supports mobile services will be to enable this type of functionality for generic services. The SOA must allow discrete, and possibly composite, services to be substituted by other instances and implementations. To be able to do this, a set of definitions and a framework for describing similarities of services is required, i.e. how can two service components be compared. The steps required in such a process are:

1. Discover candidate service components
2. Compare candidates with current component; do any of the candidates cover *satisfactorily amounts* of the functionality provided by the current component?

3. Switch between the components; i.e., realize the connection between the components

The first two steps pose the biggest challenges, while the third is simply a matter of changing a service identifier (e.g. a URI). The discovery step requires semantic searching capabilities (as approached by the Semantic Web[13][14][15] and OWL[16]/OWL-S[17]), while the description "satisfactorily amounts" must be formalized so that the entire process described above can be automatically performed by computers.

Semantic Equivalence

To be able to decide if two implementations are semantic equivalent, it is necessary to have a common vocabulary describing service concepts. This is where ontologies come into play, like those built on OWL and OWL-S. It will be necessary to define ontologies for each service domain, since service domains typically are quite different (e.g. hotels and gas stations provide services that are fundamentally different). However, it is clear that semantic equivalence of two services S_1 and S_2 is determined by:
1. Semantic equivalence of the *service concept* – what is the overall benefit offered by this service?

2. Semantic equivalence of the interfaces (e.g. semantics of methods and of parameters within these methods) used to access this service (what is it the user provides and what is it he receives in return, not talking about implementation details like data types but rather about concepts).
Semantic equivalence is not formally treated in this paper, but it is assumed that the semantic equivalence of S_1 and S_2 is defined by an equivalence relation:

S_1 ***SE*** S_2

This statement says that S_1 is semantically equivalent to S_2.

Syntactic Equivalence

The difference between *syntactically identical* and *syntactically equivalent* should be interpreted as follows:

- Two services can be stated as syntactically equivalent without knowing the internal structure of them; if the external interfaces are the same, they are syntactically equivalent.
- To state that two services are syntactically identical (which means either the same instance or two instances of the same implementation), it is necessary to also compare their internal structure; if both the internal structure and the external interfaces are the same the two service components are syntactically identical.

If one service is syntactically compatible with another, it shares some of the functionality of the other. Consider two services S_1 and S_2, it is possible to define the following relations between the two services:

Equivalent – S_1 ***E*** S_2
Identical – S_1 ***I*** S_2
Compatible – S_1 ***C*** S_2

Based on these observations, it is possible to define service equivalence, service identicalness and service compatibility:

Equivalence: A service S_1 is equivalent with a service S_2 when they are *semantic* and *syntactic equivalent*. This relation is an equivalence relation (transitive, reflexive and symmetric).

Identicalness: A service S_1 is identical to a service S_2 when they are *syntactically identical* (which implies *semantic equivalence*). This relation is an equivalence relation (transitive, reflexive and symmetric).

Compatibility: A service component S_1 is compatible with a service component S_2, S_1 ***C*** S_2, when they are *semantic equivalent* and S_1 have subsets of functions and interfaces that are equivalent with S_2. This relation is transitive, reflexive and anti-symmetric.

Corollary 1: Two identical services are equivalent.
Corollary 2: Two equivalent services are compatible.
Corollary 3: Two identical services are compatible.

The next challenge is then to determine the semantic and syntactic properties of two services, in order to be able to compare them according to the definitions above. Also, it will be necessary to compare both discrete and composite services according to the defined relations.

d. Service Continuity and SOA
The major components enabling service continuity in mobile services are service logic and service state (see Figure 2). Appropriate service logic must be discovered when the user moves into a new service environment (Location), and service state from the previous service environment should be transferred to the new service logic instance, at least when seamless service continuity is required [9]. The important mechanisms to allow service continuity are thus:

- Service discovery
- State transfer

Service discovery is dependent on a realization of logic handling and employing the service equivalence relations defined in Section 4c. This includes both semantic and syntactic discovery.

It is necessary to define ontologies that describe the service concepts in all service domains that should be used to provide mobile services. These ontologies should be course-grained, but it is also necessary that they are universally agreed upon by all mobile service providers. In GSM, this is not necessary because only a restricted set of agreed upon services are provided. Thus, when a user moves from one provider to another, it is taken for granted that if the user device can communicate using GSM protocols, he is going to access the GSM voice telephony service and the SMS service.

For a service environment based on SOA, where all communications of an unlimited number of services is performed over SOAP, this is not possible; nothing can be assumed about the required service.

After semantic discovery has resulted in some candidate services, a service with an appropriate interface must be selected, if possible. An appropriate interface is defined by having the correct semantics as well as syntax. Thereafter, it might be possible to negotiate non-functional requirements like Quality-of-Service (QoS), to further select the better service instance.

When an appropriate service instance has been located, it will often be necessary to transfer some of the state information from the previous service instance. Such state transfer will be dependent on standardized interfaces in all services that should support service continuity and that are capable of possessing a state. If the two instances are from the same implementation, this could only be a matter of serializing, transferring and de-serializing the state. If the instances are from different implementations, it is much more complicated, if not impossible. It might be necessary to decide if the result from the service discovery process (appropriate service) is more important than keeping the service state (supporting only non-seamless service continuity [9]).

To enable instances of different implementations to be able to exchange states, it will be necessary to define a set of variables that are a sub-set of the global state of a service. For example, one such variable from an Internet browser could be the field containing the current URL (the current web-page is part of the global service state). This variable could then be transferred to the new instance of a different implementation through a standardized interface. It is thus necessary to identify a set of *key state variables* of all services that should be mobile. This should be done on a per service concept basis. This means that a mobile service should instantiate a service concept (defined in an ontology) and this service concept should contain a set of key state variables which are defined as a sub-ontology for this service concept.

In Figure 4 two ontologies are illustrated. The first is an ontology of service concepts (MobileServicesOntology). The second is an ontology of service variables (MobileServicesVariableOntology). A service concept will, if it supports state transfer, include a set of key state variables (KeyStateVariableConcept) taken from the variables ontology. The variables ontology also includes variables used by interfaces in service implementations (InterfaceVariableConcept). These are however not directly connected to the service concept, because it could be possible that two

different implementations of the same service concept have slightly different interfaces (compatible services).

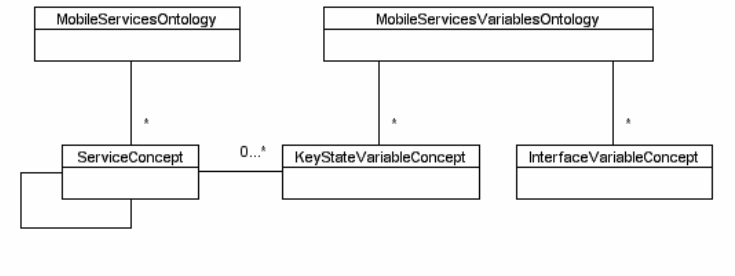

Figure 4: Two ontologies to support mobile services

e. Personalization and SOA

Personalization of services requires that users can receive the same, possibly adapted, service at all times. This implies that service continuity should be kept to a maximum, but there are other additional requirements also. The major components allowing personalization of mobile services are the service profiles and the service content.
Service profiles allow users to specify individual service behavior and look-and-feel (personalization information) and to add their own details (personal information) to a service. Service content is data consumed or produced by the user through service usage. These data also contribute to individualizing services, since user documents are very often individual.

It is thus necessary to add support for this type of information in a SOA that should support mobile services. There are basically two mechanisms to achieve this. First, a similar approach to the state transfer mechanism in the previous section could be applied. But since profiles and content are not as tightly coupled to service logic as the service state, it seems more beneficial and in accordance with the SOA strategy to instead expose service profiles and service content as services on their own. Approaches to making profiles ubiquitously available as services have been investigated earlier in [18].

An approach to exposing service content as a service in itself is therefore required, as depicted in Figure 5 and Figure 6. In Figure 5, each of service components (S_1 to S_3) in the SOA has access to some component specific content (C_1 to C_2). However, one service component can usually not access the content of another (as indicated by the question mark).

In Figure 6, all service components can access the content originally belonging to service component S_2, because this content is exposed as a service on its own in the service-oriented architecture.

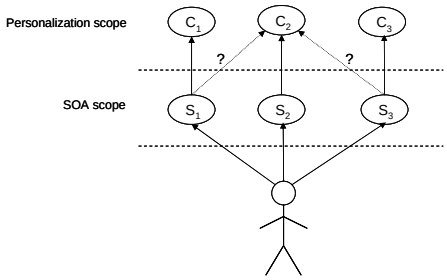

Figure 5: SOA without personalization support

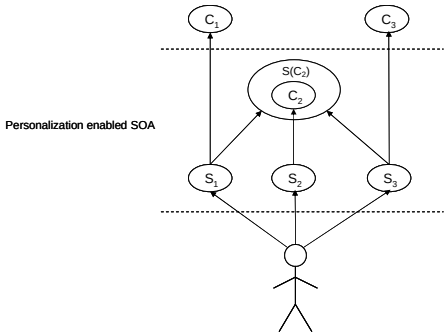

Figure 6: Exposing content as a service on its own

f. A SOA-based Mobile Services Architecture

A SOA-based mobile services architecture builds on, and extends, the basic SOA with several conceptualizations, components and mechanisms:

1. An ontology for describing mobile services concepts
2. An ontology for describing key state variables
3. An ontology for describing interface variables
4. A mobility controller component that can handle state transfer and coordinate the overall mobile service; needs to implement the service equivalence logic defined by relations earlier in this paper
5. A mobility controller stub in each composite service which can coordinate actions towards the mobility controller
6. Standardized interfaces for state transfer (between mobility controller stub and mobility controller)
7. Exposing service profiles as autonomous, self-contained services
8. Exposing service content as autonomous, self-contained services

Figure 7 displays a new view of a SOA that supports mobile services. Since all major components of a mobile service are exposed as services on their own, the only visible component added is the mobility controller. A service consumer locates an adequate service by using the *find* method towards a service directory. Service usage can then be initiated through the *invoke* method. If there is some change in location, or other circumstances require that a service component is interchanged with another, service

usage can be temporary stopped by using the *suspend* method, which is relayed to the mobility controller. This suspend action can be used to transfer the current state (or Key State Variables) for temporary storage in the mobility controller.

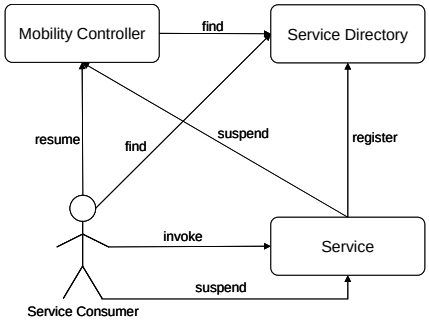

Figure 7: A simplified view of a SOA with support for mobile services

The service consumer can then *resume* the service usage (e.g. in a new location) by contacting the mobility controller, which will locate an adequate service component by looking in the service directory. Not displayed in the figure is the re-initiation of service usage with a possibly new implementation of a service component using the previously stored state (or Key State Variables).

5 Conclusion

This paper has addressed the support for mobile services in Service-Oriented architectures, and in particular the support for service continuity and personalization in such architectures. To add this support, the equivalence of service components was discussed, since it might be required to substitute service components for each other, in order to accommodate various relative movements of entities part of the service domain (e.g. users, devices and services). The concepts, components and mechanisms that must be added to a SOA in order to support mobile services were proposed.

References

[1] Papazoglou, M.P. and Georgakopoulos D. (2003). *Service-Oriented Computing*. Communications of the ACM, Vol. 46, No. 10, October 2003

[2] Cubera, F. et. al. (2003). *The Next Step in Web Services*. Communications of the ACM, Vol. 46, No. 10, October 2003

[3] Object Management Group. (2004). *Common Object Request Broker Architecture: Core Specificaion Version 3.0.3*. OMG, March 2004, Available online: http://www.omg.org/docs/formal/04-03-01.pdf

[4] Horstmann, M. & Kirtland M. (1997). *DCOM Architecture*. Microsoft Developer Network (MSDN), July 23, 1997

[5] Jørstad, I., Dustdar, S., van Do, T. (2004). *Evolution of Mobile Services: An analysis of current architectures with prospect to future*. 2nd International Workshop on Ubiquitous Mobile Information and collaboration Systems (UMICS), co-located with CAiSE 2004, Riga, Latvia, 7-11 June 2004, Springer LNCS. ISBN: 3-540-24100-0

[6] Jørstad, I., van Do, T., Dustdar, S. (2004). *An analysis of service continuity in mobile services*. 2nd International Workshop on Distributed and Mobile collaboration (DMC), WETICE conference, Modena, Italy, 14-16 June 2004, IEEE Computer Society Press. ISBN: 0-7695-2183-5

[7] Jørstad, I., Dustdar, S., van Do, T. (2004). *Service Continuity and Personalisation in Future Mobile Services*. 10th International IFIP Workshop on Advances in Fixed and Mobile Networks (EUNICE 2004), Tampere, Finland, 14-16 June 2004. ISBN: 952-15-1187-7

[8] Jørstad, I., van Do, T., Dustdar, S. (2004). *Personalisation of Future Mobile Services.* 9th International Conference on Intelligence in service delivery Networks, Bordeaux, France, 18-21 October 2004.

[9] Jørstad, I., Dustdar, S., van Do, T. (2004). *Towards Service Continuity for Generic Mobile Services*. The 2004 IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM 04), Bangkok, Thailand, 23-26 November 2004. ISBN: 3-540-23893-X

[10] Barry & Associates Inc. *Service-oriented Architecture (SOA) Definition*. Barry & Associates Inc. Available online: http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html

[11] Colan, M. (2004). *Service-Oriented Architecture expands the vision of Web Services*. IBM. April 21 2004. Available online: http://www-106.ibm.com/developerworks/webservices/library/ws-soaintro.html

[12] Heine, G. (1999). *GSM Networks: Protocols, Terminology and Implementation*. ISBN 0-8900-6471-7, January 1999

[13] World Wide Web Consortium (W3C). *Semantic Web*. Available online: http://www.w3.org/2001/sw/

[14] Lee, T. B. et. al. (2001). *The Semantic Web*. Scientific American, May 2001

[15] Beckett, D. (editor). (2004). *RDF/XML Syntax Specification*. World Wide Web Consortium (W3C), February 10 2004. Available online: http://www.w3.org/TR/rdf-syntax-grammar/

[16] McGuiness, D. & Harmelen, van F. (editors) (2004). *OWL Web Ontology Language Overview.* World Wide Web Consortium (W3C), February 2004. Available online: http://www.w3.org/TR/owl-features/

[17] Martin, D. (editor) (2004). *OWL-S: Semantic Markup for Web Services*. Available online: http://www.daml.org/services/owl-s/1.1/overview/

[18] Hartvigsen, A. M. et. al. (2002). *Offering User Profile as an XML Web Service*. Databases and Information Systems II - The Fifth International Baltic conference on DB and IS, Tallinn, Estonia, June 3-6 2002, Selected Papers, Kluwer Academic Publishers, ISBN 1-4020-1038-9