Context Handling in a SOA Infrastructure for Mobile Applications

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. Context-aware mobile applications can dynamically adapt their behaviour to changes in the user context and provide their users with relevant services anywhere and at anytime. In order to develop such applications, a flexible infrastructure is necessary that supports several tasks, such as context information gathering and services provisioning according to this information. This paper presents a SOA-based infrastructure that divides these tasks among several components. These components interoperate by making use of each other's services. In order to allow components interoperability, we propose the use of context models, which represent the relevant concepts of our application domain independently of specific design and technological choices. Moreover, we present a generic component, the context expression evaluator, which has been defined to facilitate the handling of context conditions, and we illustrate how this component has been integrated with other components of our infrastructure by using context models.

1 Introduction

Context-aware mobile applications are capable to satisfy their users' current needs by dynamically adapting their behaviour to their users' context without explicit user intervention. These applications can sense and explore their users' context, reason about that, and, based on context changes, provide relevant services to their users anywhere and at anytime. In order to develop context-aware mobile applications, an infrastructure is necessary that consists of generic components and supports general purpose functions used by such applications. Because of stringent time-to-market requirements, it is not desirable that each individual application captures and processes context information for its own use, since application development in this case would be time consuming and costly. In contrast, with a proper infrastructure, generic functions can be made available and reused in various context-aware mobile applications without developing them from scratch.

This paper presents an infrastructure that has been developed within the A-MUSE project [1]. This infrastructure is based on the Service-Oriented Architecture (SOA) approach, which aims at facilitating distributed systems design through the disciplined use of the service concept. As defined in [2], services can be described, published, discovered and dynamically assembled for developing massively distributed,

interoperable and evolvable systems. Our SOA-based infrastructure supports common functions of context-aware mobile applications by assigning them to dedicated components. Particularly, user components provide the application with user events through a user interface, context sources sense the user context and provide the application with context events, action providers execute actions in response to events, a service trader can be used to register the services offered by context sources and action providers to make them dynamically available according to SOA, and, finally, a coordinator orchestrates the components mentioned above. User components, coordinator, context sources, action providers and service trader support the goals of the application by interoperating with each others.

The main contribution of this work consists of introducing context models to allow interoperability among components of the A-MUSE infrastructure. Context models represent the relevant concepts in the application's universe of discourse and components that realize application parts may use these models as a reference to interoperate with other components. In order to show how it is possible to make additional components interoperable with our infrastructure as long as they comply with our context models, we present a *context expression evaluator*, which is a generic component dedicated to context processing that has been developed in the A-MUSE project to facilitate the handling of context conditions, and we discuss how the context expression evaluator has been integrated in the infrastructure by using context models.

The structure of the paper is the following: Section 2 presents the A-MUSE infrastructure, Section 3 discusses the importance of context models and shows an example to illustrate them, Section 4 presents the context expression evaluator component and discusses how it has been integrated in the infrastructure by using our context model, Section 5 discusses some related work, and Section 6 presents our conclusions and identifies topics for future work.

2 Infrastructure

The A-MUSE infrastructure supports general purpose functions that are commonly used by context-aware mobile applications. For example, all context-aware mobile applications are capable to retrieve context information from the user context and, based on this information, provide relevant services to their user. Therefore, our infrastructure is general enough to be used by various context-aware mobile applications. However, it may be configured based on the specific application to be developed by simply implementing functions that are not worth generalizing, since they are too specific and, therefore, not used by other applications. These specific functions should be offered by application-specific components.

Fig. 1 shows the A-MUSE infrastructure. Although it has been especially developed in the A-MUSE project for a case study called Live Contacts [3], the same infrastructure can be used by other context-aware mobile applications. Live Contacts consists of a context-aware mobile application 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 the functionality that Live Contacts offers to its user can be found in [4].

The core of the infrastructure consists of a service coordinator, which receives events and as a consequence triggers actions. 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 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.

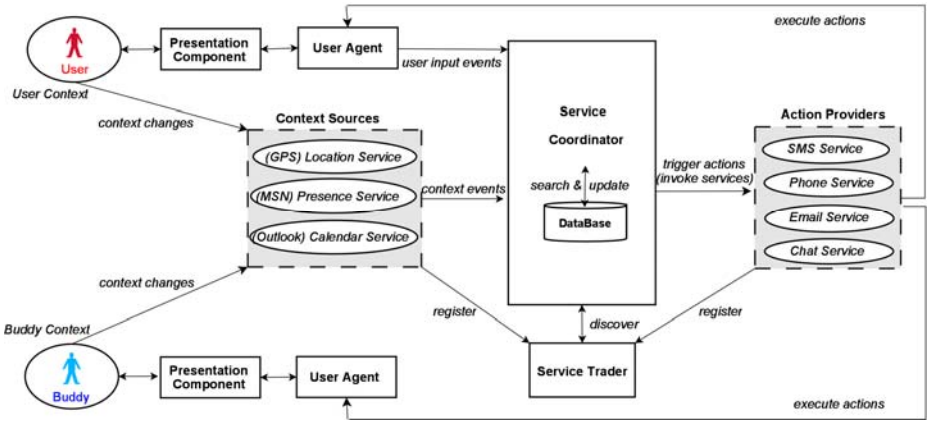

Fig. 1. A-MUSE infrastructure for context-aware mobile applications

Fig.1 depicts a user and his/her buddy (another user of the application) with their context. They interact with the application through a presentation component that provides the user interface to introduce inputs and receive outputs to/from the application. Since context-aware mobile applications should be able to provide relevant services to their users anywhere and at anytime, the presentation component is integrated in the user device, which may be either a desktop PC, in case of users with fixed location, or a Smartphone or Pocket PC phone in case of users on the move. There is one presentation component for each user.

The user agent acts on behalf of the user. Particularly, the user agent interacts with the presentation component to obtain user inputs and present user outputs, and provides the service coordinator with user input events. The user agent is located in the user device. There is one user agent for each user.

The service coordinator orchestrates all the components of the application, including a database that contains status information about users (e.g., name, password, preferred contacts means, list of buddies), and login information. In principle we assume that there is one service coordinator and one database. The service coordinator can manage multiple user instances in order to allow several users

to use the same application, and multiple application instances in order to allow several context-aware mobile applications to share the A-MUSE infrastructure. 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 triggered by the service coordinator. Fig. 1 shows an 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 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 interface that is 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. 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 [5] and the UDDI registry [6].

The interactions among components of our infrastructure are based on the service-oriented architecture (SOA) approach, which consider components only from the point of view of the service that they provide or use without any mentioning of the internal details of how the service itself is implemented. According to SOA, components make use of each other's services to cooperate in order to support the goals of the application. Therefore, the coordinator uses the service offered by context sources, which provide the coordinator with context events. However, the coordinator is not aware of the details on how context sources obtain context information from the user environment and how they process this information in order to generate context events.

3 Context Model

In this work we start from the definition of context given in [7], which is the following:

"Context is a collection of interrelated conditions in which something exists or occurs".

This definition implies that we always consider context as a set of conditions associated with a subject, which is something that "exists or occurs" in our definition. In our models, a subject of context is called an *entity*. Although the concepts of entity

and context are strongly tied, these concepts are fundamentally different. Actually, context is what can be said about an entity in its environment, which implies that context does not exist by itself [8]. The context of an entity is characterized by several and possibly interrelated conditions. Considering the context of a person, examples of these conditions are the person's geographical location, environmental conditions of the person's physical environment, such as temperature, humidity, light, etc., or the person's vital signs, like heart beat and blood pressure. Together, these context conditions form the person's context.

In order to develop context-aware mobile applications we need to identify the context conditions that are relevant for the application or family of applications that we want to develop. In order to achieve this we define a context model, which represents the relevant context conditions of entities in the application's universe of discourse [8]. Particularly, we define a context model as a conceptual model of context. Conceptual models are, in the sense of [9,10], abstract representations of a "given subject domain, independent of specific design and technological choices". Therefore, when we define a conceptual model of context, we abstract from any design and technological detail, such as the way context is sensed, provided, learned, produced, and/or used.

Context models provide us with a conceptual foundation for the development process of context-aware mobile applications. Particularly, context models allow us to provide interoperability among components of the A-MUSE infrastructure depicted in Fig. 1. In this infrastructure, each component realizes specific parts of the application logic and, in order to support the goals of the application, it has to interoperate with other components. Since our context model should represent all the concepts (entities and context) used in the application, components that realize application parts use concepts that should be represented in this model, and, therefore, are known and understandable for other components. In other words, a context model is fundamental since it provides the common vocabulary to "make our components understand the same language". We propose context models that are based on [8], which provides foundational ontologies to support conceptual modelling and situation reasoning. Fig. 2 shows an example of context model.

Fig. 2 represents the Entity and Context classes, which are foundational concepts in our context models. Any entity may be related to several different contexts and a specific context may be referred to one or more entities. In Fig.2, SpatialEntity represents tangible objects, such as a person or a device and IntangibleEntity represents intangible objects, such as an application or a network. IntrinsicContext represents a context type that belongs to the essential nature of a single entity and does not depend on the relationship with other entities. An example of intrinsic context is the location of a person or a device. In contrast, RelationalContext represents a context type that depends on the relation between distinct entities, such as the BuddyList in Fig. 2 that belongs to a User entity and contains several Buddy entities. The User and Buddy classes, which represent person types or roles in the Live Contacts application, are overlapping since we assume that a user has several buddies and at the same time this user is a buddy of other users. Therefore, a user is also a buddy and vice-versa.

Moreover, Fig. 2 depicts the ContextSituation class, which consists of contexts and entities. We consider context situations since they enable the representation of particular state-of-affairs of the applications' universe of discourse. Example of

context situations are Proximity and ChangeMSNStatus, which describe, respectively, when a certain buddy of a user is nearby this user, and when a certain buddy changes his/her status on Messenger. Fig. 3 depicts the specification of these context situations based on the context model of Fig. 2. For the ChangeMSNStatus context situation, we assume that a user, whose status is online, is interested to know when a buddy changes his/her status from busy to online.

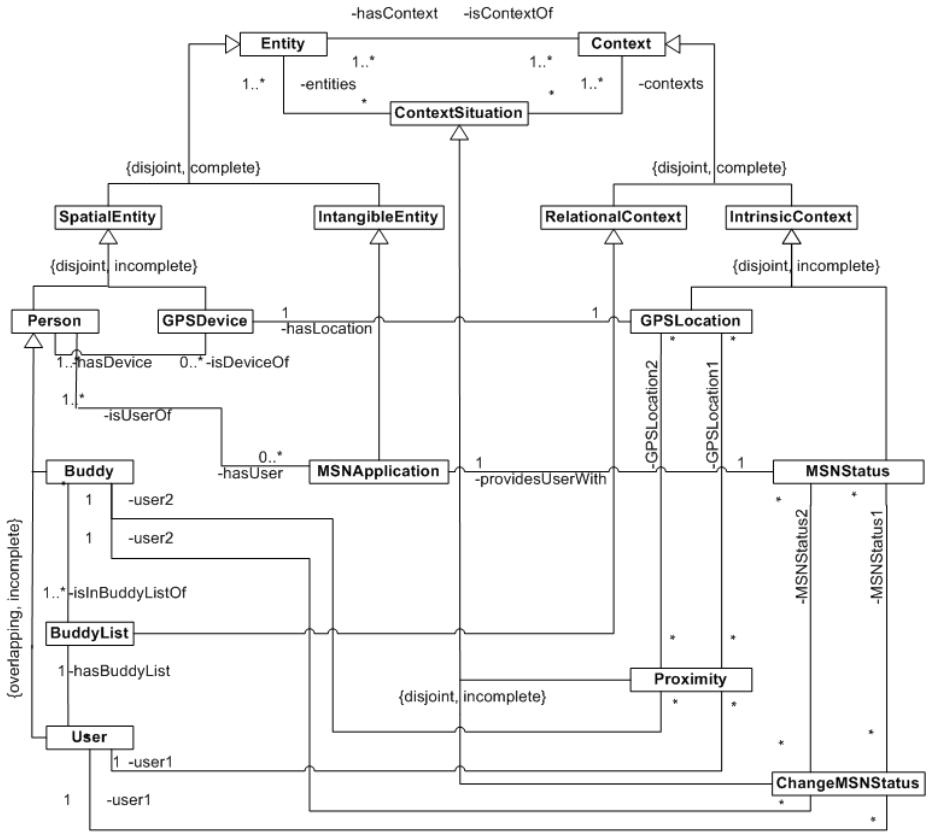

Fig. 2. Context model example for the Live Contacts application

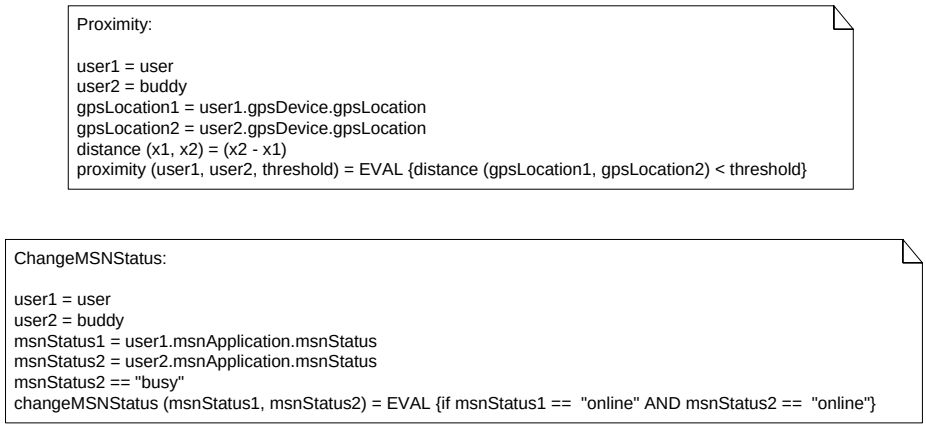

Fig. 3. Specification of the context situations Proximity and ChangeMSNStatus

4 Case Study: Integration of a Context Expression Evaluator

The context expression evaluator is a generic component developed in the A-MUSE project that can be introduced in our infrastructure and facilitate the development process of our applications. Actually, the context expression evaluator is dedicated to context processing, namely, to gather context information from context sources and reason about this information in order to provide notification of relevant events in the user context. Since in the infrastructure depicted in Fig. 1 these tasks are realized by the coordinator, which is also responsible of orchestrating all the internal interactions within the infrastructure, the introduction of the context expression evaluator can relieve the coordinator from the responsibility of managing context sources. Moreover, the context expression evaluator may provide a mean to reduce the network load that is caused by the SOA-based mechanism of registering and discovering services in the service trader of our infrastructure. The use of the context expression evaluator component is an architectural choice to facilitate context processing in our applications. However, this component is not an essential part of our infrastructure. Fig. 4 depicts the A-MUSE infrastructure with the addition of the context expression evaluator.

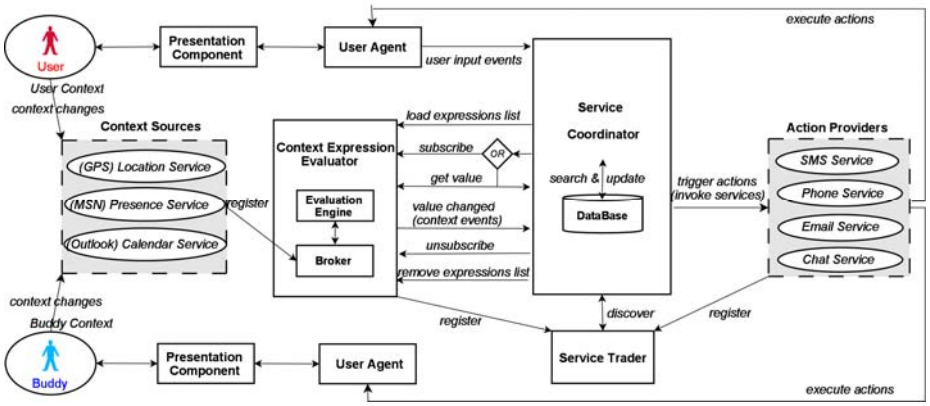

Fig. 4. A-MUSE infrastructure with the context expression evaluator component

Fig. 4 shows the context expression evaluator, which consists of an evaluation engine and a broker. Context sources, such as GPS devices that provide users location, register themselves in the broker in order to allow the evaluator component to discover and invoke their services. The evaluation engine interacts with the broker and evaluates context expressions provided by registered context sources. We are not interested in the internal construction details of the context expression evaluator, since, according to the SOA principles, it should be possible to introduce this additional component in our infrastructure by only providing a description of the service that this component offers. As depicted in Fig. 4, the description of the context expression evaluation service may be registered in the service trader. In this way, the coordinator, as well as any other component, can discover the context expression evaluation service and invoke the proper operations in order to get notifications of context events. Fig. 4 shows these operations.

In the infrastructure we have presented in Fig. 1, context sources register themselves in the service trader in order to be dynamically available to the application components and, particularly, to the coordinator. In this way, the coordinator can subscribe to context sources of interest and get notifications of context events. In contrast, in the configuration depicted in Fig. 4, context sources register their services in the broker of the evaluator in order to be available to the evaluation engine. Therefore, individual context sources are available only to the evaluator and not to other application components. Since we consider it as a context source, the evaluator is the only context source available to the application components and, particularly, to the coordinator. In this way, if the context sources are not widely spread and are placed in the proximity of the evaluator, it is in principle possible to reduce the network load on the coordinator caused by the mechanism of registering and discovering individual context sources in the service trader of the infrastructure. However, a well-founded evaluation on the reduction of the network load can be made only after implementing a prototype of our infrastructure.

The context expression evaluator works as follows. First, the coordinator has to load the evaluator with a list of context expressions that are relevant to the application

(*loadExpressionList* operation). These expressions may concern either context events, such as Proximity or ChangeMSNStatus described in Section 3, or specific values of context information, such as the GPSLocation or MSNStatus of certain users. After loading an expressions list, the coordinator can subscribe to specific expressions of this list (*subscribe* operation) in order to get asynchronous notifications of context events from the evaluator when changes occur in these expressions (*valueChanged* operation). Alternatively, the coordinator can request the current value of some context information and get a synchronous response from the evaluator (*getValue* operation). Finally, the coordinator can unsubscribe to context expressions (*unsubscribe* operation) and remove expression lists previously loaded in the evaluator (*removeExpressionList* operation).

Concerning the interactions at the context sources side, context sources register themselves in the broker as soon as they are available. For example, the GPS device of a user registers its service to the evaluator whenever the user switches on the device. Once the services are registered in the broker, the evaluation engine supports both a subscription-based context information retrieval, analogously to the *subscribe* operation of the coordinator, and a query-based context information retrieval, analogously to the *getValue* operation of the coordinator. The evaluation engine optimizes the communication with context sources by evaluating only essential information. For example, when the engine is monitoring a boolean AND context expression consisting of two sub-expressions and one of these sub-expression returns false, the engine does not retrieve the value of the other expression until the first one turns true. Moreover, when monitoring context expressions used in several user instances, the engine retrieves only once the value of that expression from the corresponding context source.

Considering the Proximity and ChangeMSNStatus context situations described in Section 3 , the coordinator first loads in the evaluator the following expressions list:

```
user1_gpsLocation = user1.gpsDevice.gpsLocation
user2_gpsLocation = user2.gpsDevice.gpsLocation
user1_user_2_proximity = proximity(user1,user2,threshold)
user1_msnStatus = user1.msnApplication.msnStatus
user2_msnStatus = user2.msnApplication.msnStatus
user2_changeMsnStatus =
changeMSNStatus(user1_msnStatus,user2_msnStatus)
```

The list above uses the language understood by the evaluator. In this language:

- `user1_gpsLocation` is an example of expression name, which is used by the evaluator to identify a context expression unambiguously;
- `user1.gpsDevice.gpsLocation` is an example of context expression concerning a specific value of context information. This expression refers to types of entity (`user1`), context source (`gpsDevice`) and context (`gpsLocation`) that are represented in our context model;
- `proximity(user1,user2,threshold)` is an example of context expression concerning context events. This expression refers to the specification presented in Fig. 3, in which `proximity(user1,user2,threshold)` has been defined as a function that evaluates whether the distance between the GPS locations of a user and one of his/her buddies is smaller than a certain threshold.

After loading the expressions list, the coordinator can subscribe to context events by passing the name of the corresponding expression, e.g., `user1_user_2_proximity`, as argument of the *subscribe* operation. Analogously, the coordinator can ask for a specific value of context information by passing the name of the corresponding expression, e.g., `user1_gpsLocation`, as argument of the *getValue* operation.

5 Related Work

The benefits of using Service-Oriented Architecture to support the development of context-aware applications have been extensively discussed in the literature. In [15], 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. Much effort has been done to develop SOA-based middleware solutions for context-aware applications [11,12,13,14] in order to provide these applications with flexible infrastructures that can be extended with little effort. Some of these solutions propose context models to provide semantic interoperability among infrastructure parts. These context models have been realized by using OWL [17], which is a web ontology markup language proposed by W3C's Web Ontology Working Group. Although these context models are ontology-based, they do not consider ontologically well-founded theories to support their modeling choices. For example, the concepts of context and entity are frequently used interchangeably, which does not reflect the fundamental characteristics of these concepts. Moreover, although related work address situation modelling, often a required level of expressiveness, especially with respect to temporal aspects, is not provided. In contrast, we have proposed context models that are based on [8], which provides support to conceptual modelling with foundational ontologies and a well-founded situation theory for situation reasoning, also with respect to the definition of temporal relationships among situations.

6 Conclusions and Future Work

We have presented a SOA-based infrastructure for the development of context-aware mobile applications that has been defined in the scope of the A-MUSE project. In [16], we have defined this infrastructure as part of a design process based on the Model-Driven Architecture (MDA) approach [18]. This design process first specifies the behaviour of the application to be developed and gradually refines this behaviour in sequences of interactions, called interaction patterns, among the infrastructure components. Therefore, the use of our infrastructure for context-aware mobile applications is tightly connected to the use of models that describe behavioural aspects of such applications. These behavioural models are based on context models,

which provide a conceptual foundation to handle context conditions used by our applications. We have discussed the importance of context models, which allow us to provide interoperability among components of the A-MUSE infrastructure, and we have presented examples of these models.

We have discussed how it is possible to integrate in our infrastructure additional components that are dedicated to specific parts of the application logic, such as, e.g., context processing, as long as they comply with our context models. Particularly, we have introduced a component, the context expression evaluator, which has been developed in the A-MUSE project to facilitate the handling of context conditions. The context expression evaluator has been defined to relieve the coordinator component from managing context sources in order to get notifications of context events, so that it can concentrate on the orchestration of the interactions between other components in the infrastructure. Moreover, the evaluator may help in reducing the network load caused by SOA interactions in our infrastructure. Particularly, we have shown how to make the evaluator and coordinator components interoperable by using a context model that represents relevant entities and context in the universe of discourse of context-aware mobile applications.

Since this work is part of an ongoing project and the A-MUSE infrastructure has not been implemented yet, we have not provided technical details, such as which protocols and technologies are used to realize the service-oriented architecture, which limits we have experienced in these technologies, and how we have integrated in the infrastructure different services like the GPS, MSN and Outlook services. These issues have to be considered in further investigation. The natural evolution of this work consists of implementing a prototype of our infrastructure that uses the context expression evaluator component for handling context conditions. In order to achieve this, current work is being done within the A-MUSE project in order to realize a running context-aware mobile application, based on the Live Contacts case study presented in Section 2, which concretely demonstrates the interoperation of the a-MUSE infrastructure components at runtime.

Acknowledgements

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. Special thanks to Mathieu Hutschemaekers and Hugo Zwaal (Ericsson Telecommunicatie B.V.) for their contribution to this work on the context expression evaluator component.

References

1. Freeband A-MUSE Project; http://a-muse.freeband.nl
2. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing: A Research Roadmap. In: Procedings of Dagstuhl Seminar on Service Oriented Computing

(2005). Internationales Begegnungs und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2007)
3. Live Contacts home; http://livecontacts.telin.nl
4. 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
5. Object Management Group: Trading Object Service Specification, Version 1.0, formal/00-06-27 (2000)
6. OASIS: OASIS-Committes- OASIS UDDI Specifications TC; http://oasis-open.org/commitees/uddi-spec/doc/tcspecs.htm
7. Merriam-Webster Online Dictionary page; http://www.m-w.com/dictionary/context
8. Dockhorn Costa, P.: Architectural Support for Context-Aware Applications: from Context Models to Services Platforms. Ph.D. thesis, University of Twente, Enschede, The Netherlands (2007)
9. Guizzardi, G.: Ontological Foundations for Structural Conceptual Models. Ph.D. thesis, University of Twente, Enschede, The Netherlands (2005)
10. Mylopoulos, J.: Conceptual Modeling and Telos. In: Conceptual Modeling, Databases, and CASE, Whiley 49-68
11. Kim, E., Choi, J.: A Context-Awareness Middleware Based on Service-Oriented Architecture. In: Procedings of the 4th International Conference on Ubiquitous Intelligence and Computing (UIC 2007). Lecture Notes in Computer Science, Vol. 4611. Springer (2007) 953-962
12. 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). IEEE Computer Society Press (2007) 946-951
13. 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). IEEE Computer Society Press (2005) 513-518
14. Gu, T., Pung, H.K., Zhang, D.Q.: A Service-Oriented Middleware for Building Context-Aware Services. In: Journal of Network and Computer Applications (JNCA), Vol.28, No.1. Academic Press Ltd, London, UK (2005) 1-18
15. Yoon, H.: A Convergence of Context-Awareness and Service-Orientation in Ubiquitous Computing. In: International Journal of Computer Science and Network Security (IJCSNS), Vol.7, No.3 (2007) 253-257
16. Daniele, L., Ferreira Pires. L., van Sinderen, M.: Interaction Patterns for Refining Behaviour Specifications of Context-Aware Mobile Services. In: Proceedings of the 4th International Workshop on Model-Driven Enterprise Information Systems (MDEIS 2008), 12 July 2007, Barcelone, Spain. INSTICC Press, to appear
17. OWL home; http://www.w3.org/TR/owl-features
18. Object Management Group: MDA-Guide, Version 1.0.1, omg/03-06-01 (2003)