Erschienen in: Ubiquitous mobile information and collaboration systems: Second CAiSE Workshop, UMICS 2004, Riga, Latvia, June 7 - 8, 2004; revised selected papers / Luciano Baresi ... (ed.). - Berlin [u.a.] : Springer, 2004. - S. 43-56. - (Lecture notes in computer science ; 3272). - ISBN 978-3-540-24100-3
http://dx.doi.org/10.1007/978-3-540-30188-2_4

Modelling Context for Information Environments

Rudi Belotti, Corsin Decurtins, Michael Grossniklaus,
Moira C. Norrie, and Alexios Palinginis

Institute for Information Systems,
ETH Zurich,
8092 Zurich, Switzerland
`rudi.belotti@switzerland.org`
`{decurtins, grossniklaus, norrie, palingins}@inf.ethz.ch`

Abstract. Context-awareness has become an important consideration in the development of mobile and ubiquitous systems. While efforts have been made to develop general context models and application frameworks, it remains the case that often the notion of context supported is very restrictive and/or the representation of context is based on simple key-value pairs. As a result, most existing systems lack well-defined semantics and typing for context that would facilitate the general implementation, maintenance and adaption of context-aware systems. In this paper, we present a general context model that consolidates the models underlying many other approaches, while moving to a higher-level of abstraction in terms of semantic context description.

1 Introduction

As humans, every time we act and speak, we do it in a given context. Adapting to context in our daily lives is very easy for us and most of the time it is a completely subconscious action. We implicitly change our behaviour depending on the situation and in relation to the environment. Context is the driving force behind our ability to discriminate what is important and what is not. It allows us to enrich our knowledge about a certain situation by drawing on related memories and experiences. Hence, only through context are we able to manage the tremendous amount of information that surrounds us at any given point in time. For instance, the implicit use of context during a dialogue between humans has been shown to increase the conversational bandwidth [1].

The nature of the information that surrounds us is not unlike the data that can be found in ubiquitous and mobile applications. As in real life, vast amounts of information are readily available from an almost unlimited number of sources. To make such information environments work, the need to distinguish relevant from irrelevant information arises, as well as the necessity to augment the information by associating related knowledge. If such applications are ever to be successful, it will entirely depend on their ability to deliver the right information to the right user at the right time. We believe that context is a powerful instrument to achieve the dissemination of relevant information that those applications demand.

Many of us dream of better applications that have at least an inkling of what we are doing or where we are working and adapt themselves accordingly. This adaptation can come in various forms. For instance, a very simple adaptation would be having the user interface adapt according to the situation by presenting the toolbars which offer the actions that are needed to perform the next task we have in mind. A more complex example would comprise the delivery of useful and relevant information via mobile devices such as mobile phones or personal digital assistants (PDAs) to support expedient collaborative work. A PDA, for instance, could be used to display background information on the knowledge and know-how of a co-worker whenever this person enters the room. Mobile phones on the other hand could simply adapt their ringing profile automatically depending on the situation. Context in this example would also allow calls that are irrelevant to the current setting, such as an important meeting, to be filtered out.

Recently, a lot of research has been dedicated to context and context-aware applications. Nowadays, the first results on context-aware computing are already appearing in various application domains. Work on context has mainly been concentrated in the areas of philosophical implications of context [11], low-level implementation of context-aware systems [30, 4] and the development of more general frameworks [27, 28] that provide a basis for the engineering of such applications. We feel, however, that all of the models used in these approaches lack precise and expressive semantics for context as well as typing. In this paper, we present our comprehensive model for context that aims at consolidating existing approaches. As it is based on a well-defined object model and therefore features well-defined semantics and typing, it propels the reusability and interchangeability of context in a given application.

In Sect. 2 existing work on context and context-aware applications is presented and related to our work. Section 3 introduces our own model for context and gives an overview of the various parts of the model. Sections 4 to 6 then examine certain aspects of the model in greater detail. Final remarks and conclusions are given in Sect. 7.

2 Related Work

Context has become a well researched topic over the past few years with contributions from a wide variety of domains, including philosophy [11], linguistics and social sciences as well as many different fields of computer science. In the following we present related work from some of these areas and state how these results relate to our own approach. The aim is not to give a complete overview of all related work, but rather to provide an overview of the various approaches that people have adopted to tackle the problem of context. A more complete survey on related work, especially in the field of mobile computing, is available in [9].

The problem of context has a long tradition in different areas of cognitive science. Several formalisations [17, 10] of context have been proposed and, in the field of Computational Linguistics [18], context is a central notion for understanding and working with text or speech. Cognitive system engineers and specifically people in the area of Human Computer Interactions (HCI) have recognised that cognition does not exist without a context and have started applying the theory to practical applications [4].

The first implementations of context-aware systems originate in the area of application development for ubiquitous and pervasive computing. The solutions tend to be application-driven, i.e. context is not necessarily regarded as the main research topic, but rather just used to implement a certain functionality for a given application. In [2], for instance, although a context layer is described in the architecture, its design is heavily affected by the fact that the system is built with the focus on multi-channel acquisition and delivery and therefore not considered as a general purpose context model. In other approaches context providers — physical sensors and software components — are added to the applications to allow them to sense their environment and gather information about users, location and possibly other properties of the physical environment. Context is an integral part of these systems. The integration of sensors and the reactions and adaptations of the system to the context are customised and tailor-made for each application. Most of the systems do not have an explicit model of context. Context is just additional data that the application can use to provide the desired functionality. An interesting example of such an application is described in [30] where active badges are used to sense the location of people and forward calls for a given person to a telephone in the same room.

The notion of context has also been integrated into various application frameworks, e.g. for web engineering. The output generated by a web application usually depends on various explicit parameters. These parameters include, for instance, the exact URL that was requested or the input of a form. In addition to these explicit parameters, the output can also depend on *implicit* parameters, i.e. parameters which are not specified explicitly by the user, but rather form the context of the current session. These implicit parameters might include the preferred language of the user, the browser that is used to access the web application (e.g. an HTML browser or a WAP phone) or the current time. Also, the history of the current session, i.e. the documents that the user has accessed before the current request, can be part of this context. A discussion on implicit and explicit application parameters can be found in [15].

The web modelling language WebML and the corresponding CASE tool WebRatio have also incorporated a notion of context [3]. This context is used to provide users with a personalised and customised view of the web information system. Pages of the application can be defined to be context-aware, which means that the context elements are available to the page in the form of data units. The page can then be customised according to these context data units.

We have developed our own context-aware web publication frameworks OMS Web Elements (OMSwe) [23, 22] and the eXtensible Content Management (XCM) [7, 8] as an extension of an object-oriented database system. The database contains, not only objects of the application domain, but also metadata that controls the assembly of documents along with templates that are used to render the pages into a specific output format. For each of these objects, multiple variations can coexist. Each variation is annotated with special attributes known as *characteristics*. For example, for an object of the application domain, this might be the language of the content, whereas, for a template object, it might be the type of the output format. In the process of responding to a specific user request, the correct variants of the objects are selected according to the context of the request.

In the realm of ubiquitous and pervasive computing, many researchers have worked on application-specific context models [29]. The focus of such applications is the physical context instances such as location, temperature etc. and the relationship between them. The application specification handles the context acquisition and manages both the context information and the interoperability with the application in a hard-wired manner.

While the approaches that we have described so far are quite feasible for standalone context-aware applications, they are certainly not practical for the integrated suites of context-aware applications that have recently been developed in, for example, the AwareHome [13] and Gaia [26] projects. To achieve a better separation of the sensors for the context aggregation and the context-aware applications, various people started to develop *context frameworks*. These frameworks allow the integration of various sensors through corresponding drivers and present a sensor-independent view of the context information to the applications. Instead of having sensors and context as an integral part of the application, the systems use a context component that provides context information for multiple applications. In some frameworks, the applications are also able to modify and insert additional context information and thus are able to act as context deliverers for other applications. A well-known representative of this approach is the Context Toolkit [27], which was used for a variety of research projects, including the previously mentioned AwareHome. Another example is the context component of the Gaia project.

Proposed frameworks define further system architectures [5, 28, 31] that offer general mechanisms and services to the underlining applications. In [12], this kind of service infrastructure is even shifted towards network-accessible middleware infrastructures.

Whereas context frameworks provide a good solution for the problems of aggregation and decoupled provision of context information, they do not really solve the problem of the lack of semantics. In most frameworks, context values are nothing but key-value pairs or higher-order tuples, with little explicit semantics. Some systems provide simple taxonomies for the values, but these conventions are neither checked nor enforced by the system. The interpretation of the context information is left completely to the application. Thus a real separation of context producers and context consumers is not achieved by context frameworks alone. With our background in information systems, we believe that the lack of semantics and expressiveness of context in the previously described approaches is actually the biggest drawback and makes them inadequate for use in a general context model and framework for information environments.

Various people have come up with definitions for context. A precise definition of context is very important for users of context frameworks and developers of context-aware applications in general. Schilit [28], for example, identified the *computing environment* (processors, devices, displays etc.), the *user environment* (location, social situation, nearby people etc.) and the *physical environment* (e.g. lighting and noise level) as the key components of context. Dey and Abowd [1] describe context as:

> [...] any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

These definitions have also been incorporated into some context frameworks. The Context Toolkit, for instance, is based on the definition of Dey and Abowd. However, the integration is very weak and the system provides little means of checking and enforcing the definitions. In most implementations, the efforts of defining context are only used for naming conventions. For this reason, other researchers have formalised definitions of context in the form of ontologies. With corresponding changes to the context frameworks, these definitions can be exploited by the systems and thus more powerful semantics can be built right into the frameworks and applications.

The previously mentioned definitions of context are mostly influenced by the ubiquitous and pervasive computing community. Another community which is very interested in context is the HCI community. Researchers from this field have brought aspects of ergonomics, psychology, sociology and even philosophy into the discussion. Dourish [6], for example, says that context is not information per se, but rather a property of existing information. Every data object can be "contextually relevant" to another object for a particular activity of a user. In other words, there is no separation between context objects and normal application objects.

The notion of a subject for context values, i.e. an entity that the context value is attributed to, has been implemented for example by the context framework of the Gaia project. They use quadruples for the representation of context: *Context(ContextType, Subject, Relater, Object)*. The context *Object*s have a *ContextType* and a *Subject* that they attribute. The nature of this attribution is described by the *Relater* property.

As for another requirement of Dourish, namely the integration of context and data model, relatively little research has been done in this area. Of course, it is always possible to use object identifiers or primary keys for context keys, values or subjects to work around this problem. However, we believe that a real integration of context models and data models is necessary to bring context-aware computing to the next level. The information modelling community certainly has great expertise in such approaches and this is actually where we see our main contribution. In the following sections, we will describe our approach for the modelling of context and explain some of the special features in more detail.

3 A Conceptual Data Model for Context

The purpose of this work is neither to develop just another context-aware application nor to create a framework and an infrastructure for building context-aware applications. Instead, by consolidating requirements and experiences from earlier context-aware applications and frameworks, including our own, we seek to develop a conceptual data model around the notion of context.

Based on the approaches presented in the previous section, we agree that frameworks, toolkits and service infrastructure in general, increase code reusability in a system. However, due to their complexity and the service-oriented description, many existing frameworks are difficult to comprehend due to the lack of a clearly defined conceptual information model. The specification of a framework reflects the infrastructure underneath which, inevitably, is influenced by the system on which it is implemented. Service-oriented specification is not sufficient in heterogeneous environments, where information exchange becomes a crucial issue due to the lack of a comprehensive general information model. The solution to such problems is the clear definition of a specific context model and infrastructure metamodel. Although this could limit the flexibility of seeing everything as an abstract service, it enables us to better comprehend and control such environments.

In the subsequent sections, we describe an abstract conceptual model to represent context. Based on this model, frameworks and infrastructures can be built by ensuring comprehension of the context domain and interoperability between the different solutions through the common model.

The semantics used to develop our model are those of the Object Model (OM) [20], an object-oriented data model influenced by ER and conceptual modelling. It is based on the simple and fundamental notions of objects participating in collections that represent real world concepts. Objects can be associated to each other using bidirectional associations. Concepts are represented by shadowed rectangles and associations by shadowed ovals, respectively. The organisation of collections and associations into classification structures together with basic constraints such as membership containment and cardinality constraints over associations increase the expressiveness of the model. Each data model is presented in a two-layer representation, one for the abstract high-level concepts, their roles and relationships, and another for the object type definitions and structures used by an underlying implementation. The simple high-level conceptual abstractions used make the model appropriate, not only for application modelling, but also as a metamodelling tool. In the past, we have used such metamodelling approaches to describe the OM model itself [19] and for the design of a modular database environment [21].

Additionally, OMS Pro, an object-oriented database management system [24] based on the model, enables to test the approach in a rapid prototyping environment. The rich functionality offered by the system ranges from typical database services such as persistence, user transactions, declarative query and definition languages, to system extensions such as XML support, web and GUI interfaces. Such a platform allows us to seamlessly integrate the modelling efforts with a preliminary proof-of-concept infrastructure and prototype application.

By consolidating the field of context-aware applications and frameworks, we highlight three basic concepts illustrated by the modules model of Fig. 1:

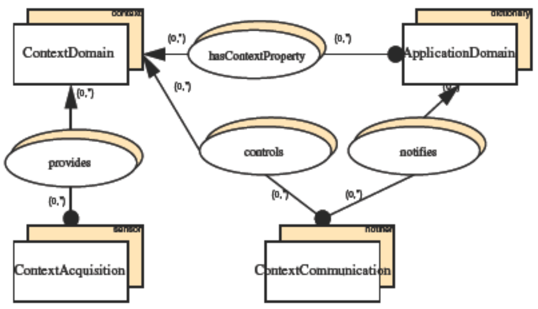

Fig. 1. Modules of the Metamodel

- The abstract notion of *context* as information that can be used to characterise the situation of an application *entity* [1]. Depending on the level of detail when considering the situation, context information can be anything from a single temperature measurement to an arbitrary complex structure, including entities of the application domain itself, representing multiple aspects of the entity's environment.
- Context information is gathered by *providers*. They supply a wide range of context information, from real world environment properties such as temperature sensing to complicated social behaviour aspects of the entity in the current situation. This is the reason for using a general notion of providers rather than the one of sensors typically used in supplying characteristics of a physical environment.
- *Notifiers* bridge the world between the environment properties and the entities interested in them. Crucial to the applicability of a context-aware application is the interaction between the context model and the application model. The mechanism allows both the automatic notification on context change or the explicit context acquisition upon request.

Instances of these modules and the interaction between them should be a subject of well defined definitions. Thus all model components have to be based on a *type* system.

In the following sections, core domain concepts of our model, such as the acquisition of context, types and operations on context, event triggering and the binding of context to an application are presented in more detail.

4 Context, Types and Application Entities

The core concept of the model is the *context* element, which represents the context abstraction. In most of the related discussions and definitions, context information characterises an *application entity* which is illustrated in the model of Fig. 2 by the association `hasContextProperty` in the lower right corner. The cardinality constraint of `(1,1)` on the source of the association reveals that any context element is associated with exactly one application entity. Reusability and semantic relevance of context elements of the same class, such as that of the temperature of a building and of the temperature of a human is introduced in our model through well defined context types. Thus, based on a context type, we instantiate for multiple application entities distinct context. We should remark here that, at this level of abstraction, it does not make sense to distinguish sensed and calculated context as in the work of Dey. As an example, the temperature reading of a sensor or the social situation are both context elements that characterise the entity "room_A45" of a building.

From the semantic point of view, this simple context/entity model is enough to express definitions and formalisms found in [1, 17, 25]. Nevertheless, the difference between the approaches is the kind of information that composes the context and the flexibility to use references to application entities. As discussed in [9], different approaches use diverse data structures to express and exchange context information. The lack of a general, yet flexible, model makes the exchange of context information a tedious task.

We propose a solution to the problem by introducing a type system along with our context model concepts. The type system provides types for three major domains:

- Types defining application domain entities, *ApplTypes*. These are used to control the interoperability with the application through the associated entities. The detailed attribute definitions of the application types are out of the scope of a context model and therefore it suffices here to consider these type objects only as unique references to the actual application type. Imagine, for instance, types such as `user` and `location` of an application from the ubiquitous computing domain. Although the detailed type definition is not of interest, inheritance relationships involving them may be used to reveal entity compatibility.
- *Base Types* that define basic value types such as `integer`, `real` etc. The model offers further bulk types to express lists of values of a given type. Thus, we could, for example, specify a context element as a `set` of integer values or a set of application references to `user` entities. `BaseTypes` can also build hierarchies of restrictions. This is a very useful construct that allows syntactical and some semantic control over base values. A similar design is found in the simple data type declarations of XML Schema. In the scope of context, imagine the base types `celsius` and `fahrenheit` modelled as a restriction of type `real`. Although both can be regarded as real numbers, we can clearly define operators between them, namely, a Celsius to Fahrenheit transformer and vice versa.
- Types defining *ContextTypes*. Each context is well defined by its context type declaration which is composed of a set of attribute definitions. Attributes have a name and can be of any type, `BaseTypes`, `BulkTypes`, `ApplTypes` or further `ContextTypes`. This is an elegant way to model aggregated context as discussed in [1] without the need of introducing separate context concepts such as *Widgets* and *Servers*

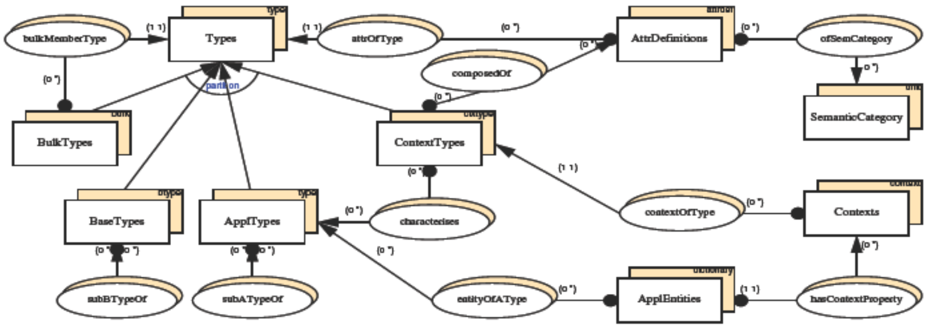

Fig. 2. Context Type Model

Attribute definitions can be classified through the `ofSemanticCategory` association to provide semantic categorisation of attributes. Imagine, for instance, a context-aware agenda application with a composite context element with two attributes, `capture_time` and `appointment_time`. Although both are of type `time`, the `capture_time` attribute designates the time that the context was captured and the `appointment_time` designates the actual time of the appointment. The context modeler can create a semantic category `capture_time` that will be used to classify context attributes for a particular purpose.

Attribute values specifying the quality or confidence of the context values could also be modelled by using the concept of semantic categories. As an example, imagine the confidence value of a face recognition sensor. Due to changes in the lighting conditions, the sensor might work better in some situations than in others. A framework based on our model can classify an otherwise convenient attribute definition, as a quality value. In other approaches, confidence might qualify a complete sensor such as a camera. Thus declaring the confidence of the measurement on the sensor level. By modelling the quality as a class of an attribute, we can define quality values for each measurement and not just a single unique value for all measurements of a sensor. Further, through the context composition, we can define the overall confidence of the sensor by composing the complex context information. Imagine, for instance, face recognition context based on a camera with a given quality value based on the camera's resolution. By using further sensors, such as lighting sensors or badge detection, we could create a composite face recognition context with a quality value that takes into account all gathered information.

The proposed type system is general enough to describe both simple value properties and complex composite context elements. By incorporating references to application types, context-aware applications can control the interaction with context information. Furthermore, context information not only consists of simple basic values, but can also use any application specific entity. In Fig. 2, this is apparent from the `attrOfType` association bound to `Types` with `ApplTypes` being a specialisation of it.

Nevertheless a type system can only ensure data type integrity and consistency, disregarding semantic constraints. By introducing restrictions on base types, subtypes of application types and semantic classification of attributes, the application can ensure a wide range of semantic constraints. An ontology as presented in [14] could be beneficial and its hierarchy, which is formed based on the constituents of each context type, can be modelled appropriately by composed contexts. For example, the `Environment:Sound:Intensity` type presented in the above mentioned work could be modelled with the following context types in our model:

```
contextType environmentContext characterising environment {
        sound: soundContext;
};

contextType soundContext {
        intensity: intensity; // restriction of string , enumeration
        frequency: hertz; // restriction of integer
};
```

The example defines two context types — the *environmentContext* and the *soundContext*. The first is constrained to be bound only to application entities of type *environment* or subtypes of it. Thus we can ensure context information especially declaring the environment of application entities of a given application type. Each of the contexts has property values, those of *sound, intensity* and *frequency*, respectively. The attributes of the *soundContext* are restrictions of `BaseType`, while the *environmentContext* context is a composite context that uses the *soundContext*.

5 Context Acquisition

Now that we have discussed the abstract notion of context and its type model, we will present the lower-level interface of context provisioning. We use the notion of a sensor as an abstract concept to represent a provider of a particular context element, for example, the location-ambience context element that is composed from the subcontext elements of temperature and lighting. The provider can be a hardware sensor, as in case of the light and temperature, or a software sensor, responsible for combining temperature and lighting to provide ambience values. Thus a sensor could either be bound to a simple context or a composite context.

To allow reusability and type safety, each sensor is an instance of some well defined *sensor type*. The sensor type defines an operation that is responsible for providing the information for the context. This operation can optionally have some parameters of any type defined in the model, with the purpose of controlling the execution of the operation execution (see `sensorParameterTypes` in Fig. 3). Further, a sensor type defines the type of context for which the sensor should provide values. With this information, the system can check that a sensor will always provide information of the correct context type.

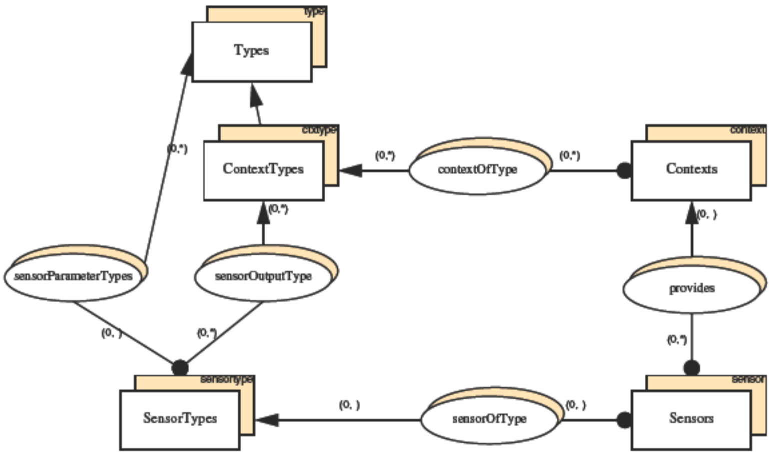

Fig. 3. Context Acquisition

The system provides a powerful algebraic query language based on set algebra that can be used for querying the system. The following constraint ensures that each sensor is only associated with a context of a type compatible to the sensor's output type:

```
( ("sensorOfType" compose "sensorOutputType")
     compose (inverse "contextOfType")
) minus "provides" = ∅
```

The left hand side of the constraint expression is a query that searches for sensors with output context type different than the type of the context bound to the sensor. The constraint checks that the result set is empty, meaning that there are no sensors bound to an incorrect output type.

Examples of sensors could range from as simple as a thermometer to complex pieces of software. To install a new sensor, the developer simply needs to initialise a sensor instance and bind the sensor to a given context. The sensor can either run individually and update the status of its context or it can be of a reactive nature, responding to an update request from its associated context.

6 Event Triggering and Application Binding

We discussed previously the fact that each context characterises an application entity. This is the only link between the application and the context system. Based on the well defined context model and typing, an application can easily access the context of interest. On the other hand, in reactive applications, it is necessary to adapt the application behaviour upon some context alternation.

We therefore need a mechanism for notifying parties interested in a given context change. Analogous to the concept of producers, we incorporate into our model the notion of notifiers. They notify subscribed application entities upon some occurring context event. As seen in the event model of Fig. 4, a notifier is subscribed to a context. Each time the context changes, the notifier is invoked. Based on some logic, the notifier evaluates the context change and, if desired, the appropriate application entity is notified. The specific notifier logic can be implemented in any language that has access and an interface to the context model and the application domain. In ongoing work, we are investigating a language based on boolean expressions and application callbacks to assist the notifier logic.

Again, our model allows the declaration of abstract notifier classes, based on which, multiple notifier instances can be instantiated. Imagine, for instance, a notify class that is subscribed to a room context. Whenever the lighting conditions change, the notifier is invoked. The notifier checks if the value is above or below a certain threshold and, if so, notifies the associated application entity by passing the corresponding context.

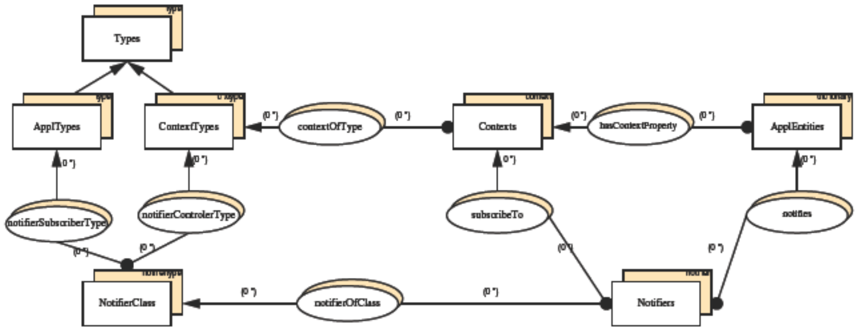

Fig. 4. Event triggering and application binding

It is not the purpose of our model to describe how to physically react to the incoming changes of context. These changes are propagated to the interested entities in the application layer, which are responsible for taking the appropriate actions. This feature makes our solution compatible with the active context awareness model presented in [16].

7 Conclusions

We have presented a general and abstract conceptual model for context and consolidated it with related work from various domains. The model supports a flexible and semantically expressive context representation that can be used for various applications scenarios. A sophisticated type system enables consistency checks at the system level and enables the definition and exchange of context information, something missing in the diverse data structure approaches used in earlier works. With the proposed model, it is possible to add new context elements at any time, and define new behavioural patterns for them, resulting in a very flexible and adaptable solution. Additional flexibility gives us the possibility to use object references as context values rather than just simple basic values. We believe that the proposed model contributes in the understanding and representation of context and could offer a common information model, based on which, application specific context models and ontologies could be defined.

Future work will include the implementation of the context model and its integration in a platform for ubiquitous and mobile information environments. We are currently developing a context-aware news application with a specific application context model based on the current proposal.

References

1. Anind K. Dey and Gregory D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of the CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness*, The Hague, Netherlands, apr 2000.
2. Luciano Baresi, Devis Bianchini, Valeria De Antonellis, Maria Grazia Fugini, Barbara Pernici, and Pierluigi Plebani. Context-aware composition of e-services. In *Proceedings of 4th International Workshop on Technologies for E-Services, 4th International Workshop (TES)*, pages 28–41, Berlin, Germany, September 2003.
3. Stefano Ceri, Florian Daniel, and Maristella Matera. Extending WebML for Modeling Multi-Channel Context-Aware Web Applications. In *Proceedings of MMIS'2003, International Workshop on Multichannel and Mobile Information Systems*, December 2003.
4. Anind K. Dey, Daniel Salber, Gregory D. Abowd, and Masayasu Futakawa. The conference assistant: Combining context-awareness with wearable computing. In *ISWC*, pages 21–28, 1999.
5. Anind K. Dey, Daniel Salber, Masayasu Futakawa, and Gregory D. Abowd. An architecture to support context-aware applications. Technical report, Georgia Institute of Technology, June 1999.
6. Paul Dourish. What we talk about when we talk about context. *Personal and Ubiquitous Computing*, 8(1), 2004.

7. Michael Grossniklaus and Moira C. Norrie. Information Concepts for Content Management. In *Proc. Intl. Workshop on Data Semantics in Web Information Systems (DASWIS 2002)*, Singapore, December 2002.
8. Michael Grossniklaus, Moira C. Norrie, and Patrick Büchler. Metatemplate Driven Multi-Channel Presentation. In *Workshop on Multi-Channel and Mobile Information Systems, WISE 2003*, Roma, Italy, December 2003.
9. Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.
10. Ramanathan V. Guha. *Contexts: A formalization and some applications*. PhD thesis, Stanford University, December 1991.
11. Martin Heidegger. *Sein und Zeit*. Neomarius Verlag, Tuebingen, 1927.
12. Jason I. Hong and James A. Landay. An Infrastructure Approach to Context-Aware Computing. *Human-Computer Interaction*, 16, 2001.
13. Cory D. Kidd, Robert Orr, Gregory D. Abowd, Christopher G. Atkeson, Irfan A. Essa, Blair MacIntyre, Elizabeth D. Mynatt, Thad Starner, and Wendy Newstetter. The aware home: A living laboratory for ubiquitous computing research. In *Cooperative Buildings*, pages 191–198, 1999.
14. Panu Korpipää and Jani Mäntyjärvi. An Ontology for Mobile Device Sensor-Based Context Awareness. In *CONTEXT 2003, LNAI*, volume 2608, pages 451–458, Berlin, 2003. Springer-Verlag.
15. Henry Lieberman and Ted Selker. Out of context: Computer systems that adapt to and learn from context. *IBM Systems Journal*, 39(3):617–631, 2000.
16. Louise Barkhuus and Anind K. Dey. Is Context-Aware Computing Taking Control Away from the User? Three Levels of Interactivity Examined. In *UbiComp 03 Conference*, Seattle, oct 2003. To Appear.
17. John McCarthy. Notes on formalizing contexts. In Ruzena Bajcsy, editor, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 555–560, San Mateo, California, 1993. Morgan Kaufmann.
18. Surav Mehmet and Akman Varol. Modeling context with situations. In *Proceedings IJCAI-95 Workshop on Modelling Context in Knowledge Representation and Reasoning*, pages 145–156, Montreal, Canada, 1995.
19. Moira C. Norrie. A specification of an object-oriented data model with relations. In *Proceedings of International Workshop on specifications of Database Systems*, pages 211–227, Glasgow, July 1991.
20. Moira C. Norrie. An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In *Proceedings of ER'93, 12th International Conference on the Entity-Relationship Approach*, December 1993.
21. Moira C. Norrie and Alexios Palinginis. A Modelling Approach to the Realisation of Modular Information Spaces. In *14th Conference on Advanced Information Systems Engineering (CAiSE'02)*, May 2002.
22. Moira C. Norrie and Alexios Palinginis. Empowering Databases for Context-Dependent Information Delivery. In *Ubiquitous Mobile Information and Collaboration Systems (UMICS), CAiSE Workshop Proceedings*, June 2003.
23. Moira C. Norrie and Alexios Palinginis. Versions for context dependent information services. In *Conference on Cooperative Information Systems (COOPIS 2003)*, Catania-Sicily, Italy, November 2003.
24. Moira C. Norrie, Alain Würgler, Alexios Palinginis, Kaspar von Gunten, and Michael Grossniklaus. *OMS Pro 2.0 Introductory Tutorial*. Institute for Information Systems, ETH Zürich, March 2003.

25. Jason Pascoe. Adding Generic Contextual Capabilities to Wearable Computers. In *ISWC*, pages 92–99, 1998.
26. Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, pages 74–83, Oct-Dec 2002.
27. Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99)*, pages 434–441, Pittsburgh, PA, May 1999.
28. Bill N. Schilit. *A System Architecture for Context-Aware Mobile Computing*. PhD thesis, Columbia University, 1995.
29. Bill N. Schilit, Norman Adams, and Roy Want. Context-Aware Computing Applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, 1994.
30. Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.
31. Terry Winograd. Architectures for Context. *Human-Computer Interaction*, 16, 2001.