A SOA-Based Platform-Specific Framework for Context-Aware Mobile Applications*

Laura M. Daniele, Eduardo Silva, Luís Ferreira Pires, and Marten van Sinderen

Centre for Telematics and Information Technology,
University of Twente, Enschede, The Netherlands
{l.m.daniele,e.m.g.silva,l.ferreirapires,
m.j.vansinderen}@ewi.utwente.nl

Abstract. Context-aware mobile applications are intelligent applications that can monitor the user's context and, in case of changes in this context, consequently adapt their behaviour in order to satisfy the user's current needs or anticipate the user's intentions. The design of such applications relies on dynamic middleware platforms that consist of a variety of components. These components are distributed in the environment and interoperate by making use of each other's services. In the A-MUSE project, we defined a design methodology based on MDA principles that relies on a SOA reference architecture for context-aware mobile applications. This paper shows how abstract concepts in the design of such applications can be applied to realize concrete components that guarantee architectural interoperability. We also present a platform-specific framework that uses BPEL, UDDI registry and web services as target technologies to implement our reference architecture.

Keywords: Service-Oriented Architecture, Model-Driven Architecture, context-awareness, BPEL, web services, UDDI.

1 Introduction

Context-aware mobile applications are intelligent applications that can monitor the user's context and, in case of changes in this context, consequently adapt their behaviour in order to satisfy the user's current needs or anticipate the user's intentions. For example, a context-aware mobile phone could be able to know when its user is sitting in a movie theatre and consequently mutes itself without explicit user's intervention. When the user is travelling and dinner time is approaching, the same context-aware mobile phone could suggest a suitable restaurant based on the user's location and his/her previous dining history. Anywhere and anytime, context-aware mobile applications should be able to provide relevant services to their users. The design of such applications relies on dynamic middleware platforms that consist of a variety of components [1,8,11,12]. These components are distributed in the environment and interoperate by making use of each other's services.

* This work is part of the Freeband A-MUSE Project (http://a-muse.freeband.nl). Freeband is sponsored by the Dutch government under contract BSIK 03025.

R. Poler, M. van Sinderen, and R. Sanchis (Eds.): IWEI 2009, LNBIP 38, pp. 25–37, 2009.

In the A-MUSE project, we have defined a middleware platform based on a reference architecture tailored to context-aware mobile applications. This reference architecture includes all the components typically used by such applications. In [4] we have also defined an (automated) design approach based on this reference architecture. This approach refines the monolithic abstract specification of a context-aware mobile application into the distributed behaviour of concrete components that interoperate with each other in order to achieve the goals of the application. This paper aims at showing how the abstract concepts in the design can be mapped to concrete components that guarantee interoperability in our reference architecture, and how these components can be built with specific target technologies. Towards this aim, we have defined and implemented a framework based on specific target technologies that is correct and consistent with the original monolithic abstract specification of our applications. We have made a specific choice on these target technologies, namely, we have used BPEL, UDDI registry and web services. However, our design is platform-independent and can be realized with other specific target implementations.

The structure of the paper is the following: Section 2 introduces the design methodology and reference architecture we have defined in the A-MUSE project for the development of context-aware mobile applications, Section 3 investigates which concrete architectural components are necessary to provide interoperability in the reference architecture and how these components can be built and integrated in a platform-specific framework, Section 4 presents a case study that illustrates how the abstract concepts of our reference architecture can be realized with the concrete components of the platform-specific framework, Section 5 discusses some related work, and Section 6 presents our conclusions and identifies topics for future work.

2 Design Methodology

This section introduces our reference architecture and the design methodology in which this architecture is embedded. The reference architecture has been defined and applied in the A-MUSE project to realize the Live Contacts case study [13,20]. Live Contacts consists of a context-aware mobile application that runs on Pocket PC phones, Smartphones and desktop PCs and allows its users to contact the right person, at the right time, at the right place, via the right communication channel. The reference architecture is general enough to be reused for other context-aware mobile applications by simply redefining some application-specific components, such as context sources and action providers. Moreover, the use of this architecture does not limit our design methodology to context-aware mobile applications, since the same methodology can be applied (with minor adjustments) to other categories of applications based on different reference architectures.

2.1 Reference Architecture

The control component of our reference architecture is the service coordinator, which receives events and triggers actions as reactions to these events. Events may be either *user input events*, which consist of explicit user requests to the application, or *context*

events, which consist of relevant changes in the user context. For example, a user input event may be a request for the user's list of buddies, and a context event may be the proximity event triggered whenever a buddy is nearby the user. Actions represent application reactions to user input and context events, and may be an invocation of any internal or external service, such as the generation of a signal, the delivery of a notification or a web service request.

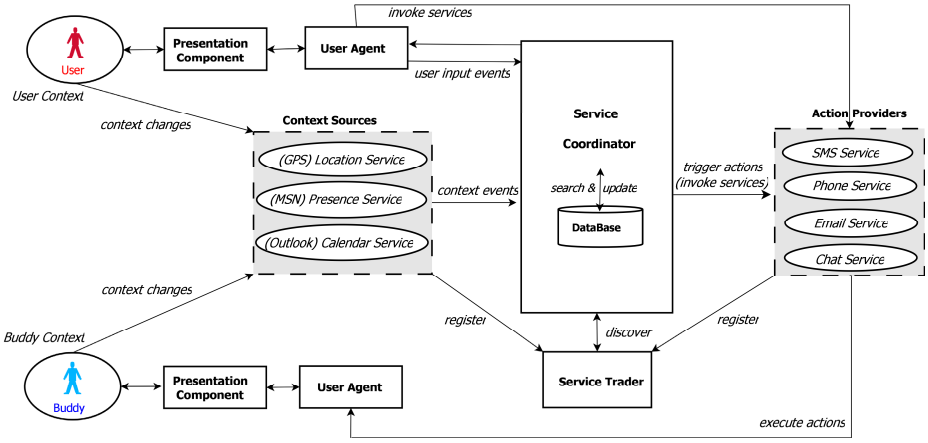

Fig. 1. A-MUSE reference architecture for context-aware mobile applications

Fig. 1 shows a single user instance that interacts with the system and a buddy of this user. The presentation component takes care of the interactions with the end-user and there is one presentation component for each user. In this paper, we do not provide any implementation of this component. The user agent (one for each user and located in the user device) interacts on behalf of the user with the presentation component to obtain user input and present user output, and provides the service coordinator with user input events. The service coordinator orchestrates all the other components, searching and updating a database, which contains information about users (e.g., name, password, preferred contact means and list of buddies). To simplify the discussion without loss of generality, we assume a system configuration with one service coordinator and one database. The service coordinator also interacts with context sources and action providers.

Context sources sense changes in the user context and provides the service coordinator with context events. Fig. 1 shows a (GPS) location service that provides information about users' current location, a (MSN) presence service that provides indications whether users registered in the application are available online in the network, and a (Outlook) calendar service that provides information about users' appointments and activities. We assume that there is one (GPS) location service, one (MSN) presence service and one (Outlook) calendar service for each user agent in this particular configuration. These services are registered in the service trader.

The action providers are responsible for performing actions that follow user input and context events. Fig. 1 shows an SMS service, phone service, e-mail service and chat service, which enable users to communicate with each other through sending messages, making a phone call, sending e-mails or chatting, respectively. These services are also registered in the service trader.

The service trader registers all the available services offered by context sources and action providers. This allows the coordinator to dynamically discover available services based on the service descriptions that are published in the service trader. After discovering the proper service, the coordinator can invoke it by using the endpoint location contained in the service description. Alternatively, the coordinator can forward this endpoint to the user agent, which can directly invoke the service without intervention of the coordinator. This use of a service trader is a well established pattern of service discovery in service-oriented architectures. Examples of service traders in middleware platforms are the OMG CORBA trader [17] and the UDDI registry [15].

The interactions among components of this architecture are based on the service-oriented architecture (SOA) approach, which considers components only from the point of view of the service that they provide or use without considering the internal details of how the service itself is implemented. According to SOA, components make use of each other's services to interoperate in order to support the goals of the application. In this paper, we focus on the right part of Fig. 1, namely on the interactions between the user agent, the coordinator, the database, the service trader and the action providers. Information on the interactions between the coordinator and context sources can be found in [3].

2.2 MDA-Based Methodology

The reference architecture of Fig. 1 has been defined as part of a design methodology based on the Model-Driven Architecture (MDA) approach [16]. Fig. 2 shows this methodology, which divides the design of context-aware mobile applications in different levels of models with different degrees of abstraction and platform-independence. The *service* specification is the highest level of abstraction and describes a context-aware mobile service[1] as a monolithic behaviour from an external perspective. At this level, we specify the functionality that our service offers to its user and we do not consider any structural detail of the service, i.e., we abstract from its internal components. The *platform-independent service design* model describes a context-aware mobile application from an internal perspective revealing our SOA-based reference architecture. The *platform-specific service design* model describes the realization of a context-aware mobile application in terms of specific target technologies. Several alternative Platform-Specific Models (PSMs) may implement a Platform-Independent Model (PIM) as long as correctness and consistency are guaranteed. Therefore, it is in principle possible to use different middleware technologies to realize the platform-specific service design.

[1] The term *service* at this level denotes the observable behaviour of the whole application, as opposed to the use of the term service in service-oriented architectures to denote the functionality supported by a service provider reachable from some middleware.

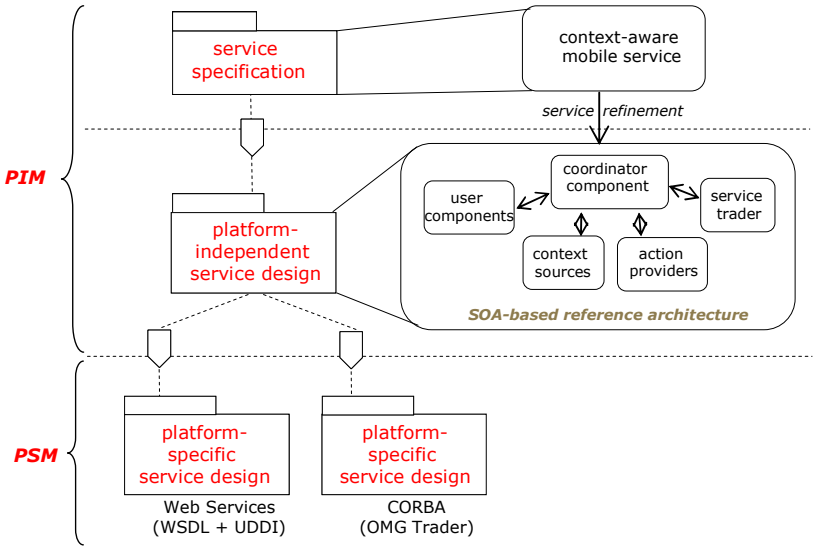

Fig. 2. MDA-based methodology

Our previous work [4,5] focuses on the PIM level of this methodology, namely on the service specification and platform-independent service design model, and the transformations between these models. These transformations consist of gradual (automated) refinements that preserve correctness and consistency particularly of behavioural aspects, which are usually overlooked at the PIM level in the MDA community [14]. This paper focuses on the transformation from the platform-independent to the platform-specific design models and provides an implementation framework for a specific part of the reference architecture, i.e., user agent, coordinator, database, service trader, and action providers. This implementation shows that the PSM level preserves the interoperability that we have designed at the PIM level.

3 Platform-Specific Framework

We consider the following scenario:

"A user wants to contact one of his/her buddies with a specific communication means, such as SMS, phone, chat or e-mail. Therefore, the user provides the application with the name of this buddy and the communication means to be used. In order to fulfil the user request, the coordinator has to retrieve the contact details of the buddy from the buddy list of the user in the database, and discover a proper service in the service trader according to the desired communication means. Once the coordinator has retrieved contact details of the buddy and the endpoint location of the communication service, it can forward this information to the user agent, which is finally able to invoke the proper service and put the user in communication with the desired buddy".

Fig. 3 shows our platform-specific framework for this scenario. In this framework, components of the reference architecture are mapped on target technologies. The same framework can be used with different scenarios. We realized the coordinator as BPEL process exposed as a web service to all the other components of the architecture. These components provide and/or use services, which are orchestrated by the coordinator BPEL process.

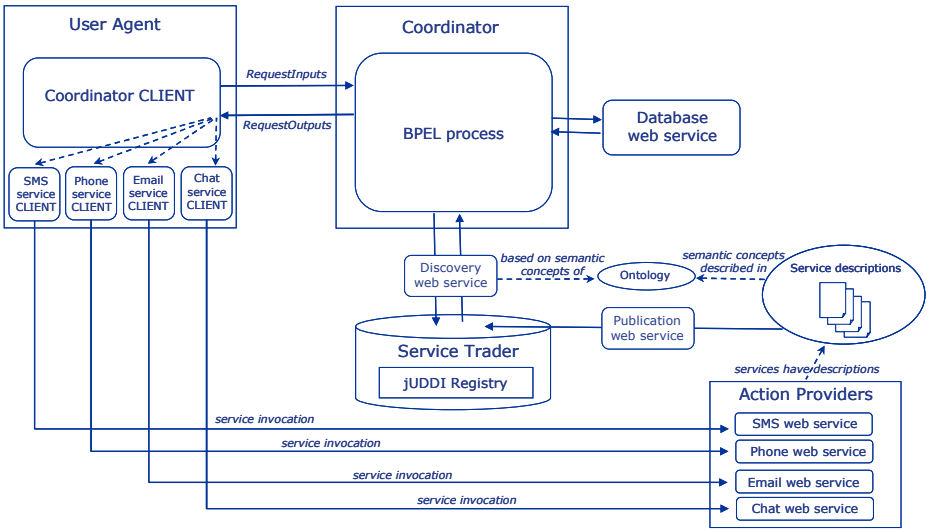

Fig. 3. Platform-specific framework

Fig. 3 shows that the coordinator BPEL process receives some inputs from the *coordinator client* in the user agent (*RequestInputs*). These inputs instantiate a new BPEL process. In the above mentioned scenario, the inputs are the name of the buddy and the preferred communication means to contact this buddy. In order to put the user in contact with his/her buddy, the coordinator BPEL process has to retrieve information from the database component, which is exposed in the framework as a web service (*database web service*). The coordinator BPEL process also needs to discover a suitable service in the *Service Trader* to provide the communication means selected by the user.

We realized the service trader as a UDDI registry using jUDDI [10], which is a Java implementation of the UDDI standard. Our jUDDI registry contains the descriptions of the services available in the framework. In our scenario, the available services are SMS, phone, e-mail and chat services. The service descriptions consist of XML documents with the `name`, `type` and `endpoint` of the service. The service `type` refers to semantic concepts described in an ontology supported by our framework. The `endpoint` is the concrete address where the service is deployed. Fig. 4 shows an example of service description for the SMS service. To support the publication of service descriptions in this format, we have extended the jUDDI with tModels that represent each of the service parameters, i.e., `name`, `type` and `endpoint`. To group the `name`, `type` and `endpoint` tModels under the same service, we have used the categoryBag UDDI element.

```
<ServiceDescription>
  <Name>SMS</Name>
  <Type>http://localhost:8080/ontologies/LiveContacts.owl#SmsService</Type>
  <Endpoint>http://localhost:8080/sms/services/sms</Endpoint>
</ServiceDescription>
```

Fig. 4. SMS service description

Service descriptions are published in our jUDDI registry through the *publication web service* in Fig. 3, which offers a publication interface to the service developers. This interface accepts a service description, parses this description and publishes the service `name`, `type` and `endpoint` in the jUDDI registry.

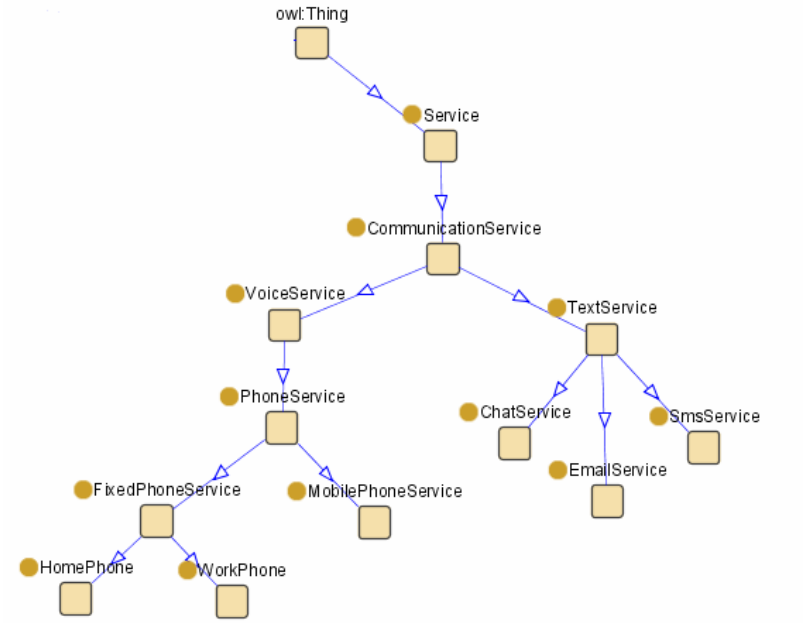

Fig. 5. Framework ontology excerpt

The coordinator BPEL process can discover the services published in the jUDDI registry through the *discovery web service* in Fig. 3. The discovery is based on the service `type` semantic concept, as the one used in the service descriptions. The discovery mechanism retrieves all the services with `type` semantically related to the requested type. For example, assume that we are looking for the service type '`FixedPhoneService`', which is a semantic concept, as shown in the excerpt of the framework ontology depicted in Fig. 5.

The discovery mechanism retrieves the following matches, which are semantically related to the requested type:

i) `FixedPhoneService` ⊂ `PhoneService` (`FixedPhoneService` is a *subsume* match of `PhoneService`)

ii) `FixedPhoneService` ⊃ `WorkPhone` (`FixedPhoneService` is a *plug in* match of `WorkPhone`)

iii) `FixedPhoneService` ⊃ `HomePhone` (`FixedPhoneService` is a *plug in* match of `HomePhone`)

iv) `FixedPhoneService` ≡ `FixedPhoneService` (`FixedPhoneService` is *exact* match of `FixedPhoneService`)

The discovery mechanism selects the *best match* among the options above. The best match is the *exact* match, followed by the *plug in* matches and then by the *subsume* match. The *discovery web service* in Fig. 3 returns the `endpoint` of the best match to the coordinator BPEL process. We realized the publication and discovery mechanisms as web services, so that they are eventually accessible from any component of the framework. The publication and discovery mechanisms are based on the work presented in [19].

The BPEL process finishes once the service endpoint has been discovered in the jUDDI registry and the contact details of the buddy have been retrieved from the database. Endpoint and contact details are given as output to the coordinator client located in the user agent (*RequestOutputs*). Fig. 3 shows that the user agent also contains the clients to invoke the SMS, phone, e-mail and chat services (one client for each service). These are generic clients for the services, i.e., they do not have a specific service endpoint. Provided with the endpoint, the user agent can finally invoke the proper communication service (*service invocation*) and provide this service with the contact details of the buddy in order to finally put the user in contact with his/her buddy via the right communication channel.

We have performed an initial implementation of the presented components, to demonstrate its practical feasibility.

4 Case Study

Fig. 6 shows an example of the platform-independent service design model, which is the result of the behavioural refinements at the PIM level of our methodology. These behavioural refinements are out of the scope of this paper and are presented in [4,5].

Fig. 6 shows part of the functionality of the Live Contacts case study, namely *contactRequest*, which is described in the scenario presented in Section 3. This part of functionality involves several components, which are the user agent, the coordinator, the database, the service trader, and two action providers (the SMS and phone services). Fig. 6 uses ISDL (Interaction System Design Language) [9], which allows the specification of behavioural aspects of interacting components. Particularly, ISDL allows us to specify the control flow of each component in terms of causality relations, and the interactions between components in terms of two contributions, one for each component involved in the interaction.

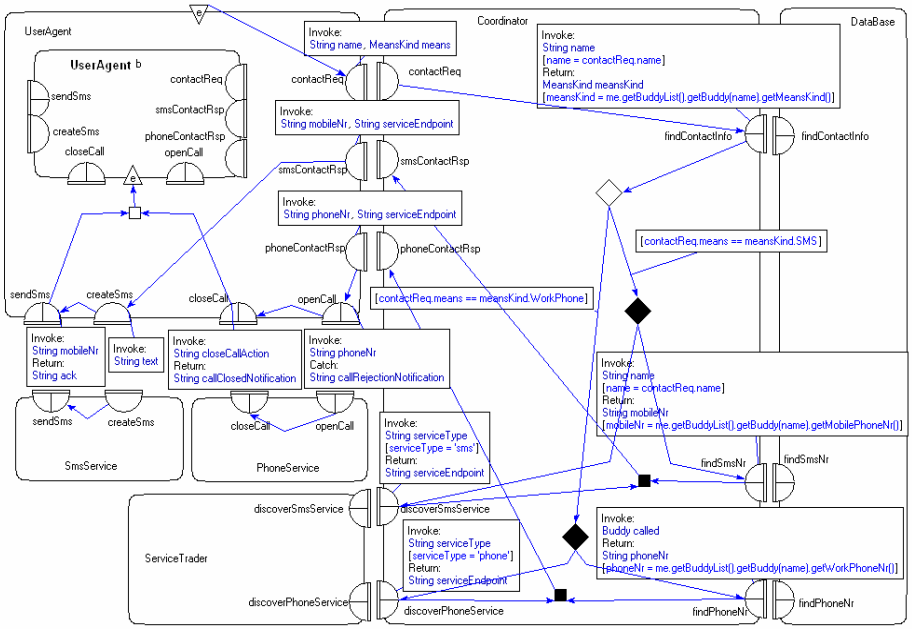

Fig. 6. Platform-independent service design model (exported from Grizzle [7])

Fig. 6 shows that the user request to contact a buddy with a specific communication means (*contactReq*) is forwarded by the user agent to the coordinator. This request contains two parameters, which are the name of the buddy (*name*) and the communication means to contact this buddy (*means*). The coordinator retrieves from the database the communication means available for the buddy (*findContactInfo*). Afterwards, the coordinator evaluates the parameters of the contact request. Depending on the means selected by the user (*SMS* or *WorkPhone*), a proper communication channel is selected (*SMS* or *phone*). In both cases, the coordinator performs two activities concurrently, namely, retrieving from the database the number where to contact the buddy (*findSmsNr* or *findPhoneNr*), and asking the service trader to discover the proper service to contact the buddy (*discoverSmsService* or *discoverPhoneService*). In the discovery, the coordinator indicates the service type to dicover (*sms* or *phone*), and the service trader returns the endpoint of this service (*serviceEndpoint*). Once both the service discovery and the database retrieval are concluded, the coordinator sends a response to the user agent (*smsContactRsp* or *phoneContactRsp*) with the information necessary to invoke the service, i.e., the contact details of the buddy (*mobileNr* or *phoneNr*) and the endpoint location of the service (*serviceEndpoint*). In this way, the user agent is able to invoke the proper action provider (*SmsService* or *PhoneService*) and provide it with the necessary input, which may be the mobile number or the work phone number of the buddy. We assume here that all the services published in the service trader with *serviceType* = *'sms'* present the same behaviour as *SmsService* in Fig. 6. Analogously, all the services published in the service trader with *serviceType* = *'phone'* present the same behaviour as *PhoneService* in Fig. 6.

We realized a prototype based on the platform-independent service design model of Fig. 6 by using the platform-specific framework described in Section 3. We experimented and tested this prototype. Fig. 7 shows the BPEL process that implements the coordinator, which orchestrates all the components of our platform-specific framework.

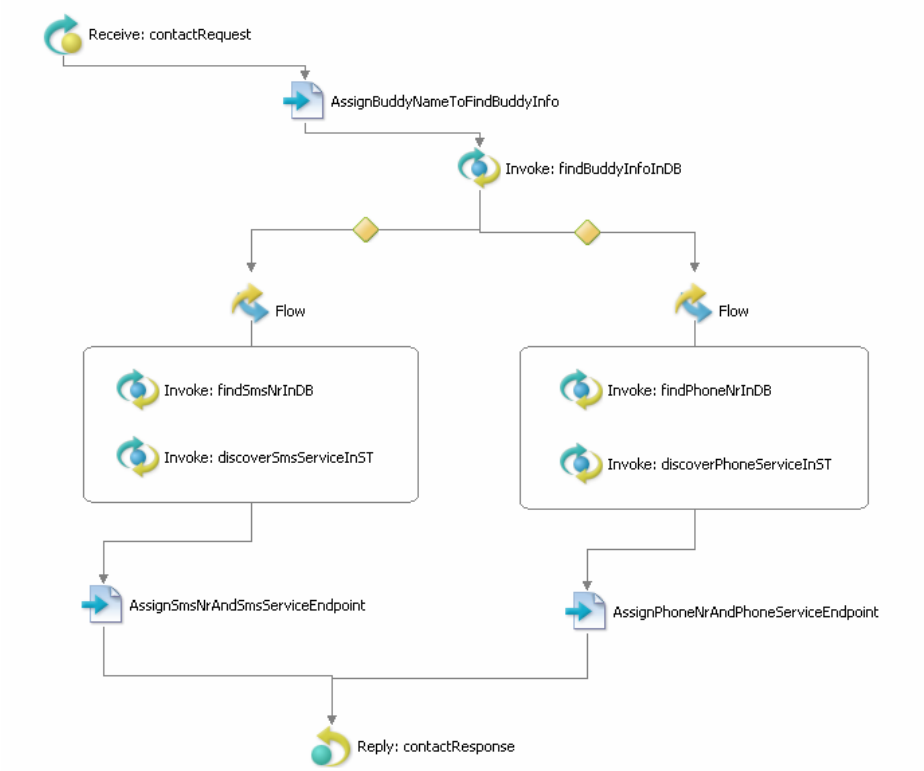

Fig. 7. Platform-specific service design model: the coordinator BPEL process

The BPEL process starts with a *receive* activity (*contactRequest*) that accepts as inputs the name of the buddy and the communication means to contact the buddy. The *assign* activity *AssignBuddyNameToFindBuddyInfo* copies the name of the buddy of the *contactRequest* activity to the *invoke* activity called *findBuddyInfoInDB*. This latter activity consists of an invocation of the database web service in order to retrieve the communication means available for the buddy. The BPEL process in Fig. 7 continues in two alternative flows, one in case the selected communication means is '*SMS*', and the other one in case it is '*WorkPhone*'. These flows execute two *invoke* activities in parallel: the invocation of the database service to retrieve the contact details of the buddy, and the invocation of the discovery web service to discover the endpoint of the service. When both *invoke* activities in the flow are concluded, their output is assigned to the *reply* activity (*contactResponse*) that ends the BPEL process.

The *contactResponse* activity sends the outputs of the process to the coordinator client in the user agent.

5 Related Work

Much effort has been done to develop SOA-based middleware solutions for context-aware services and applications [1,8,11,12]. The benefits of using SOA to support the development of such applications have been extensively discussed in the literature [2,21]. In [21], the convergence of context-awareness and service-orientation in ubiquitous computing is discussed by comparing context-awareness principles, such as *adaptation* and *extension*, to SOA principles, such as *abstraction* and *loosely coupling*. Particularly, it is shown how abstraction and loosely coupling principles in SOA support, respectively, adaptation and extension principles in context-awareness.

In [2], service-oriented context-aware application design is discussed and a service-oriented architecture that separates context parameters from application data is proposed. Although this architecture reflects the need to distinguish components devoted to context management and application core in the design of context-aware services, [2] does not describe a design process that supports this architecture. In contrast, we present a SOA-based reference architecture for context-aware mobile applications that is embedded in a comprehensive design methodology that supports the architecture.

Our design methodology is based on the MDA principles and addresses behavioural issues of model transformations in the design of the applications. These behavioural issues are usually overlooked in common MDA approaches [14]. In this paper, we show that behavioural aspects, which we have addressed already at the Platform-Independent Model (PIM) level, can be consistently realized at the Platform-Specific Model (PSM) level without any need to incorporate them later in the development process, by adding hand-written code as annotations to PSMs or to implementation code skeletons.

6 Conclusions and Future Work

This paper presented a prototype of a platform-specific framework for the realization of context-aware mobile applications. This prototype is one of the possible realizations with target technologies of a platform-independent model obtained through gradual behaviour model transformations of an abstract service specification. This paper shows the feasibility of this prototype. The prototype actually reflects the interoperability among components that we modelled in our design. Therefore, we can conclude that the transformation from platform-independent level to platform-specific level preserves correctness and consistency of the original behaviour of the application. However, this is only a first step towards the validation of our methodology and further work needs to be done to validate the complete design and implementation.

In this paper, we do not discuss the transformation from the platform-independent model in Fig. 6 to the platform-specific model in Fig. 7. We only provide the source and target models of this transformation. The mapping from ISDL to BPEL is part of the work presented in [6,18].

We realized only a limited part of the functionality of the Live Contacts case study, in which the coordinator handles one of the possible user request to the application. However, in the complete case study the coordinator has to handle several user requests and context events at the same time. These requests and events are realized by interacting components with different but interdependent executions threads. Therefore, the coordinator has to handle concurrency and synchronization issues of interacting components. This is part of the work presented in [5].

We provided a feasible implementation of part of our reference architecture for context-aware mobile applications. We did not consider here context source components that retrieve context information from the user environment and provide the coordinator with context events in case of changes in this context. The integration of these components in our reference architecture using a context expression evaluator is discussed in [3]. However, we envision an alternative realization of these components with web services technologies. In this work, by implementing the action providers as web services, we learned that this is a feasible and interesting solution to guarantee flexibility, interoperability and portability in our platform-specific framework. Further study needs to be performed in order to integrate context source components in the framework and expose them as web services. These components require mechanisms to allow the coordinator to dynamically subscribe to context events as soon as these components become available to the application. However, we believe that the experiments we have performed in this paper by building action providers as web services, have brought us a step forward towards the realization of context sources with this technology.

References

1. Bai, Y., Ji, H., Han, Q., Huang, J., Qian, D.: MidCASE: A Service Oriented Middleware Enabling Context Awareness for Smart Environment. In: International Conference on Multimedia and Ubiquitous Engineering (MUE 2007), pp. 946–995. IEEE Computer Society Press, Los Alamitos (2007)
2. Chaari, T., Laforest, F., Celentano, A.: Service-Oriented Context-Aware Application Design. In: First International Workshop on Managing Context Information in Mobile and Pervasive Environments (MCMP 2005), Cyprus (2005)
3. Daniele, L., Ferreira Pires, L., van Sinderen, M.: Context Handling in a SOA Infrastructure for Context-Aware Applications. In: Proceedings of the 2nd International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC 2008), Porto, Portugal, July 2008, pp. 27–37. INSTICC Press (2008)
4. Daniele, L., Ferreira Pires, L., van Sinderen, M.: An MDA-Based Approach for Behaviour Modelling of Context-Aware Mobile Applications. In: Paige, R., Hartman, A., Rensink, A. (eds.) ECMDA-FA 2009, Enschede, The Netherlands, June 2009. LNCS, vol. 5562, pp. 206–220. Springer, Heidelberg (2009)
5. Daniele, L., Ferreira Pires, L., van Sinderen, M.: Ferreira Pires. L., van Sinderen, M.: Towards Automatic Behaviour Synthesis of a Coordinator Component for Context-Aware Mobile Applications. In: Proceedings of the International Workshop on Mobile Technologies in Enterprise Computing Systems (MTECS 2009), Auckland, New Zealand, September 2009. IEEE Computer Society Press, Los Alamitos (2009)

6. Dirgahayu, T., Quartel, D., van Sinderen, M.: Development of Transformations from Business Process Models to Implementations by Reuse. In: Proceedings of the 3th International Workshop on Model-Driven Enterprise Information Systems (MDEIS 2007), Portugal, June 2007, pp. 41–50. INSTICC Press (2007)
7. Grizzle home, http://isdl.ctit.utwente.nl/tools/grizzle
8. Gu, T., Pung, H.K., Zhang, D.Q.: A Service-Oriented Middleware for Building Context-Aware Services. Journal of Network and Computer Applications (JNCA) 28(1) (2005)
9. ISDL home, http://isdl.ctit.utwente.nl
10. jUDDI home, http://ws.apache.org/juddi
11. Kiani, S.L., Riaz, M., Sungyoung, L., Young-Koo, L.: Context Awareness in Large Scale Ubiquitous Environments with a Service-Oriented Distributed Middleware Approach. In: 4th Annual ACIS International Conference on Computer and Information Science (ICIS 2005), pp. 513–518. IEEE Computer Society Press, Los Alamitos (2005)
12. Kim, E., Choi, J.: A Context-Awareness Middleware Based on Service-Oriented Architecture. In: Indulska, J., Ma, J., Yang, L.T., Ungerer, T., Cao, J. (eds.) UIC 2007. LNCS, vol. 4611, pp. 953–962. Springer, Heidelberg (2007)
13. Live Contacts home, http://livecontacts.telin.nl
14. McNeile, A., Simons, N.: Methods of Behaviour Modelling: A Commentary on Behaviour Modelling Techniques for MDA. Metamaxim Ltd home, http://www.metamaxim.com/download/documents/Methods.pdf
15. OASIS: OASIS-Committes- OASIS UDDI Specifications TC, http://oasis-open.org/commitees/uddi-spec/doc/tcspecs.htm
16. Object Management Group: MDA-Guide, Version 1.0.1, omg/03-06-01 (2003)
17. Object Management Group: Trading Object Service Specification, Version 1.0, formal/00-06-27 (2000)
18. Quartel, D., Dirgahayu, T., van Sinderen, M.: Model-Driven Design, Simulation and Implementation of Service Compositions in COSMO. Int. J. of Business Process Integration and Management (to appear)
19. Silva, E., Martínez López, J., Ferreira Pires, L., van Sinderen, M.: Defining and Prototyping a Life-cycle for Dynamic Service Composition. In: Proceedings of the 2nd Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC 2008), Porto, Portugal, July 2008, pp. 79–90. INSTICC Press (2008)
20. Ter Hofte, G.H., Otte, R.A.A., Kruse, H.C.J., Snijders, M.: Context-Aware Communication with Live Contacts. In: Conference Supplement of Computer Supported Cooperative Work (CSCW 2004), Chicago, USA (November 2004)
21. Yoon, H.: A Convergence of Context-Awareness and Service-Orientation in Ubiquitous Computing. International Journal of Computer Science and Network Security (IJCSNS) 7(3), 253–257 (2007)