Interaction Patterns for Refining Behaviour Specifications of Context-Aware Mobile Services*

Laura Daniele, Luís Ferreira Pires, and Marten van Sinderen

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

Abstract. In the context of Model-Driven Architecture (MDA), little attention has been given to behavioural aspects of service design. This paper proposes a MDA-based approach that considers these aspects in the development of context-aware mobile services. Starting from the specification of the external observable behaviour of a service, we gradually refine this behaviour considering the internal structure of the service. Particularly, we present a structure that is general enough to be used for several context-aware mobile services. However, it may be configured based on the specific service to be developed. An important step of our approach consists of identifying sequences of interactions, which we call interaction patterns, which can be mapped into a behaviour model of the components that execute the service. This model is platform-independent and may be realized in terms of several specific target technologies.

1 Introduction

Context-aware mobile services are capable to sense changes in the user's environment and consequently adjust their behaviour in order to provide relevant functionality to their user anywhere and at anytime. The design of such services is a challenging task, which has justified the development of novel methods, abstraction and infrastructures [19,20,21,22]. Moreover, the complexity, diversity and fast-changing nature of enabling technology platforms require design approaches that shield designers from platform-specific details in order to concentrate their efforts on the services to be developed [3]. A valuable alternative to face the challenge of designing such services consists of using the OMG's Model Driven Architecture (MDA) [1], which aims at facilitating service design through the separation of platform-independent (PIMs) and platform-specific models (PSMs), and the use of model transformations.

Although considerable effort has been done in MDA to define technologies and techniques for designing services, behavioural aspects of design are still neglected. In order to address these behavioural aspects, we have developed within the A-MUSE

* 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.

project [2] an MDA-based design process, which is described in [3,10]. The process defines three different levels of models with different degrees of abstraction and platform-independence. The highest level of abstraction, which we call service specification, specifies the external observable behaviour of a context-aware mobile service. The intermediate level, which we call platform-independent service design, describes the service in terms of its internal structure and the interactions between components within this structure. The lowest level, which we call platform-specific service design, describes the realization of the service in terms of specific target technologies.

This paper focuses on model transformations between the behaviour specification and the platform-independent design of context-aware mobile services, given a specific structure for these services. This structure is general enough to be used for several context-aware mobile services. However, it may be configured based on the specific service to be developed. Particularly, we propose a systematic approach for this model transformation that is based on refinements and interaction patterns. This approach should allow us to guarantee correctness and consistency between the behaviour service specification and the platform-independent design model that reveals the service internal structure.

The structure of the paper is the following: Section 2 presents an overview of the A-MUSE design process, Section 3 describes our approach to realize the model transformation between service specification and platform-independent design levels, Sections 4 and 5 discuss the service specification level and the platform-independent service design level in further detail by using a case study, Section 6 discusses some related work in the context of MDA, and Section 7 presents our conclusions and identifies topics for future work.

2 A-MUSE Design Process

According to MDA principles, the A-MUSE design process divides the design of context-aware mobile services in different levels of models with different degrees of abstraction and platform-independence. Fig. 1 shows this process [3,10].

The service specification level describes the behaviour of a context-aware mobile service 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., its internal components. Towards this aim, we use a language specially developed in the A-MUSE project, which is called A-MUSE Domain-Specific Language (DSL). The A-MUSE DSL supports the specification of context-aware mobile services at a high abstraction level during the initial phase of the service design process. This language allows us to specify the behavioural aspects of our service in terms of user inputs, user outputs, data actions, and their causality relations.

The platform-independent service design level describes a context-aware mobile service from an internal perspective. At this level, we refine the service specification while considering a (given) internal structure for our service. Fig. 1 shows that this structure consists of context sources, action providers, coordination component, service trader and user components. Context sources sense events in the user's

environment and provide these events to the coordination component, which as a consequence triggers actions that are executed by action providers. Context sources and action providers are registered in the service trader in order to be dynamically available to the coordination component. Each user accesses the service through a user component, which provides the user interface and forwards requests to the coordination component [3]. The core of our structure consists of the coordination component, since it orchestrates all the internal interactions. These interactions are: (i) user requests from user components, (ii) context events from context sources, (iii) actions to be executed by action providers, and (iv) resources registered in the service trader. At the platform-independent service design level, we use the A-MUSE DSL for the refinements, the Interaction System Design Language (ISDL) [4,5] for specifying component behaviours and interactions between components, and UML class diagrams for the data models.

The platform-specific service design level describes the realization of a context-aware mobile service in terms of specific target technologies. Since the relation between the platform-independent service design and the platform-specific service design levels is flexible, it is in principle possible to use different middleware technologies, such as, for example, web services or CORBA, as it is shown in Fig. 1.

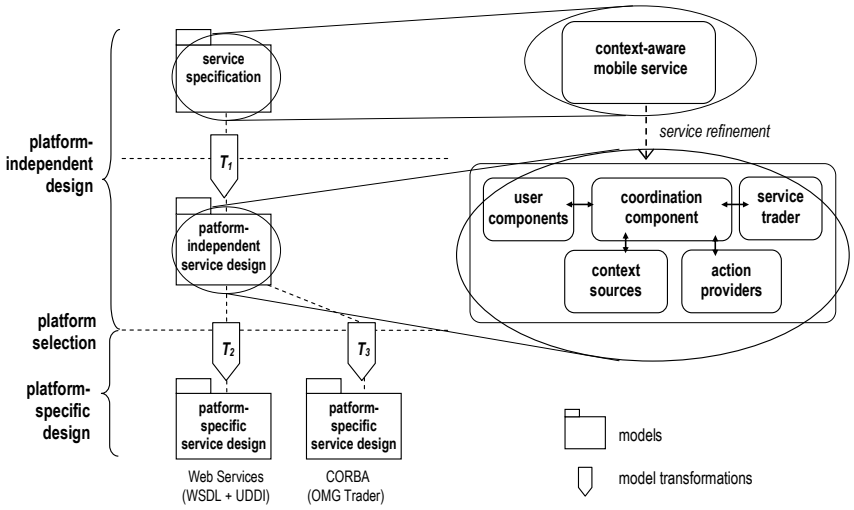

Fig. 1. Design process with different levels of abstraction and platform-independence

The rest of this paper focuses on the first two levels of the design process shown in Fig. 1, namely the service specification level and the platform-independent service design level.

3 Approach

The aim of this work consists of defining a platform-independent model that specifies the behaviour of the components that realize the service. This model, which preserves the service behaviour specification, should be used as input to the platform-specific service design in order to concretely realize these components in terms of specific target technologies. Fig. 2 shows the approach we have taken towards this aim.

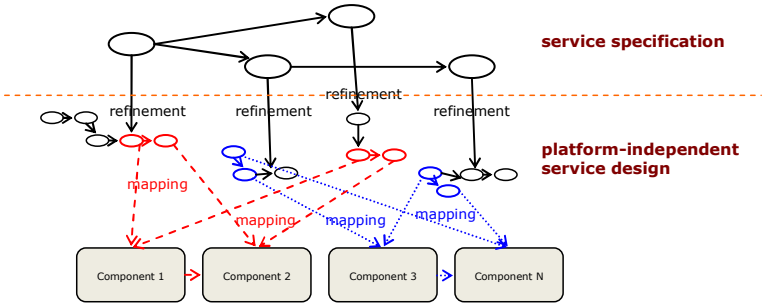

Fig. 2. Refinement of behaviour specifications by using interaction patterns: approach

The starting step consists of the service specification, in which we define the functionality offered to the user in terms of actions and causality relations between these actions. The actions at this level are too abstract to be directly realized with platform-specific technologies. Therefore, the second step consists of defining a platform-independent service design model in which we refine these actions into multiple actions that can be directly supported by the realization platform. While doing these refinements, we consider the internal structure of our service in order to create a correspondence between refined actions and components that performs these actions. Fig. 2 shows the refinement of actions at the service specification level into multiple actions at the platform-independent level.

A further step in our approach consists of comparing all the refinements in order to identify sequences of actions, which we call *interaction patterns*, recurring in several refinements. Since the concept of interaction pattern is extremely important in our approach and this concept differs from the one is commonly used in the literature [6], we give here the following definition:

"When designing a service and considering its internal perspective, we define an interaction pattern as a recurring sequence of actions performed by two (or more) components interacting to each other". Fig. 2 shows two interaction patterns represented with dash and dot lines.

Once we have identified interaction patterns in the refinements, the last step of the platform-independent service design consists of mapping these patterns into the behaviour of components. In our example, these components are the coordination component, context sources and action providers, as presented in Section 2. Since an interaction pattern involves two (or more) components, the mappings must create a

proper correspondence from refined actions of the pattern to components, and guarantee the correct functioning of the interacting components.

4 Service Specification

This section introduces a more detailed view on the steps performed at the service specification level and presents examples of the specification by using a case study. This case study is called Live Contacts [7] and consists of a context-aware mobile service running on Pocket PC phones, Smartphones, desktop PCs that allows its users to contact the right person, at the right time, at the right place, via the right communication channel. Further information about specific functionality that Live Contacts offers to its user can be found in [8].

4.1 Models

In order to guarantee consistency between service specification and service design levels, we have produced two models for the service specification. The first model has been defined from a "pure" user perspective, in which we describe the functionality that our service offers to the user in terms of user inputs and outputs. We consider the service from an external view without any internal detail and the user is only aware of what he/she can ask to the service and, eventually, get back from the service. Fig. 3 presents an example of this model within the Live Contacts case study. The Grizzle tool [9] has been used for model editing and simulation of service specifications.

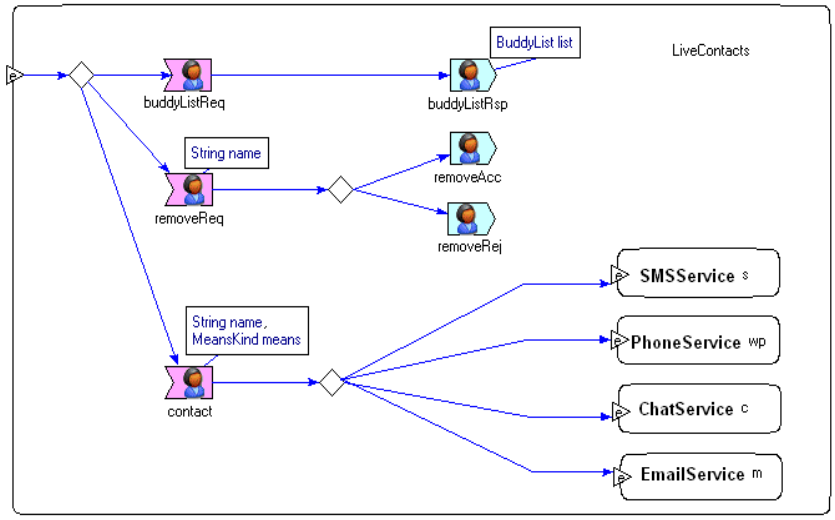

Fig. 3. Service specification: example of first model (exported from Grizzle [9])

Fig. 3 shows that a user may ask information about all his/her contacts (*buddyListReq*) and receive back a list containing this information (*buddyListRsp*). The user may also remove a contact from his/her list (*removeReq*) and receive a confirmation (*removeAcc*) or a rejection (*removeRe*j) for this request. Moreover, the user may contact one of his/her buddies by selecting a specific means of communication, such as SMS, phone, chat or e-mail.

The second model we have produced for the service specification adds to the user perspective a global view on the status information handled by the service to provide the user with the functionality of the previous model. However, internal components of the system are not introduced at this point. This second model, which is a refinement of the previous model, consists of an intermediate step towards the platform-independent design. Fig. 4 presents an example of this second model within the Live Contacts case study.

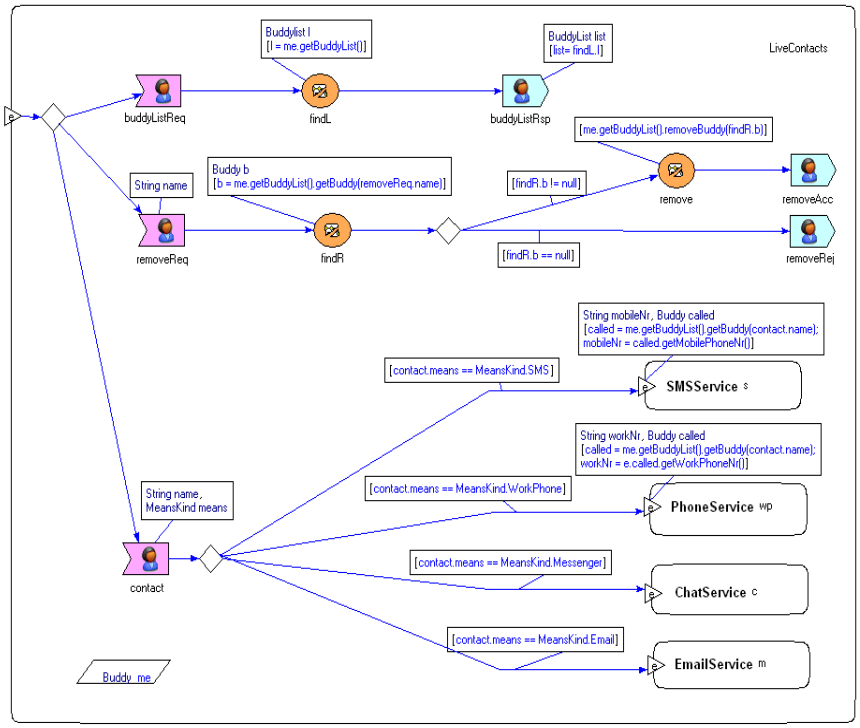

Fig. 4. Service specification: example of second model

Fig. 4 shows the status information that the Live Contacts service uses to handle user requests. For example, when the user wants to remove a buddy from his/her list, the Live Contacts service first selects this buddy from the user's buddy list (*findR*). Afterwards, if the buddy is in the user's list (*findR.b != null*), the list is updated by

removing the buddy (*remove*), otherwise (*findR.b == null*) the user request is rejected.

4.2 Interactions Classification

In the service specification we have defined the external behaviour of a context-aware mobile service. This external behaviour presents some general-purpose characteristics that are common to context-aware mobile service independent of the specific service to be developed. We decided to classify these general-purpose characteristics based on the kind of interaction that they imply. For example, all context-aware mobile services are characterized by the capability to retrieve context information from the user context and provide relevant functionality to their user based on this information. This implies two kinds of interactions, namely an interaction with context sources in order to retrieve context information, and an interaction with action providers in order to provide the relevant functionality to the user. By identifying these interactions, we can relate the total functionality provided by our service to a classification, which is general enough to be (re)used with a wide range of context-aware mobile services regardless of the specific service to be realized. Particularly, we have identified the following set of basic interactions:

1. *simple* → it provides a way to receive input from the user and, eventually, present output to the user;
2. *search* → it provides a way to retrieve information stored in some knowledge source of the service;
3. *update* → it provides a way to update information stored in some knowledge source of the service;
4. *context* → it provides a way to retrieve context information from the user context;
5. *invocation* → it provides a way to invoke external services;
6. *discovery* → it provides a way to discover external services that are available in a certain place and at a certain moment.

We can combine these basic interactions to define more complex interactions and mark the second model of our service specification as a preparation for the transformation to the platform-independent service design model. For example, considering the Live Contacts case study, we can mark the model of Fig. 4 as follows:

- *buddyList* → *simple* + *search*, since the user asks for his/her buddy list (*simple*) and Live Contacts retrieves the list that is stored in some information source (*search*).
- *removeBuddy* → *simple* + *search* + *update*, since the user asks to remove a buddy from his/her buddy list (*simple*), Live Contacts retrieves this buddy from the information source (*search*) and, eventually, removes this buddy from the list (*update*).
- *contactBuddy* → *simple* + *search* + *discovery* + *invocation*, since the user asks to contact a buddy (*simple*), Live Contacts retrieves this buddy's number from the information source (*search*) in case the means of communication is SMS or work phone, discovers the proper service to open the communication (*discovery*) and, finally, invokes this service (*invocation*).

5 Platform-Independent Service Design

This section introduces a more detailed view on the steps performed at the platform-independent service design level and presents examples of refinements and mappings from the Live Contacts case study. The Grizzle tool [9] has been used for model editing and simulation of platform-independent service design.

5.1 Refinements

At the platform-independent service design level we refine the service specification while considering the internal structure of the service that is depicted in Fig. 1. Although this structure considers all the components typically involved in context-aware mobile services, in order to realize correct refinements we need to define this structure in more detail. Fig. 5 shows the specific structure that we use in the Live Contacts case study. This structure has been defined within the A-MUSE project to realize the Live Contacts case. However, it can be reused for other context-aware mobile services by simply redefining context sources and action providers that are specific to that service.

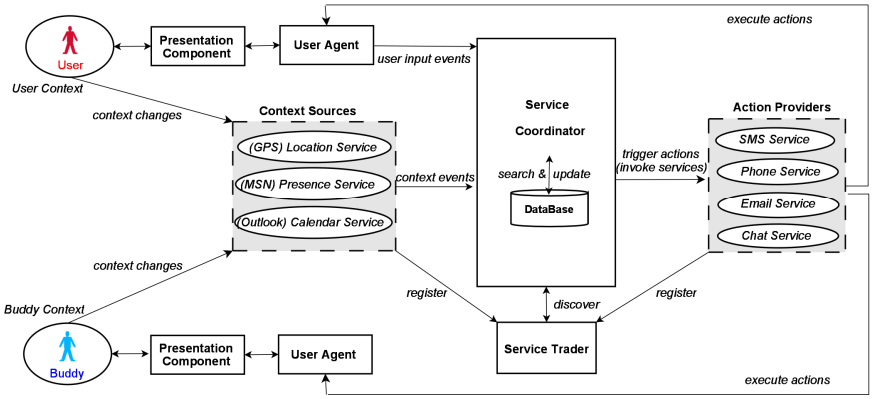

Fig. 5. Service structure

The presentation component takes care of the interactions with the end-user. There is one presentation component for each user. The user agent acts on behalf of the user with the other components. There is one user agent for each user. Particularly, the user agent interacts with the presentation component in order to obtain user input and present user output, and provides the service coordinator with user input events. The service coordinator takes care of orchestrating the other components, searching and updating the database, which contains information about users (e.g., name, password, preferred means of contacts and list of buddies), and login information. There is one

service coordinator and one database. The service coordinator also interacts with context sources and action providers.

The context sources sense changes in the user context and provides the service coordinator with context events. Particularly, Fig. 5 shows the (GPS) location service that provides information about a user's current location, the (MSN) presence service that provides indications whether users registered in the Live Contacts application are available online in the network, and the (Outlook) calendar service that provides calendar information. There is one (GPS) location service, one (MSN) presence service and one (Outlook) calendar service for each user agent. These services are registered in the service trader.

The action providers are responsible for performing actions triggered by the service coordinator. Actions represent application reactions to user input events and context events. Particularly, Fig. 5 shows the SMS service, phone service, e-mail service and chat service, which enable users to communicate with each other through, respectively, sending messages, making a phone call, sending e-mails or chatting. There is one SMS service, phone service, e-mail service and chat service for each user agent. These services are registered in the service trader. The service trader registers all the available context sources and action providers in order to allow the coordinator to discover and invoke them.

Fig. 6 presents an example of refinements within the Live Contacts case study revealing the structure mentioned above.

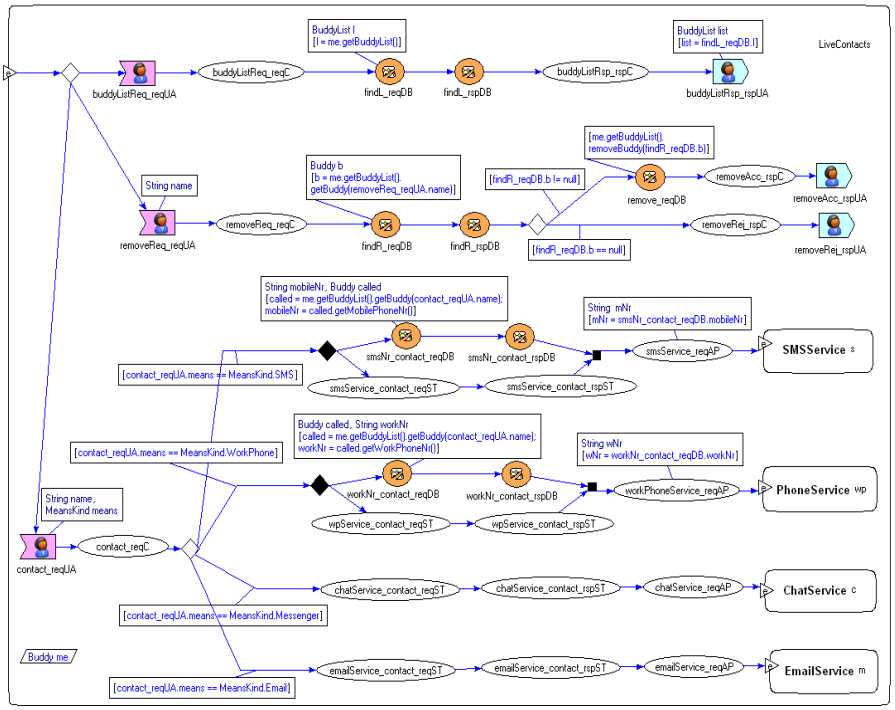

Fig. 6. Service design: example of refinements

As depicted in Fig. 6, the user request to get the buddy list arrives to the user agent (*buddyListReq_reqUA*), which forwards the request to the coordinator (*buddyListReq_reqC*). The coordinator retrieves the list from the database (*findL_reqDB* and *findL_rspDB*) and sends the list to the user agent (*buddyListRsp_rspC*). Finally, the user agent sends the list to be presented to the user (*buddyListRsp_rspUA*).

The user request to remove a buddy arrives to the user agent (*removeReq_reqUA*), which forwards the request to the coordinator (*removeReq_reqC*). The coordinator checks the buddy list of the user whether the buddy is in the contacts list *(findR_reqDB* and *findR_rspDB*). If this is the case, the coordinator removes the buddy from the list (*remove_reqDB*) and sends a positive response to the user agent (*removeAcc_rspC*), which presents the result to the user (*removeAcc_rspUA*). If the buddy is not in the list of the user, the coordinator sends a negative response to the user agent (*removeRej_rspC*), which presents the result to the user (*removeRej_rspUA*).

The user request to contact a buddy arrives to the user agent (*contact_reqUA*). The user agent forwards the request to the coordinator (*contact_reqC*), which evaluates the parameters of the contact request. Particularly, depending on the contact means selected by the user, the coordinator selects one of the options depicted in Fig. 6. If the contact means consists of SMS or work phone, the coordinator performs two activities concurrently, namely, retrieving from the database the number to contact the buddy (*<numberType>_contact_reqDB* and *<numberType>_contact_rspDB*) and asking the service trader to discover the proper service to contact the buddy *(<serviceType>_contact_reqST* and *<serviceType>_contact_reqST*). If the contact means consists of chat service or e-mail service, the coordinator only asks the service trader to discover the proper service to contact the buddy. For each option, the coordinator is finally able to invoke the proper action provider *(<serviceName>_reqAP)*, which opens the communication with the user agent of the buddy to be contacted.

5.2 Interaction Patterns and Mapping

We have identified interaction patterns that often recur in the refinements. Particularly, we have created a correspondence between these interaction patterns and the basic interactions that we have classified in Section 4.2. Table 1 shows this correspondence for our case study.

The ultimate result of the transformation from service specification to platform-independent service design consists of a model in which we map the interaction patterns specified in the refinements into the behaviour of the specific components responsible of realizing these patterns. This ultimate result is used as input to the last level of our design process (see Fig. 1), namely the platform-specific service design, in order to concretely realize components in terms of specific target technologies. Since the refinements explicitly specify which component performs a certain action, the mapping of interaction patterns to the behaviour model of a specific component is straightforward. However, the interoperability between components interacting with

each other to provide certain functionality is not straightforwardly mapped into the behavior model of individual components. Therefore, it is necessary to consistently guarantee this interoperability in the mapping. Fig. 7 presents an example of mapping within the Live Contacts case study.

Table 1. Interaction patterns for the Live Contacts service

	BuddyList	RemoveBuddy	ContactBuddy
simple	buddyListReq_reqUA buddyListReq _reqC buddyListRsp_rspC buddyListRsp_rspUA	removeReq_reqUA removeReq _reqC removeAcc_rspC *OR* removeRej_rspC removeAcc _rspUA *OR* removeRej _rspUA	contact_reqUA contact _reqC
search	findL_reqDB findL_rspDB	findR_reqDB findR_rspDB	smsNr_contact_reqDB *OR* workNr_contact_reqDB smsNr_contact_rspDB *OR* workNr_contact_rspDB
update		remove_reqDB	
discovery			smsService_contact_reqST *OR* wpService_contact_reqST *OR* chatService_contact_reqST *OR* emailService_contact_reqST smsService_contact_rspST *OR* wpService_contact_rspST *OR* chatService_contact_rspST *OR* emailService_contact_rspST
invocation			smsService_reqAP *OR* wpService_reqAP *OR* chatService_reqAP *OR* emailService_reqAP

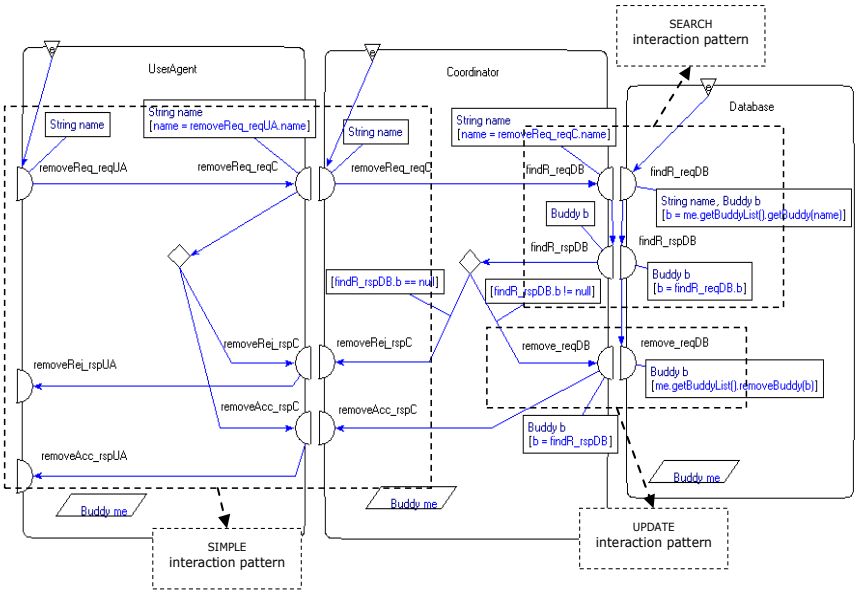

Fig. 7. Service design: example of mapping (exported from Grizzle [9])

Consider the refinement in Fig. 6 that describes the Live Contacts functionality of removing a buddy from the list of a user. This refinement involves three components,

which are the user agent, the service coordinator and the database where the list is stored. Fig. 7 depicts the behaviour model of these components considering their mutual interactions. Particularly, dashed lines show the mapping of interaction patterns of Table 1 (*RemoveBuddy* column) into this model.

6 Related Work

We consider here some related work in the literature that deals with the design of context-aware services and applications [3,10,11,12]. In [12], service-oriented context-aware design is discussed, and a service-oriented structure that separates context parameters from application data is proposed. Although this structure reflects the need to distinguish components devoted to context management and application core in the design of context-aware services, a design process that supports this structure is not described. In contrast to [12], we here provide a MDA-based design process that supports our structure.

In [13,14], a model-driven service-oriented approach for service development closely related to our approach is presented. Although this approach relies on the same MDA-based design process that we have described in Section 2, it focuses on a different level of this process, namely the model transformation from platform-independent design to platform-specific technologies. Therefore, this approach does not consider the refinement of the service specification based on interactions classification and corresponding interaction patterns for the platform-independent service design.

In the context of MDA, much effort has been done on model transformations from PIMs to PSMs in several application domains. However, traditional MDA development practices [15,16,17] do not consider behavioural aspects of the design and directly start the development by realizing the service design based on the structure that implements the service. Therefore, much attention is given to PSMs and generation of code to implement these PMSs, and less attention is given to the PIM level and the behaviour of the service to be developed. In contrast, we focus on model transformations at the PIM level in order to obtain models that can be implemented with current technologies and that reflect the service structure we have defined. Above all, these models are designed to be consistent and correct with respect to the original behaviour of our service.

7 Conclusions and Future Work

We have defined an approach for the development of context-aware mobile services that starts from the specification of its external behaviour and produces a platform-independent model in terms of components that may execute the service. This approach uses a model transformation based on refinements and interaction patterns. Interaction patterns allow us to identify common pieces of behaviour of context-aware mobile services that can be used to easily assemble new services instead of

developing them from scratch. Moreover, we have illustrated the application of our approach by means of a case study, i.e., the Live Contacts service.

We have presented the steps to realize the model transformation from service specification in A-MUSE DSL to a platform-independent service design in ISDL. We have realized these steps by manually applying refinements, identifying interaction patterns, and creating mappings. We have not discussed any transformation language nor provided any transformation rules, and we have not considered yet the rules to check whether certain combinations of interaction patterns are legal. These are topics for future work. However, we have provided a starting point to define guidelines that can be used to automatically realize our transformation. The automation of the transformation is also subject of further study. Towards this aim, we are investigating tool support for model transformations in the context of the A-MUSE project. We are currently investigating how to specify and execute these transformations using *medini QVT* [18], which is a tool that implements the Query/View/Model (QVT) Relations specification defined by OMG for model-to-model transformations.

We have provided examples of refinements exported from the Grizzle tool, which supports both A-MUSE DSL and ISDL, which are the languages we have used to specify our models. These are simple examples that represent single instances of behaviours. They have been introduced mainly to clarify how refinement interaction patterns can be identified and mapped into components in our approach. Although these refinements seem to grow as the examples become more complex, the A-MUSE DSL and ISDL allow us to modularize single instances of behaviours in order to reduce the complexity and facilitate the understanding of the resulting behaviour diagrams. Scalability issues in case multiple instances of behaviour have to be considered in further investigation.

We have used A-MUSE DSL and ISDL since these are general-purpose languages that allow us to model behavioural aspects in terms of causality relations between interactions without constraining the internal implementation of the services. Particularly, A-MUSE DSL is a specialization of ISDL and both languages are based on the COSMO framework [23], which is a framework used in the A-MUSE project as a conceptual basis for modelling services. By using A-MUSE DSL and ISDL we can support a broad spectrum of abstraction levels, from the highest level of abstraction of our approach, namely the service specification level, to the lowest level of platform-independence, namely the service design level.

References

1. Object Management Group: MDA-Guide, Version 1.0.1, omg/03-06-01 (2003)
2. Freeband A-MUSE Project; http://a-muse.freeband.nl
3. Almeida, J.P.A., Iacob, M.E., Jonkers, H., Quartel, D.: Model-Driven Development of Context-Aware Services. In: 6th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS 2006). Lecture Notes in Computer Science, Vol. 4025. Springer (2006) 213-227.
4. ISDL home; http://isdl.ctit.utwente.nl
5. Quartel, D., Ferreira Pires. L., van Sinderen, M.: On Architectural Support for Behaviour Refinement. In: Journal of Integrated Design and Process Science. Vol. 6, No.1. IOS (2002)

6. Wikipedia page; http://en.wikipedia.org/wiki/Interaction_design_pattern
7. Live Contacts home; http://livecontacts.telin.nl
8. 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 (CSCW2004). November 2004, Chicago, USA.
9. Grizzle home; http://isdl.ctit.utwente.nl/tools/grizzle
10. Almeida, J.P.A.: Model-Driven Design of Distributed Applications. Ph.D. thesis, University of Twente, Enschede, The Netherlands (2006)
11. Shishkov, B.B., van Sinderen, M.: Model-Driven Design of Context-Aware Applications. In: Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS 2007), June 2007, Funchal, Portugal. INSTICC Press (2007), Vol. 3, 105-113.
12. 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), Ayia Napa, Cyprus.
13. van Sinderen, M., Almeida, J.P.A., Ferreira Pires, L., Quartel, D.: Designing Enterprise Applications Using Model-Driven Service-Oriented Architectures. In: Enterprise Service Computing: from Concept to Deployment. Idea Group Publishing (2006), Hershey, 132-155.
14. Almeida, J.P.A., Ferreira Pires, L., van Sinderen, M.J.: Abstract Platform and Transformations for Model-Driven Service-Oriented Development. In: Proceedings of the 2nd International Workshop on Model-Driven Enterprise Information Systems (MDEIS 2006), 23 May 2006, Paphos, Cyprus. INSTICC Press (2006), 49-63.
15. Jones, V., Rensink, A., Ruys, T., Brinksma, E., van Halteren, A.: A Formal MDA Approach for Mobile Health Systems. In: Proceedings of the Second European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations (EWMDA 2004). Computing Laboratory, University of Kent, Canterbury, Kent CT2 7NF, UK, Canterbury, 28-35.
16. Fink, T., Koch, M., Pauls, K.: An MDA Approach to Access Control Specifications Using MOF and UML Profiles. In: The First International Workshop on Views on Designing Complex Architectures (VODCA 2004). Electronic Notes in Theoretical Computer Science (2006), Vol. 142, 161-179.
17. Eissen, S. M., Stein, B.: An MDA Approach to Implement Personal IR Tools. In: 18th International Conference on Database and Expert Systems Applications (DEXA 2007). IEEE Computer Society Press (2007), 259-263.
18. medini QVT: ikv++ technologies home; http://www.ikv.de
19. Dey, A.K., Salber, D., Abowd, G.D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction (2001), 16(2-4), 97-166.
20. Chen, H., Finin, T., Joshi, A.: An Ontology for Context-Aware Pervasive Computing Environments, Knowledge Engineering Review, Special Issue on Ontologies for Distributed Systems, Vol. 18, No. 3. Cambridge University Press (2003) 197-207.
21. Dockhorn Costa, P., Ferreira Pires, L., van Sinderen, M.: Designing a Configurable Services Platform for Mobile Context-Aware Applications. In: International Journal of Pervasive Computing and Communications (JPPC), Vol.1, No. 1. Troubador Publishing (2005)
22. McFadden, T., Henricksen, K., Indulska, J., Mascaro, P.: Applying a Disciplined Approach to the Development of a Context-Aware Communication Application. In: 3th IEEE International Conference on Pervasive Computing and Communications (PerCom). IEEE Computer Society Press (2005) 300-306.
23. Quartel, D.A.C., Steen, M.W.A., Pokraev, S.V., van Sinderen, M.J.: COSMO: a Conceptual Framework for Service Modelling and Refinement. In: Information Systems Frontiers, Vol.9, Springer Science and Business Media (2007), 225-244.