

# SOPHIE

# Context Modelling and Control

---

*Diploma Thesis*

**Rudi Belotti**

<rudi.belotti@switzerland.org>

Prof. Dr. Moira C. Norrie

Supervisors: Michael Grossniklaus  
Corsin Decurtins

Institute for Information Systems  
Global Information Systems Group  
Wintersemester 2003/2004

Version 1.0

Last Update 25th February 2004



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich





Ai miei genitori.



# Abstract

Context is an important information that can be used to enrich the existing computer applications generating several benefits. For instance, it provides new computing power in terms of an extended interaction with the user. Moreover, by automating the acquisition of the relevant information it can reduce the need to interrupt the user. In the field of information environments, context information can be used to enrich the existing content. Consequently, it will enable the smart delivery of information to the right person at the right time.

Although context-awareness has been investigated initially for ubiquitous computing applications, there are numerous other areas of computer science that can take several advantages from context-awareness. In this work we focus in particular on the field of information environments.

The goal of this work is to present a comprehensive data model for context that can be used to describe context in database applications. The proposed model is very general and flexible, so that it is possible to add new contextual characteristics at any time. Moreover, our solution provides a mechanism for defining how to control the context information from its acquisition by means of appropriate sensors, to the delivery of the relevant contextual information to the interested entities by means of events.

Our model describes context in a general way and provides mechanisms that allow to control it. The whole system is driven by the database application, thus it is not necessary to hard-code the functionality on the application side. Moreover, it is possible to reuse several components. We demonstrate the generality and flexibility of our solution by means of a prototype implementation.



# Acknowledgments

While working on my diploma thesis I have received valuable assistance from numerous people. I first thank Prof. Moira Norrie for her dedication to the Global Information Systems Group at the Swiss Federal Institute of Technology, and for offering very interesting and challenging student projects.

I am very grateful to my supervisors Michael Grossniklaus and Corsin Decurtins for their valuable support and appreciable discussions during the whole time of my project. I would like to thank Alexios Palinginis for his precious hints and advices; he kindly helped me every time I needed support with the OMS Pro database management system.

I express my gratitude to Marco Dubacher for providing the source code of the `UserTracker.java` that I used for the integration of a web cam and the ARToolKit [ART04] as pattern recognition sensor within the prototype system. Besides, I am thankful to all members of the GlobIS Group for the precious feedback and discussion during my presentations.

Ringrazio in modo speciale Gabriella per essermi stata vicina, offrendomi interessanti spunti di riflessione e la motivazione necessaria per raggiungere gli obiettivi prestabiliti; inoltre per aver letto questo lavoro dandomi preziosi consigli.

Questo lavoro è dedicato ai miei Genitori, i quali mi hanno permesso di intraprendere questo cammino, aiutandomi e sostenendomi in ogni momento. Vi ringrazio per l'educazione che mi avete dato, per avermi insegnato ad apprezzare le vittorie, e ad accettare anche le sconfitte, traendone insegnamento e non averne paura. Grazie di cuore.

Zurich, 25th February 2004

—Rudi Belotti



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Context</b>	<b>5</b>
2.1	Definitions . . . . .	5
2.2	Taxonomies and Formats . . . . .	7
2.3	Perspectives for Computer Applications . . . . .	8
2.4	SOPHIE Needs Context-Awareness . . . . .	9
<b>3</b>	<b>The Context Architecture</b>	<b>11</b>
<b>4</b>	<b>The Context Model</b>	<b>15</b>
4.1	The Need of a General Model . . . . .	15
4.2	Context Information Model . . . . .	17
4.3	Acquisition of Context . . . . .	18
4.4	Context Aggregation . . . . .	19
4.5	Event Notification . . . . .	20
4.6	Model Types and Interface . . . . .	22
4.7	Overview of the Full Model . . . . .	23
<b>5</b>	<b>Prototype Implementation</b>	<b>25</b>
5.1	Binding Executable Code to Database Objects . . . . .	25
5.2	Aggregate Context . . . . .	28
5.3	Definition of Events . . . . .	30
5.4	Sensor Abstraction and Simulation . . . . .	31
5.5	Connecting a Real Sensor . . . . .	33
5.6	Quality Information . . . . .	34
5.7	Prototype Overview . . . . .	35
<b>6</b>	<b>Future Work</b>	<b>37</b>
<b>7</b>	<b>Conclusion</b>	<b>41</b>
<b>A</b>	<b>Glossary</b>	<b>43</b>
<b>B</b>	<b>Method Description</b>	<b>45</b>
B.1	Context Producers . . . . .	45
B.2	Mappers . . . . .	45
B.3	Context Elements . . . . .	46
B.4	Context Operators . . . . .	47

B.5 Context Events . . . . .	47
B.6 Macros . . . . .	48
<b>Bibliography</b>	<b>49</b>
<b>Index</b>	<b>59</b>

*There are more things in heaven and earth, Horatio,  
Than are dreamt of in your philosophy.*

—William Shakespeare, *Hamlet*

# 1

## Introduction

Context-aware computing is a relatively new field of research, however many books and papers on this subject have already been published. Context is a very important concept that is relevant not only to computer applications but also to a number of other disciplines in the field of humanistic sciences such as linguistics, philosophy and sociology.

Our senses continuously capture a huge quantity of information, which is then processed and elaborated by our brain. All this information, together with the history of precedent information, our memory, and our social relations, build the context we are related to.

We implicitly change our behaviour depending on the situation, in relation to the environment, or more generally, with respect to the actual context we are in. The context is the driving force behind our ability to decide what is important and what is not. It allows us to enrich our knowledge about a certain situation by drawing from related memories and experiences. Hence, only through the context are we able to manage the enormous amount of information that surrounds us at any instant. One typical example is the use of the context to resolve the meaning of polysemous words [Wid03] when reading or participating at a conversation. Actually, most of the words have multiple meanings that can be determined only when they are placed in a sentence, or even in a larger context. You can easily check this by searching for the meaning of a word of your choice in the dictionary.

Our context-awareness is constantly active, every second of our life, even if we do not think about it. And it is so powerful because we are somehow unaware of this awareness. When we are well trained in doing something such as driving a car or using a mobile phone, we are allowed to perform these actions without thinking. Consequently, things that we learned sufficiently well disappear and we can focus beyond them on new goals [Wei91]. Thus, why not try to use the context information that surrounds us to build better computer

applications, and therefore provide new computing power to the user in terms of smart software, that helps us when dealing with an enormous quantity of information. Moreover, context-aware applications can adapt their output according to the current situation, limit the need for input, therefore reduce the need to interrupt the user. If the interruption of the user is really needed, then the application has to find a good time to interrupt him, and already reduce the selection space [Sch00] facilitating the user's choice.

The notable importance of context is also demonstrated by the great amount of research and publications available in the humanistic fields of linguistics and philosophy [Wid03]. More recently many interesting research works have also been published in the field of computer science. These works are mainly in the areas of ubiquitous and pervasive computing [PRM99, DA00, Dey01, BPR01, Pas01, HIR02, KM03, CFJ03], human-computer interaction (HCI) [SBG99, Sch00, TGF03], and artificial intelligence (AI) [MPN01, BBG01, GM03].

On one hand context seems to be implicitly well understood and recognised; on the other hand, context-awareness is a very challenging and difficult task to achieve for computers up to now. Some good results are available in the field of the graphical user interfaces (GUI), but these are not fully satisfactory, particularly because it is not possible to apply them directly for other purposes.

We suppose that many of us dream of better computing applications that have at least an inkling of what we are doing or where we are working and that adapt themselves accordingly. This adaptation can be realised in various forms. For instance, the user interface could adapt to the situation by presenting the toolbars which offer the instruments that are needed to perform the next task we have in mind. A more complex example would include the delivery of useful and relevant information via mobile devices such as mobile phones or personal digital assistants to support expedient collaborative work. For instance, a PDA could be used to display background information about the knowledge and know-how of a co-worker whenever a new person enters the room. On the other hand, mobile phones, which represent the most used and successful mobile-computing application up to now, could simply adapt their ringing profile automatically depending on the situation. In this example context would allow to filter out all calls that are irrelevant to the current setting, such as an important meeting.

One possible approach to the aforementioned examples is to design a corresponding application that suits the specific requirements in each single case, but this is not convenient since the same problems are encountered many times. Moreover, this solution results in less flexible applications and code reusability is compromised.

The purpose of this work is not to develop just another context-aware application, but to investigate the research on context and context-aware applications in various disciplines, determine the common characteristics and consolidate them aiming at a higher abstraction model to describe context as information in a very generic way. Hence to provide a model for context that could be easily reused in many applications.

Chapter 2 presents various definitions of context that can be found in literature and discusses them briefly in order to better understand the meaning of this central concept. Furthermore it contains an overview of benefits that

can be obtained by context-awareness. In chapter 3 we illustrate a structured architecture for dealing with context information that is based on four main principles, i.e. *context acquisition*, *context augmentation*, *contextual adaptation*, and *contextual resource discovery*. The context data model developed within this project is presented in chapter 4. We motivate the need of a general data model with an application specific approach, and then we present and discuss the main components of the developed model separately. Chapter 5 considers the prototype implementation details. It demonstrates how the developed model can be employed and how the context information can be defined and controlled within the model. Furthermore it describes how it is possible to integrate external sensors into the system. We outline how this work can be extended further by indicating directions for improvements and future work in chapter 6. Final remarks and conclusions are given in chapter 7.



*There is more to context than location.*

—Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen [SBG99]

*Delivering the right information, to the right person at the right time.*

—Global Information Systems Group, ETH Zurich

# 2

## Context

Considering the numerous publications that have already addressed the meaning of context, we think that it is worth examining them briefly. Therefore, in this chapter we consider several definitions of context that can be found in literature in order to better understand what context means, how it can be classified and what are the benefits for the computing applications implementing context-awareness. In section 2.4, at the end of the chapter, we discuss the needs and the goals for the *Social Philanthropic Information Environment* project.

### 2.1 Definitions

As starting point, we propose to consider the definitions of context that can be found in the English dictionary. Although these definitions are especially related to the language semantics, they are useful to get a general overview of the meaning of this relevant concept.

1. Etymology: Middle English, weaving together of words, from Latin *contextus* connection of words, coherence, from *contexere* to weave together, from *com-* + *texere* to weave 1: the parts of a discourse that surround a word or passage and can throw light on its meaning 2: the interrelated conditions in which something exists or occurs: environment, setting.<sup>1</sup>
2. (Text/Speech) The text or speech that comes immediately before and after a particular phrase or piece of text and helps to explain its meaning: *In this exercise, a word is blanked out and you have to guess what it is by looking at the context.* (Cause of event) The situation within which something exists or happens, and that can help explain it: *It is important*

---

<sup>1</sup>from Merriam-Webster's Online Dictionary, <http://www.m-w.com>, November 27, 2003

*to see all the fighting and bloodshed in his plays in historical context. This small battle is very important in the context of Scottish history.*<sup>2</sup>

3. 1. In archeology, the context (physical location) of a discovery can be of major significance. 2. In linguistics, context refers to the meaning of a word or phrase with regard to its place in the sentence and the topic being discussed. 3. In computer science, context refers to the circumstances under which a device is being used, e.g. the current occupation of the user.<sup>3</sup>

From the aforementioned definitions, we can recognise that *context* is a term with a set of meanings that can be very general or more specific, depending on the application field, i.e. according to the context in which it is used. We can also find similar definitions in previous research works in the field of context-aware computing. For instance, [SBG99, BPR01] define context as in *The Free Online Dictionary of Computing*<sup>4</sup> as *that which surrounds, and gives meaning to something else*. This definition is very general and similar to the one quoted in the *Cambridge Dictionary* (item 2).

Another definition can be found in [HL01a], where context is considered as the “W”, who, what, where, when. Although it is expressed in another form, it is not so different from the previous one and from the one in [HIR02] that defines context as *the circumstances or situation in which a computing task takes place*. Moreover, they consider that the context information is sensed by hardware or software sensors, or derived from other sources, e.g. from the user’s agenda. The context-aware application can obtain several information, such as important meetings and projects schedules, hence it can adapt accordingly by providing the relevant resources to the user.

Many works have tried to give a definition of context; however most of them were unsuccessful because they simply used synonyms such as environment or situation. On the contrary, Dey and Abowd [DA00] have given a new general and useful definition of context, which is applicable to the realm of computer applications:

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

A similar definition is also stated by Chalmer [Cha02]. Besides, the authors of both publications define a context-aware system as *a system that uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task*. In [PRM99], context-awareness is also explained with a separate definition as the ability of a device or a program to sense, react or adapt to its environment of use.

<sup>2</sup>from the Cambridge International Dictionary of English, <http://dictionary.cambridge.org>, November 27, 2003

<sup>3</sup>from Wikipedia, the free Encyclopedia, <http://www.wikipedia.org>, November 27, 2003

<sup>4</sup><http://www.foldoc.org>, February 2, 2004

All these definitions are similar, but at the same time they differ slightly in their generality. One of the most important commonly acknowledged points is that context is information that describes other information, therefore it can be modelled as such.

## 2.2 Taxonomies and Formats

The first work exploring the area of the context-aware computing is the *Olivetti's Active Badge* system in 1992 [WHFG92]. The Active Badge system exploits the person location to forward your phone calls to the nearest telephone, i.e. a telephone situated in the room you are currently in. Afterwards, many works followed in the field of ubiquitous computing, which have considered context only as the location of the user. A survey of the research in field of context-aware computing is offered by [CK00]. Of course, the survey is not complete due to the amount of work in this area, but it gives an idea of the different approaches taken by several applications.

Nowadays, the notion of context has evolved also for computer scientists to a wider and more complete concept that takes into account more characteristics, in particular thanks to [SBG99, PRM99, Sch00].

Previous investigations have also individuated several groups that permit to categorise context. According to Dey and Abowd [DA00], the most important categories are location, identity, activity and time. The authors of [TGF03] choose a slightly different terminology by describing beyond location the *physical context*, addressing properties like the temperature, the lighting, and the noise of a given environment. A second category is represented by the *user context*, which describes, among other information, the user's identity, his profile, his social situation, and his activity. A third class is called *computing context*, representing the system's connectivity, the communication costs, and also bandwidth and information about other resources. Figure 2.1 summarises the aforementioned dimensions of context and their properties.

According to [HIR02], context can be represented in various forms. For instance, we consider time and location. Time can have multiple representations: the absolute format in terms of a timestamp, or many other representations that provide smaller resolution units such as AM/PM, week, month or season. The same mechanism can be adopted in the case of location, which can be described in terms of GPS coordinates, or by means of room names, floor numbers or building information.

The different context representations depend on the abstraction level in which the information is used. However, it is important to collect the contextual information with the finest possible resolution, in order to enrich the content. This is relevant for instance if we consider the user location context, because we can derive the building information (e.g. IFW) from the room name (e.g. IFW D35), but it is impossible to infer the room name from solely the building information.

The various context categories present different characteristics. Among these, it is important to notice that the meaning of context information may





- **Computing context**
  - Connectivity
  - Communication costs
  - Bandwidth
  - Resources
- **Time**
  - Absolute time
  - AM/PM
  - Week
  - Month
  - Season
- **User context**
  - Identity
  - Profile
  - Social situation
  - Activity
- **Physical context**
  - Temperature
  - Lighting
  - Noise

Figure 2.1: Several dimensions of context

depend on several cultural and subjective parameters [KM03], e.g. the temperature representation in degrees is objective, but whether it is cold or warm, it depends also on the humidity of the environment and several individual parameters. Thus, the context information should be processed in an objective way until the last step, when it is delivered to the final recipient. Only then the objective information can be interpreted and personalised according to the final consumer's profile.

The authors of [GBSV03] investigate context-awareness, by classifying it in two classes called *hard context-awareness* and *soft context-awareness* in order to indicate the physical parameters measured by hardware sensors and the information gathered by software sensors monitoring the information base, services and applications respectively. Furthermore, they illustrate the advantages of *soft context-awareness*; this approach in particular, does not need any integration of specific hardware sensors. They demonstrated that such a system can be successful in highly interconnected environments, achieving context-awareness only by using the information provided in an enterprise application.

## 2.3 Perspectives for Computer Applications

Mark Weiser [Wei91] already affirmed that computers can significantly adapt their behaviour only by knowing which room they are in, even without any use of artificial intelligence. Context-awareness is the key enabler of smart environments, because it allows to render the computing process invisible, opening the field for pervasive computing [BPR01] and new technologies that are aiming at applying computers in social and more dynamic environments other than the desktop [PRM99].

For our purposes, context represents the enabling technology that allows to select and present the relevant information to the interested recipients at the right time, on various output channels [NP03b]. For instance, the relevant information about a given project can be automatically displayed on wall-mounted screens when the members of the project team enter the laboratory at working hours, whereas when lunch time is approaching the current menu proposal of the local cafeteria is displayed according to the user context. The use of context can be extended further, by showing only meat-free dishes for vegetarians.

Examining this relatively simple example, we recognise how rich the information that surrounds us is, which is not considered by the existing computer applications up to now. However, today many techniques are offered allowing us to collect such information automatically. Thus, we need a technology that allows to integrate these systems in order to exploit the context information to improve the services offered to the user.

A good solution enabling the integration of several kinds of sensors is provided by the existing *context frameworks*, which have been developed to provide a general technology facilitating the development of context-aware applications. A well known representative is the *Context Toolkit* [SDA99, Dey00], which has been used for many projects such as the simple applications *In/Out Board*, *DUMMBO* and the more complex project called *Aware Home* [KOA<sup>+</sup>99].

If on one side these frameworks provide a good solution for the integration and abstraction of different sensors; on the contrary, the context information they provide still lacks a semantic interpretation. For this reason, a data model that describes the context information is suitable in order to define its semantics. Moreover it represents the enabling technology for enhanced context-awareness in information environments.

## 2.4 SOPHIE Needs Context-Awareness

SOPHIE is the Social Philanthropic Information Environment. It is an integrated and reactive information environment for the GlobIS laboratory. The strength of this system resides in the fact that it is aware of the surrounding environment, in particular of the presence of humans and physical objects within the laboratory. SOPHIE can keep track of the continuous changes taking place in the environment and adapt to them accordingly, for instance by delivering the right information to the various output channels depending on several contextual information. In order to enable the context-dependent delivery of information, the data model and the context model must not be separated, allowing any content information to change and adapt to context. This means that not only is the presented information adapted, but also the corresponding representation and the appropriate output channel are chosen accordingly.

The system has to fulfill several requirements to work appropriately. In order to know the properties of the surrounding environment, the first task that the system has to achieve is to acquire the relevant information. Typically this is performed by an explicit interaction with the user. Unfortunately this is not satisfactory for our purposes because we think that a great part of the

relevant information is already available, stored within some database, or can be collected automatically by the employment of appropriate hardware or software devices. Thus, the system has to be able to collect relevant pieces of information automatically and without the explicit intervention of the user, reducing the need for interruption. Furthermore, the automatic acquisition of the relevant information decreases the need for a direct user intervention, facilitating his tasks. Moreover, it is often very difficult for the user to know which information is really relevant to the application [DA00] in order to guarantee its proper functioning. This particular task is achieved by the sensory part of the system, which should be as flexible as possible and therefore allow to easily integrate new sensors [HL01b].

After the relevant contextual information has been collected, it has to be processed and augmented [Pas98] by setting the correct associations between the relevant information. This is a very central point, because the context always refers to some entity [DA00, GS01], which is the subject of the gathered information. Consequently, this mechanism allows to retrieve the relevant application objects by taking advantage of their contextual information. It is important to notice that context is a very general concept and only the application developer can decide what kind of contextual information is needed. Therefore, our model has to be as general as possible, and has to provide the means to describe any type of context and the mechanisms that facilitate its processing and management.

In order to enable the information to adapt itself according to the current context, the model has also to supply components, which allow to control and notify incoming changes or events to the interested entities, making them aware of the current context. Furthermore, this feature allows to bind the augmented context with the contextual adaptation part and the application objects that are responsible for controlling the different output channels such as public information displays, lighting, music and so on.

*If you want to build a ship, don't herd people together to collect wood and don't assign them tasks and work, but rather teach them to long for the endless immensity of the sea.*

—Antoine de Saint-Exupéry

# 3

## The Context Architecture

In this chapter we will present how the different tasks and properties of a general context-aware system can be organised into an architecture that allows to distribute the responsibilities to different components in order to build a structured system.

The proposed architecture is illustrated in figure 3.1 and is based on the terminology introduced by Pascoe [Pas98]. It consists of four main abstractions, i.e. *context sensing*, *context augmentation*, *contextual adaptation* and *contextual resource discovery* that are placed between the environment and the application layers.

The ability of the system to sense and acquire new contextual knowledge is embodied by the context sensing layer, which is placed just on top of the environment, and can encompass several kinds of either hardware or software sensors. In this level of abstraction, it is important to generalise the context acquisition process, offering a socket that allows to plug-in any kind of sensors that provide relevant information. The environment is considered in a very abstract way, in fact we can imagine the environment as the physical properties of the current location, but it would be too restrictive. Therefore, the environment also encompasses the currently available information base, which is very important and can be gathered by appropriate sensors and used as context. The *Context Toolkit's widgets* that are presented in [DSFA99, DMAC02] provide a useful abstraction mechanism to work with several kinds of sensors. The ability to model the context producers allows us to automate the information collection process and consequently to abridge the need for explicit input by the user, so that he will be seldom interrupted, and consequently he can concentrate on his work.

When a sensor measures a new value, the sensed data are passed to a higher abstraction level, where the context is augmented by considering its semantic properties, and it is associated to its subject according to Gray and

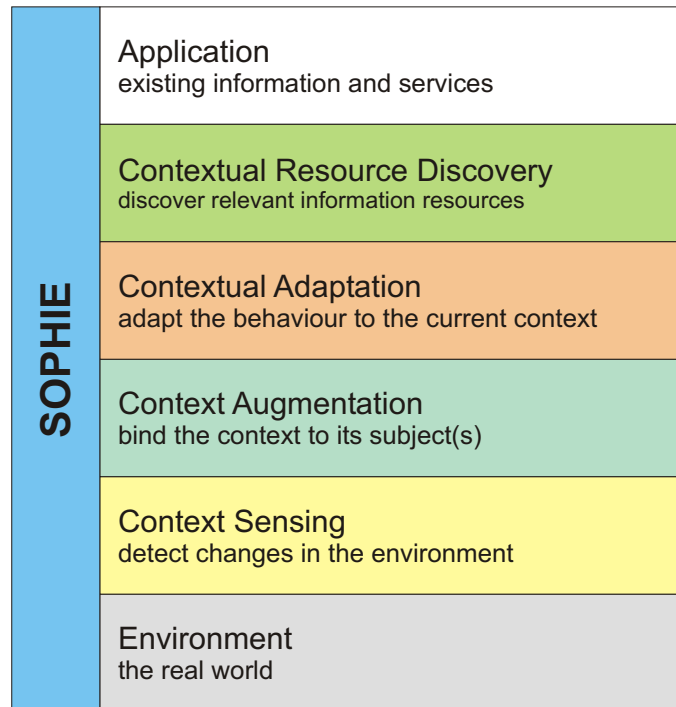


Figure 3.1: Context architecture

Salber [GS01]. Therefore, the context augmentation layer is very important, because it is responsible to semantically interpret the context information and also to add contextual value to the existing content by associating them. The context augmentation mechanism is very important because the context information does not exist independently, in fact it is always associated to some entity, i.e. its subject.

The next step after the correct interpretation of the context value, is to adapt the system according to the new situation. The contextual adaptation abstraction layer allows for instance to choose between alternative output channels or select the appropriate language for the current user or group of users. Besides, within the adaptation layer it is possible to control also the underlying sensing layer. For instance, it should be feasible to adapt the sensors' parameters to the current context increasing the measurement quality, or switch faulty sensors off in order to enhance the system's reliability.

A further abstraction is called contextual resource discovery. This layer is responsible to select additional resources and information, such as finding the nearest local printer for your mobile application or fetch additional detail information related to the same context from other sources that may be either external databases or information services.

A concrete example of the behaviour of a context-aware information system is depicted in figure 3.2. The system collects the relevant context both from the sensors installed in the environment and from existing databases storing

the user preferences and resource profiles. Thus, it is possible to detect the presence of users within a given environment such as the GlobIS laboratory, the current time, the temperature and other arbitrary properties. Furthermore, it is possible to fetch more information about the current users by querying the user preferences and profiles database. A common language spoken by all users and a common project they are working at may represent the current context. The information environment exploits the current context to retrieve the relevant information about the projects that relate all the users currently in the laboratory and chooses a language that is comprehensible for all users, finally it delivers the right information correctly formatted to the appropriate communication channels as described by their profiles.

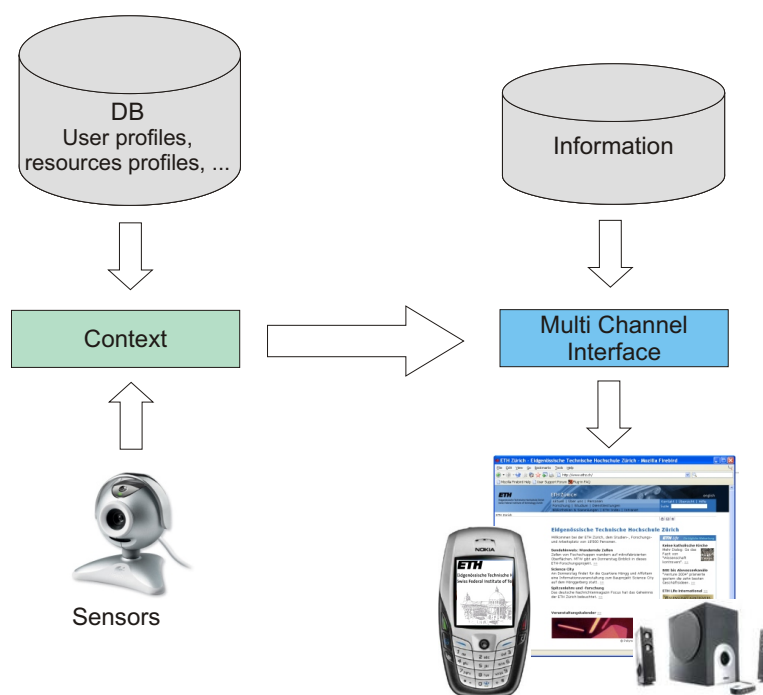


Figure 3.2: Context-aware information system

The delivery of information in the case of SOPHIE will include multiple communication channels such as visual displays and acoustic communication. Therefore, the database that controls the whole environment has to select the relevant content to display and accordingly choose an appropriate representation, e.g. HTML<sup>1</sup> or VoiceXML<sup>2</sup>. Furthermore, a smart environment could also detect the presence of personal devices such as PDAs and use them to deliver information, which is relevant only to the owner of this private device.

The automated information delivery gives rise to important information security issues. In particular, the system should not violate the user's privacy.

<sup>1</sup><http://www.w3.org/MarkUp/>, February 18, 2004

<sup>2</sup><http://www.w3.org/Voice/>, February 18, 2004

For instance it is important to define policies in order to provide the information only to authorised users on both public and personal devices. A simple solution that could be investigated, is to present the private information only on personal devices and never disclose it on public information displays.

*A model is a representation in a certain medium of something in the same or another medium. The model captures the important aspects of the thing being modeled from a certain point of view and simplifies or omits the rest.*

—James Rumbaugh, Ivar Jacobson, Grady Booch, *The Unified Modeling Language Reference Manual*

# 4

## The Context Model

This chapter describes the context model developed within the SOPHIE project. As stated in chapter 1, the main aim of this work is to present a comprehensive and general model for context, which is not limited to some context characteristics only. The proposed model deals with the acquisition, categorisation, augmentation and notification of context. Moreover it contains important meta-data describing several properties such as the source and the quality of a given context [WS01]. In the first section, we motivate the need of a general and comprehensive model for context by presenting an application specific model as a counterexample for the purpose to demonstrate the restrictions encompassed by similar solutions. In the following sections, we describe each component of our model in detail. The complete overview of the model is illustrated in figure 4.7 at the end of the chapter and the attribute definitions of the model types as well as the provided interface are shown in figure 4.6.

### 4.1 The Need of a General Model

Up to now, most of the attempts to realise context-aware computing applications adopted a similar approach: first they defined the characteristics that are needed to the specific application, and then they implemented all the processing of this information in the application code, encompassing aspects ranging from the data acquisition to the utilisation of this knowledge in order to enrich the application with context information.

Since a great part of the research in the field of context-aware computing has been supported by ubiquitous computing specialists, the context that is most considered in the technical literature is the user location, which has been used to develop interesting applications such as interactive museum tour guides [PNZ<sup>+</sup>01]. However, according to Schmidt *et al.* [SBG99, Sch00], *there is more context than location*. In response to this issue, some solutions such

as [CDM03] propose very specific adapted models that consider only a few more characteristics such as the user identity, his current activity, the device he is using and time.

The schema proposed in the figure 4.1 is an example of an application specific context data model that has been designed on the basis of an example scenario from [GBSV03]. The model describes the relationships between **Users**, **Applications**, **Locations** and **ServiceProfiles**. In particular, it models the use of a specific application that varies depending on the current user and location context, i.e. the application chooses its service profile according to the current user and the present location. Moreover, it provides the possibility to adapt itself according to the time by remembering the **lastUsedProfile**.

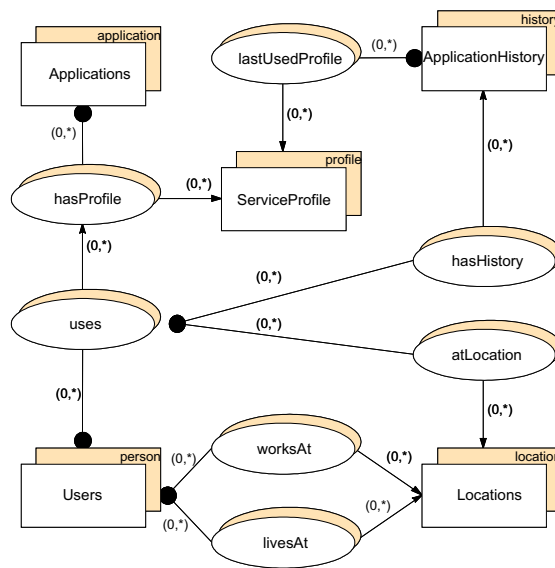


Figure 4.1: An application specific context data model

Although the aforementioned model already seems to be very powerful, it describes only a specific situation. For instance it would be impossible to model another kind of adaptation such as changing the application behaviour to suit the current lighting setting or depending on the agenda information of the current user. In order to achieve these goals, we would need to extend the model further including the appropriate information by means of other data collections and then rewrite the application code accordingly. However, this solution results in changes in both the application and the data model, which is tedious, not flexible and therefore undesirable.

One of the first steps towards the generalisation of the concept of context has been done by Dey *et al.* with the *Context Toolkit* [SDA99, Dey00], which is a framework that allows to integrate various sensory systems responsible to collect the context information and provide some basic facility to process the acquired data. However, the proposed solution lacks of a general data model for context, which can enrich the collected data by increasing its semantic value and transforming it into information. A very general approach is investigated

by the authors of [KM03]. Presenting an ontology for mobile context aware applications, they introduce new perspectives that allow to represent the context information in a very general way. Moreover, they also consider several meta-data, which play an important role for the effective description of context information.

In the next sections we present in detail all the components of the data model that we have developed to describe and control the context information.

## 4.2 Context Information Model

The context information is represented by means of three main components, i.e. the *element*, the *type* and the *instance* as depicted in figure 4.2. The *context element* describes the specific context abstraction such as the temperature in the room or the people in the laboratory. The values of these characteristics are subject to change depending on time and are modelled in terms of *context instances*, which describe the value of the associated context element. Each instance includes a timestamp attribute, in order to describe its temporal characteristics according to [HIR02] as shown in figure 4.6. Besides, they define a confidence attribute, so that it is possible to describe the accuracy of the values. The collection of context instances also defines the history of the values of a given context, which is similar to our memory. The cardinality constraint (1,1) on the target of the association *instances* ensures that each context instance is associated to exactly one element.

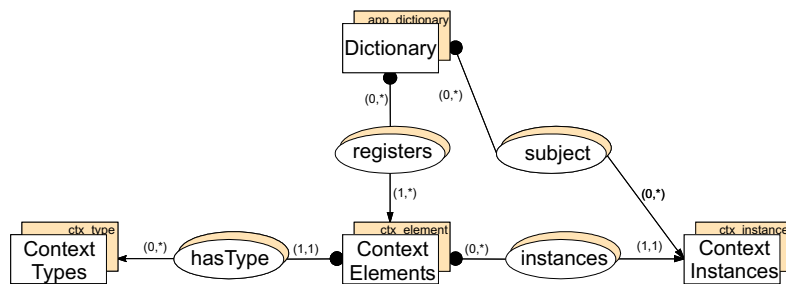


Figure 4.2: Core model of context

This approach is similar to the key-value pairs already presented in other solutions [NP03a, SG02] that is very powerful because in its simplicity it allows to represent potentially any kind of information.

Every context element is associated to one *context type* as defined by the

(1,1) constraint on the source of the association `hasType`. Hence it is possible to define types of context that allow to categorise several context elements with similar properties and operations. The context type defines a more general abstraction of the context elements, e.g. it is possible to define a common type for temperature elements, and subtypes to represent the measurement units such as temperature in Kelvin, or temperature in Celsius. For the sake of simplicity, in figure 4.2 the type model is reduced to the collection `Context Types`. The complete model for context types, supporting complex types and subtyping, is shown in figure 4.7.

The cardinality constraint (1,\*) on the target of the `registers` association ensures that the context information is always bound to some entities, which are represented by the `Dictionary` collection. These entities may be application objects, or other information stored in external databases. In the proposed solution we modelled them in term of a uniform resource identifier (URI)<sup>1</sup> attribute, whose referential integrity has to be guaranteed.

When a new instance is created, the entities that are registered to the correspondent context element are registered to the incoming instance as `subject`. Thus, many entities can dynamically subscribe the same context. Consequently, they will be associated to the relevant instances.

### 4.3 Acquisition of Context

The `Context Producers` collection in figure 4.3 represents the socket where it is possible to plug-in all kind of context providers such as a face recognition system that detects the users in a given environment or a thermometer. Each producer is associated with a `Quality` profile describing characteristics such as coverage, resolution, accuracy and frequency as proposed by Gray and Salber [GS01]. The producer's quality is a useful information to compute the confidence value for each sensed instance.

Although the simplest solution would be to consider the static accuracy attribute of the sensor as the instance's confidence, this is often not the best solution because the final measurement confidence depends also on other parameters such as the light intensity or temperature depending on the physical properties of the sensing device. Therefore, a policy to compute the instance's confidence should be defined appropriately for each type of provider or in particular cases for single provider instances.

The `Context Elements`, which are acquired by a context producer, are classified in the subcollection of context elements called `Sensed Context` [GS01], and the correspondent instances are related to their producer by means of the association `providedValues`. Hence, every instance of sensed context encompasses the meta-information about its source, which allows to detect potential problems, recalibrate imprecise producers or shut down malfunctioning ones.

Besides, it is possible to associate a `Mapper` to a producer for a specific context element. The goal of the mapper is to adapt the value that is provided by the producer in order to agree with the defined type of the context element,

<sup>1</sup><http://www.w3.org/Addressing/>, February 6, 2004

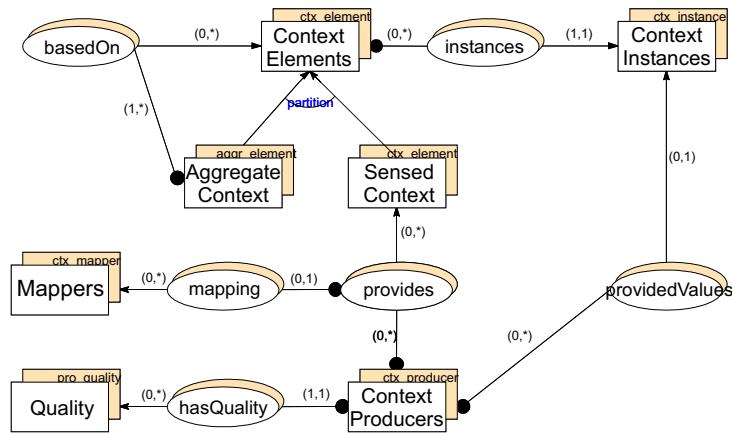


Figure 4.3: Context acquisition model

i.e. the mapper performs an important step towards the transformation of data into information. Its generality allows to define any kind of transformation that maps an atomic value into another one. Furthermore, a context mapper can be reused for many producers and elements.

It is important to notice that the context information can be both implicit and explicit. The implicit information is automatically collected by appropriate sensors as described before. The explicit context information is the result of a direct interaction between the user and the computer. In this case, the component that is responsible to collect the corresponding context information is modelled as context producer. Moreover, it may encompass several interaction channels such as the classical desktop environment or an audio communication based on speech recognition.

## 4.4 Context Aggregation

The `Context Elements`' collection is partitioned into two subcollections, the `Sensed Context` described in section 4.3 and the `Aggregate Context`. The aggregate context depends on other context elements as shown in figure 4.4 and allows their instances to be aggregated in order to create a new instance representing new information. For instance it is possible to combine the values coming from different sources concerning the same context, such as the temperature and compute an average value over the measurements of the last minute, or decide the correct identity of a person from uncertain data delivered by different person recogniser systems.

The appropriate operations are described in terms of `Aggregators` that are classified in a subcollection of `Context Operators`, which are reusable components, which perform operations over some `Context Type(s)`. The associations

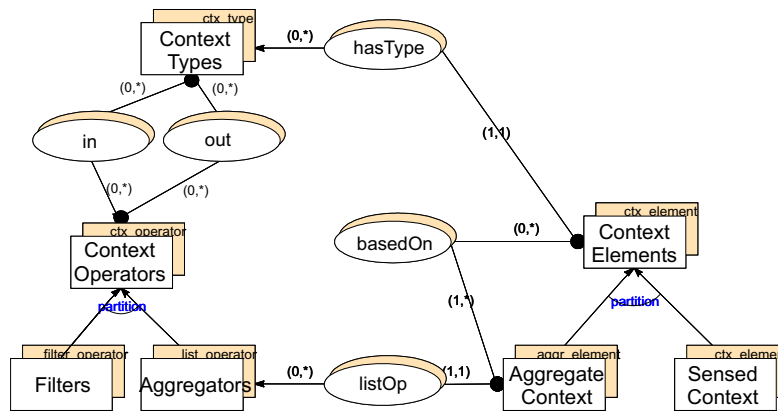


Figure 4.4: Context aggregation model

`in` and `out` describe their input and output parameters respectively. The aggregator operator takes a list of instances as input and produces a new instance, which is derived from the input.

The aggregate elements are notified and updated when a new value of a context they depend on is added to the collection of `Context Instances`. Since their instances are derived from other instances and not directly provided by a producer, they are not referenced in the association `providedValues`.

## 4.5 Event Notification

Context is used to enable the application to adapt and react to incoming changes in the environment. Hence, we need a mechanism to communicate with the interested entities, in order to inform them if some relevant change in context has occurred. It is important to define a mechanism that detects changes in context, because the sensors delivers their measurements periodically to the system. However, the application it is interested only in relevant changes in context and not in the whole set of values. Therefore, we need a mechanism that allows to filter out the irrelevant values and notifies only the relevant ones.

The `Events` are triggered only when the new context instances mean a significant change in the current context. These conditions are embodied by the `Filter` collection that is modelled as a partition of `Context Operators`. The filter can control several contexts and describe different situations. When an event is triggered, all the entities that are interested in the current context are notified.

There are two approaches to notify the changes in context to the application [SB00, BD03]. The first approach consist in implementing a passive context-aware system. In such case the context information is made persistent

for the applications, which regularly issue polling requests to retrieve the current context information and adapt accordingly. The second approach consists in allowing the context-aware system to interrupt the running application in order to dispatch the current notification, resulting in an active context-aware system.

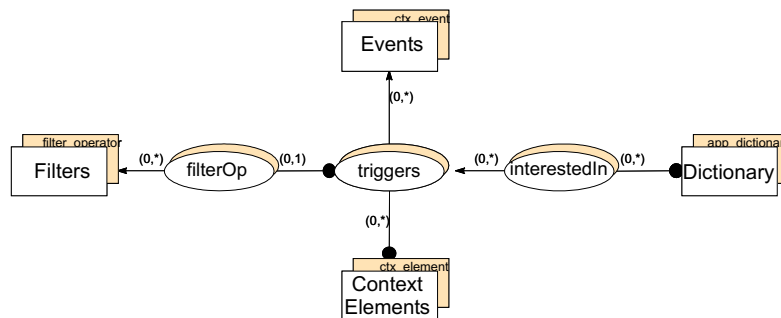


Figure 4.5: Event notification model

The proposed model enables both solutions by defining the incoming changes in context as events that hang on certain conditions associated to specific context elements as described in figure 4.5. Depending on the implementation requirements, the triggered event can be handled in both ways. Passive context-awareness can be easily achieved by generating a new instance of the event and adding it to an event queue, which should be regularly checked by the interested application. In this case, the events are made persistent for later retrieval. On the other hand, events can be implemented providing the ability to interrupt the application, realising an active context-aware system, which interrupts the application only on the incoming changes. Consequently, the application can promptly adapt and react.

The events are transformed into actions by the concerned application. Actions can be classified into three categories that indicate when they are executed [Sch00]. The first category encompasses the actions that are performed when a certain entity enters a given context. Similarly, actions can also be carried out when an entity leaves the current context. Additionally, there are actions that are performed while an entity is within a certain context, for instance at regular intervals.

## 4.6 Model Types and Interface

The model described in the previous sections defines its types and functions as shown in figure 4.6. The types are described by means of a UML [RJB99] static diagram that illustrates also the provided interface. We chose to represent this schema by means of UML because it provide a good overview of the type's attributes and methods, whereas the graphical OM model [NW97] does not allow to show these details. The basic types of the attributes are represented by means of the interface definition language (IDL)<sup>2</sup> notation.

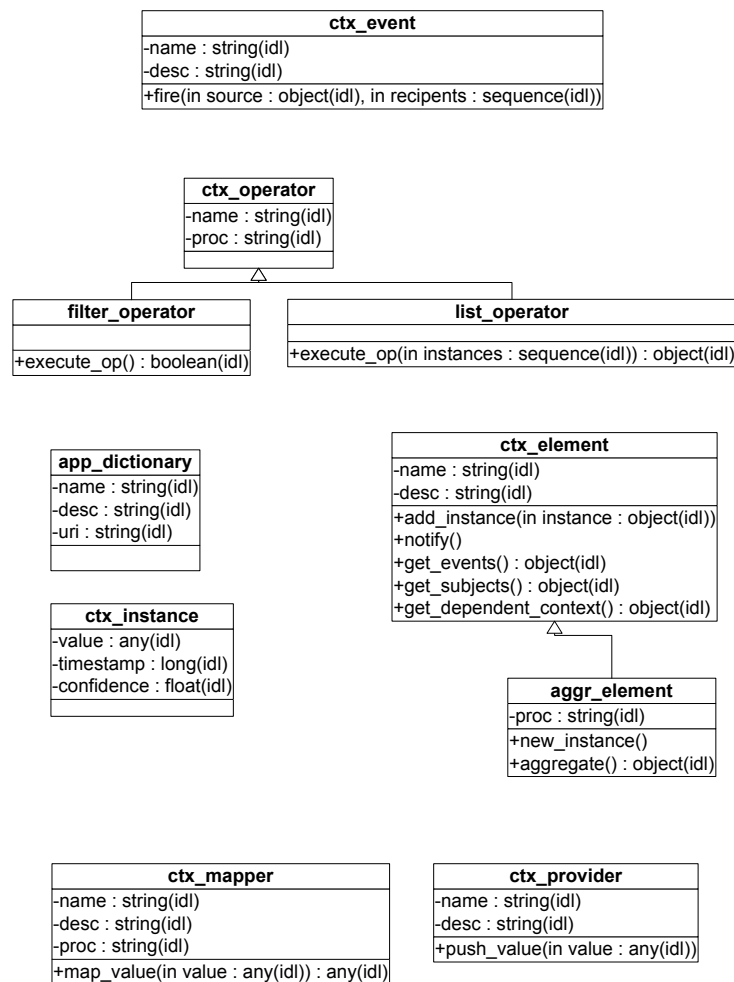


Figure 4.6: UML static diagram of the the model types

<sup>2</sup>[http://www.omg.org/gettingstarted/omg\\_idl.htm](http://www.omg.org/gettingstarted/omg_idl.htm), February 25, 2004

## 4.7 Overview of the Full Model

The components described in the previous sections cooperate as shown in figure 4.7, which represents the full OM model. Moreover, the full model illustrates the collection of context types in more details. The context type model is similar to the type model of the OMS Pro system. Therefore, if the context model has to be integrated into the database management system, we can bind the context model to the type model of the system to avoid duplicates.

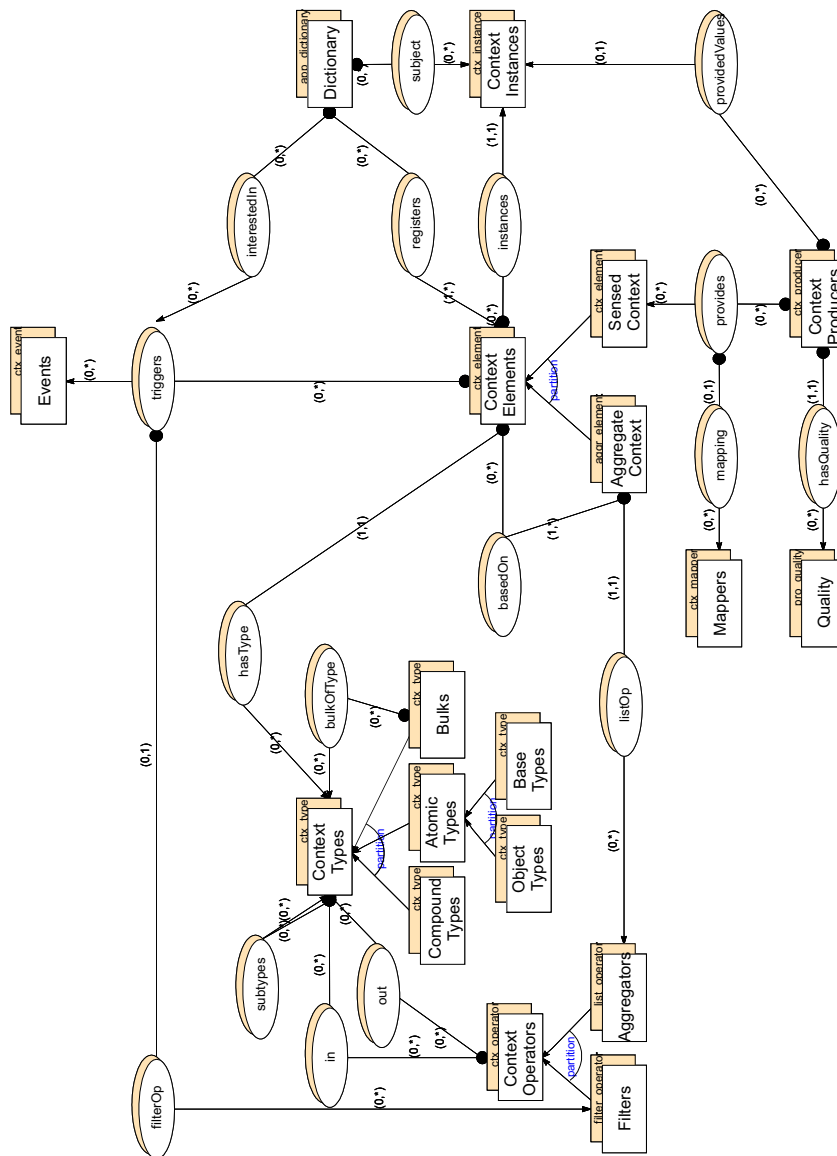


Figure 4.7: Overview of the full model



*Give me a lever long enough and a fulcrum on which to place it, and I shall move the world.*

—Archimedes

# 5

## Prototype Implementation

In this chapter we illustrate the details of the prototype implementation of the model discussed in chapter 4 in order to demonstrate its ability to describe and control the relevant context information. The prototype implementation is based on the object-oriented database management system OMS Pro [NWP<sup>+</sup>03, OMS04], which offers a rapid prototyping interface, and the Prolog<sup>1</sup> programming language. The description of the methods and macros implemented within the prototype is given in appendix B. An overview of the prototype architecture is presented in the figure 5.5 by means of a UML sequence diagram [RJB99] at the end of the chapter. The sequence diagram illustrates the interaction between the various components and demonstrates the prototype interface.

### 5.1 Binding Executable Code to Database Objects

The OMS Pro system allows to define a set of database operations in terms of macros, which are small applications on top of the database, methods and triggers that are bound to an object type and are called upon an object instance or a given event respectively. Methods and triggers are a kind of *class methods*, because they are bound to a specific object class. In the developed prototype, the four types `filter_operator`, `list_operator`, `aggr_element` and `ctx_mapper` define an attribute called `proc`, which allows to implement an *instance method* defined by means of the Prolog language. The instance method is bound to a specific object instance of the corresponding type. By using this feature, it is possible to create a set of objects instances of the same type that implements different methods. These objects represent reusable functionality elements, such as operations that can be performed on a set of instances as in the case of the `list_operator`, e.g. compute an average over a list of values. Another example is described by the transformations allowing to convert the

---

<sup>1</sup><http://www.sics.se/sicstus>, February 4, 2004

incoming sensor values to the correct internal representation, e.g. transformation of the temperature measurements from Celsius to Kelvin units as in the case of an instance of the type `ctx_mapper`.

In order to provide the ability to store and execute specific pieces of code for particular database objects, three main steps are required. First, we define the external procedure `ir_check_hook`, which is employed by OMS Pro to check the *intermediate representation*<sup>2</sup> of the binary code on the basis of its signature information. The lines 4-5 of the listing 5.1 describe the input and output parameters of the procedure defined in the `proc` field of the type `list_operator`, i.e. the procedure takes a set of `ctx_instances` as input and returns a new `ctx_instance`. This is done by means of an expression having the format `[(string,type,bulk)]`.

This technique is applied in a similar way also to the other three aforementioned types. For each type implementing this feature, you need to define a new `hook` predicate adapting the lines 4-5 according to the input and output parameters of the corresponding procedure. The definitions<sup>3</sup> are placed in the external source file `context_logic.pl` located in the database directory.

Listing 5.1: Intermediate Representation Check Hook

---

```

1 user:ir_check_hook(proc,Obj-list_operator,ir(prolog,IR),Cir)
  :-
2   Cir = cir(prolog,prolog(InT,OutT,IR1)),
3   user:exp(IR,IR1),
4   InP = [(instances,ctx_instance,set)],
5   OutP = [(cinstance,ctx_instance,uni)],
6   attr_props2defs(InP,InT),
7   attr_props2defs(OutP,OutT)
8 .

```

---

The second step consists in implementing a trigger on update [NWP<sup>+</sup>03] for the corresponding type as shown in listing 5.2. The trigger is responsible to invalidate the previously compiled code, which is in the system's cache memory, ensuring that the executed object code is always up to date. Hence, exploiting the flexibility provided by OMS Pro, it is possible to modify the code at any time, without the need of tedious manual recompilations. The second parameter of the predicate `invalidate_cached_bin` is an arbitrary unique identifier for the specific piece of code.

Listing 5.2: Trigger on update of the type `list_operator`

---

```

1 invalidate_list_operator_proc :-
2   self(Self),
3   invalidate_cached_bin(proc,Self-list_operator)
4 .

```

---

Finally, we need a method associated to the according type that is responsible to fetch, compile and execute the code that is stored within the current

<sup>2</sup>OMS Pro API specification, <http://www.oms.ethz.ch/omspro>, February 10, 2004

<sup>3</sup>Every time we change the hook implementation, it is required to restart the OMS Pro system.

database object. The method `execute_op` defined by the type `list_operator` in listing 5.3 is responsible to get the code from the `proc` field of the appropriate object, compile and execute it. The `execute_op` method is used to compute a new instance from a list of existing instances as shown in figure 5.1. The method's signature defines inputs and outputs according to the specific object code signature; in this case it takes a list of instances as input and returns a new single instance, as defined in listing 5.1. The second parameter of the `compile` rule on line 4 is the same unique identifier as in listing 5.2.

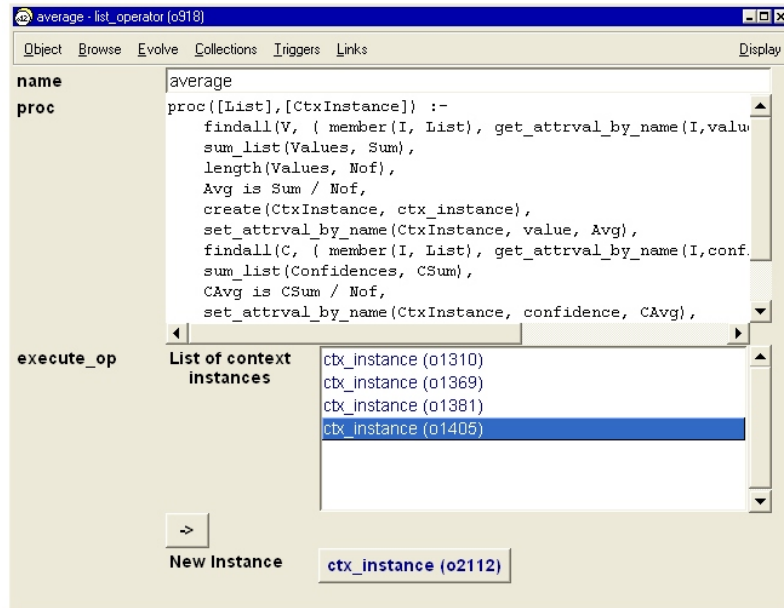


Figure 5.1: Instance of list operator

Listing 5.3: `execute_op` code of the `list_operator` type

---

```

1 method ([ List ], [ Outputs ]) :-
2     self ( Self ),
3     get_attrval_by_name ( Self , proc , Proc ),
4     compile ( proc , Self-list_operator , Proc , Bin ),
5     execute_proc ( Bin , [ List ], [ Outputs ] )
6 .

```

---

As an example of `list_operator`'s functionality consider the listing 5.4. The proposed code is responsible to calculate the average value from a list of instances. The computed average is then stored as `value` in the new created instance (lines 5-6). The `confidence` attribute is estimated as the average over the input instances' confidence values (lines 8-10). Finally the current time is set as `timestamp` (lines 11-12).

The figure 5.1 illustrates an example of a `list_operator`. Exploiting the strengths of the OMS Pro's rapid prototyping interface, the depicted component can be used also to test the implemented operation by choosing a list of

appropriate instances and executing the method `execute_op`, which returns a new instance.

Listing 5.4: list\_operator implementation example

---

```

1 proc([List],[CtxInstance]) :-
2     findall(V, ( member(I, List), get_attrval_by_name(I,value,
3         V) ), Vals),
4     list_to_set(Vals, Values),
5     length(Vals, Nof),
6     create(CtxInstance, ctx_instance),
7     set_attrval_by_name(CtxInstance, value, Values),
8     findall(C, ( member(I, List), get_attrval_by_name(I,
9         confidence,C), number(C) ), Confidences),
10    sum_list(Confidences, CSum),
11    CAvg is CSum / Nof,
12    set_attrval_by_name(CtxInstance, confidence, CAvg),
13    nowOMS(Now),
14    set_attrval_by_name(CtxInstance, timestamp, Now)
15 .

```

---

## 5.2 Aggregate Context

The aggregate elements, defined by the type `aggr_element`, call the method `execute_op` of the associated operator to get a new context instance as depicted in figure 5.5. Furthermore they use the `add_instance` method defined by their supertype `ctx_element`, which is responsible to associate the new instance with the appropriate subject entities, represented by the `Dictionary` collection. The aggregate elements define a `proc` code field as described in the previous section. This code is responsible to select the convenient context instances, which are passed to the method `execute_op`. Its signature defines that this method does not take any input and returns a list of instances.

Listing 5.5: Implementation of an aggregate\_element

---

```

1 proc(_, [Out]) :-
2     aql('
3         range(
4             map I in
5                 (range (instances dr (all E in c"
6                     ContextElements" having (E=o754))))
7                 by (I.timestamp x I)
8         ), -, Set),
9
10    get_ext(Set, Ext),
11    % order chronologically
12    reverse(Ext, Instances),
13    % take 10 instances
14    getX(Instances, 10, Out)
15 .

```

---

An example selection is given in the listing 5.5; in this case we select the last ten instances of the temperature context identified by the object with `oid=o754`, by means of the AQL query language [NWP<sup>+</sup>03] and the OMS Pro API [OMS04]. The Prolog rule `getX(+List,+X,-First)` in line 14 is defined in the external definition file `context_logic.pl` and it is responsible to get the first `X` elements in a `List`.

Another example of context aggregation implemented within the prototype is the generation of a list of people present in a given room or environment from the single person identity instances provided by a pattern recognition system located within the GlobIS laboratory. This example is presented in listing 5.6.

Listing 5.6: `aggregate_element` selecting the instances of the last minute

---

```

1  proc(-,[Out]) :-
2      % get the users in lab in the last minute
3      nowOMS(Now), Time is Now-60,
4      tokens_to_token(
5          [
6              all C in range(
7                  map I in
8                      (range (instances dr (all E in c"ContextElements"
9                          having (E=o1447))))
10                     by (I.timestamp x I)) having (C.timestamp > ', Time, '
11                 ], Query),
12      aql(Query, -, Set),
13      get_ext(Set,Out)

```

---

In this case, the aggregate element is responsible to select all the instances that identify the users within the GlobIS laboratory provided by the corresponding sensors in the last minute as described by the query in the lines 6-9 in listing 5.6. Finally, the selected instances, which are indicated by the `Out` variable, are passed to the associated operator, which is given as example in listing 5.7.

First, the `list_operator` fetches the value of each instance setting them in a list and eliminates the duplicates by means of the `list_to_set` predicate in line 4. Second, the `confidence` attribute is derived as the average over the `confidence` values of the aggregated instances. Finally, the operator sets the `timestamp`, providing the temporal information for the current context.

Listing 5.7: `list_operator` responsible to produce a list of users

---

```

1  proc([List],[CtxInstance]) :-
2      % compute the aggregate value
3      findall(V, ( member(I, List), get_attrval_by_name(I,value,
4          V) ), Vals),
5      list_to_set(Vals, Values),
6      length(Vals, Nof),
7      % create a new instance
8      create(CtxInstance, ctx_instance),
9      set_attrval_by_name(CtxInstance, value, Values),

```

---

```

9      % set the confidence value
10     findall(C, ( member(I, List), get_attrval_by_name(I,
11               confidence ,C), number(C) ), Confidences),
12     sum_list(Confidences , CSum),
13     CAvg is CSum / Nof,
14     set_attrval_by_name(CtxInstance , confidence , CAvg),
15     % set the timestamp information
16     nowOMS(Now) ,
17     set_attrval_by_name(CtxInstance , timestamp , Now)

```

In the aforementioned example we demonstrate how it is possible to derive new context information from other context. For instance we considered the location information of single users, to produce a more specific context such as the group of users that are present within a given environment. The aggregation of context is achieved by means of two components. The first one is responsible to select the relevant instance to aggregate, while the second one defines the operation responsible to create the new instance, which is structured in three sections: compute the new value based on the input instances, set the appropriate confidence information and provide the corresponding temporal characteristic in terms of a timestamp.

Since the aggregate elements are not directly measured by a context producer, their instances are not referenced in the `providedValues` association, thus they are not bound to any specific producer.

### 5.3 Definition of Events

The context elements (`ctx_element`) implement the `notify` method, which is called whenever a producer provides a new context instance in order to start the context adaptation mechanism as shown in figure 5.5.

When the `notify` method is called, the aggregate elements first update themselves by creating a new instance by means of the method `new_instance` implemented by the type `aggr_element`, then the current context values are checked using the optional `filter_operator`, which defines the conditions under which the corresponding event has to be triggered. Moreover each context element notifies the directly dependent elements, as described by the `basedOn` association in figure 4.7. Consequently, all related contexts are updated by calling their `notify` method.

An example of definition of such a filter is provided in listing 5.8. In this case the filter checks if the users identified by the names *Moira*, *Michael* and *Corsin* are present in the GlobIS laboratory by controlling the last instance of the appropriate context element. The filter simply returns a boolean value indicating whether the conditions are satisfied or not.

Listing 5.8: Implementation of a filter\_operator

```

1  proc ([], [Out]) :-
2     aql('
3  range(

```

```

4     last(
5         map I in
6             (range (instances dr (all E in c"ContextElements"
7                 having (E=o1452))))
8             by (I.timestamp x I))', -, Last),
9     get_attrval_by_name(Last, value, Users),
10    (member('Moirira', Users), member('Michael', Users), member(
11        'Corsin', Users) ->
12        Out=true
13    ;
14    Out=false
15 )
16 .

```

The filter operator takes several contexts as input and returns a boolean flag indicating whether or not a certain event has to be triggered. Actually, a filter may control many contexts of different context types, such as temperature, humidity and location. At the moment, the prototype does not provide any type checking mechanism to ensure the correctness of the implemented operations.

The events are triggered by the associated context element by calling the `fire` method, which is implemented by the objects of type `ctx_event`. The element that causes the event indicates itself as the source responsible for the event, and the corresponding recipients, which are described by the `interestedIn` collection in figure 4.7.

In the implemented prototype, the events are simulated by means of screen information messages as illustrated by figure 5.2. The message includes the name of the event, the name of the source context, the last context instance value and the entities that should be notified. The implementation of the event handler should be adapted in order to integrate the model within an application.

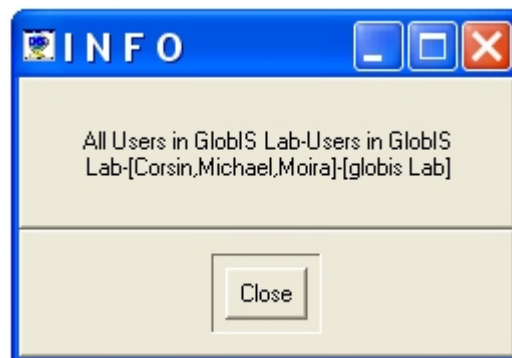


Figure 5.2: Event notification example

## 5.4 Sensor Abstraction and Simulation

Context producers are defined as shown in figure 5.3 by their name and a description. Furthermore, they implement the method `push_value`, which is

responsible to create the corresponding context instance and associate it with the producer by referencing them in the `providedValues` association.

The method `push_value` calls `add_instance` and `notify`, which are provided by the context element's interface in order to associate the new instance with the appropriate objects and notify to the system the incoming changes. This mechanism allows to transform the measured data coming from the sensing devices into veritable information by enhancing its semantic value and setting the appropriate associations. The figure 5.5 illustrates the calling sequence of the interface methods.

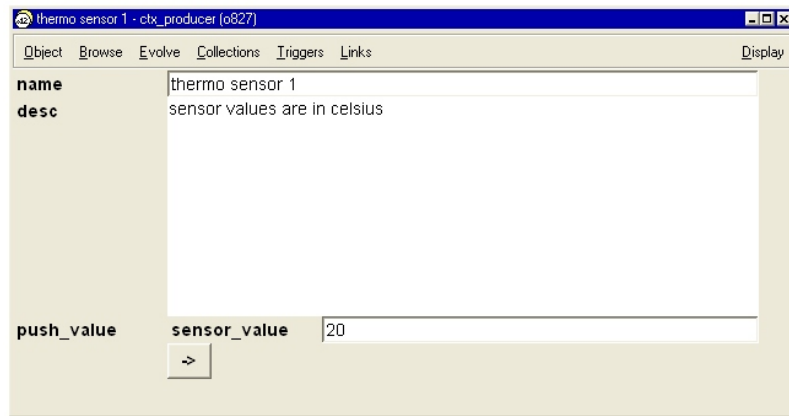


Figure 5.3: Context producer's graphical user interface

The producer may measure the context properties in a different format from the one defined by the context type in our system. For this purpose, it is possible to associate a producer and a given context with an optional context mapper (`ctx_mapper`), which converts the measured value into a format that complies the internal context representation. For instance, it is possible to convert the temperature provided by the producer shown in figure 5.3 from Celsius to Kelvin by means of a mapper defining the functionality described in listing 5.9. The lines 3-11 ensure that the input value is of correct numeric format, and finally the arithmetic operation is performed in line 12.

Especially for testing purposes, a single producer may provide many contexts, i.e. the same sensed data is transformed into different formats, by creating an instance for each provided context.

Listing 5.9: Converting sensor values from Celsius to Kelvin

```

1 proc ([In] , [Out]) :-
2     % check the input value and convert it into 'real'
3     ( atom_codes(In, InS) ->
4         ( catch(number_codes(Celsius, InS), -, fail) ->
5             true
6         )
7         ; sthrow(type_error(celsius2kelvin(In, Out), 1, real, In))
8     )
9     ;

```

```

10         sthrow( type_error( celsius2kelvin( In , Out ) , 1 , real , In )
11     ),
12     Out is Celsius + 273
13 .

```

---

## 5.5 Connecting a Real Sensor

In order to connect an external sensor to the prototype, it is required to start the OMS Pro server as explained in [NWP<sup>+</sup>03] selecting the OPL option. This option allows to feed in data in the database from an external application driving the sensor by calling an appropriate defined macro.

The macro `newSensorValue` represents the socket for the external sensors. It takes a sensor value and the corresponding sensor `oid` as input parameters, thus it calls the method `push_value` of the indicated sensor in order to add the new value into the system. An example of this technique is illustrated by means of a simple Java<sup>4</sup> program in the listing 5.10.

By using the JARToolkit [JAR04], a Java binding to the ARToolKit [ART04], i.e. the Augmented Reality Toolkit, and a simple web cam as sensing hardware, it is possible to implement a pattern recognition system that simulates a person recogniser installed within the GlobIS laboratory. This application provides the user identity according to the recognised pattern. A pattern example is illustrated in figure 5.4.

It is important to notice that the sensory system can only deliver an identifier associated with the recognised pattern. The OMS Pro server connection enables the person recogniser application to feed in the person identity information. Consequently, the gathered information is augmented. The identifier is transformed into an application object that represents the corresponding person, the appropriate associations are set, and the depending contexts are updated. Finally the correspondent events are triggered.

Listing 5.10: Java code to bind an external sensor

---

```

1  import ch.ethz.globis.omspro.server.OMSConfig;
2  import ch.ethz.globis.omspro.server.OMSConnection;
3  import ch.ethz.globis.omspro.server.OMSRequest;

5  public class Test {

7      public static void main(String [] args) {
8          // server properties
9          String host = "localhost";
10         int port = 1071;
11         int timeout = 5000;

13         // sensor information
14         String sensorID = "o1445";
15         int sensorData = 20;

```

---

<sup>4</sup><http://java.sun.com>, February 4, 2004

```
17     try {
18         OMSConfig config =
19             new OMSConfig(false, host, port, timeout);
20         OMSConnection connection =
21             new OMSConnection(config);
22         String message =
23             "newSensorValue("
24                 + sensorData
25                 + ","
26                 + sensorID
27                 + ").";
28         OMSRequest request =
29             new OMSRequest(config, "", "op1", message);
30         connection.send(request);
31     } catch (Exception e) {
32         e.printStackTrace();
33     }
34 }
35 }
```

---



Figure 5.4: ARToolkit pattern example

## 5.6 Quality Information

An important meta-information that is modelled within our solution is the quality of the context producers, which is relevant to the computation of the confidence attribute of each context instance.

The quality of the sensors is described by a set of characteristics embodied by the type `pro_quality`. However, the behavioural profile of a sensor may vary depending on several contextual parameters; consequently computing the confidence value for specific contexts provided by sensors can be very complex. For instance, the quality of a pattern recognition process performed using a video camera may vary depending on the lighting, whereas other physical sensors such as accelerometers and gyroscopes behave in different ways depending on parameters like temperature and humidity.

For the sake of simplicity, the implemented prototype simply assumes the provider's accuracy as the instances' confidence value. Aggregate context elements on the other hand, should define a mechanism to compute the confidence information in the `proc` field as in listing 5.4.

## 5.7 Prototype Overview

The UML sequence diagram proposed in figure 5.5 illustrates how the different components of the implemented prototype collaborate with each other. The diagram is based on the interface explained in section 4.6. It illustrates the operations that are required in order to insert a new value into the prototype system. It explains how to compute an aggregate context and finally it shows how the appropriate events are triggered.

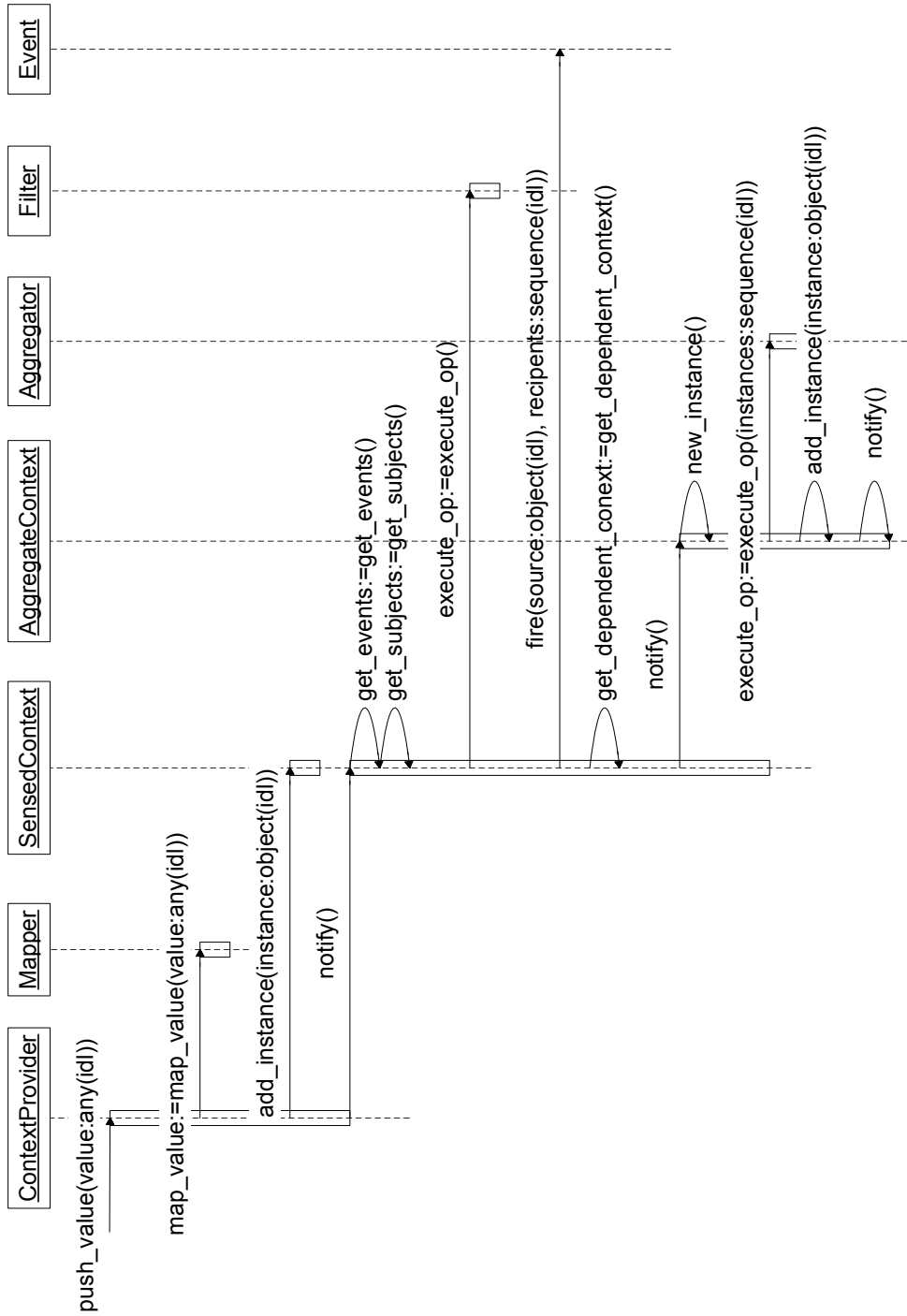


Figure 5.5: UML sequence diagram of the prototype

*There is nothing like a dream to create the future.*

—Victor Hugo

# 6

## Future Work

Our data model for context information may be considered as a starting point in order to integrate context-awareness into more complex systems. In this work, we concentrate in particular on some basic aspects of context information. The implemented prototype has shown its strength and has demonstrated that it is possible to build a comprehensive and general model, which describes the context that characterises the content and allows to define specific control mechanisms without the need to hard-code them in the application.

For instance, the flexibility of the proposed model should allow to deploy it for user-computer interaction by means of dialogues, which can be described for instance using the VoiceXML<sup>1</sup> technology. The computer dialogues can be controlled by means of special events responsible to control the computer messages, whereas the user's answers can be processed by means of appropriate drivers, which are connected to the system through the context producer abstraction.

Each model represents a simplification of reality, hence some characteristics are inevitably not considered within the proposed model. The implemented prototype also shows several perspectives that may be considered and investigated in future works.

At the moment, the proposed prototype system allows to define the control mechanisms by means of objects implementing their functionality in Prolog. A possible extension of the prototype consists in the integration of support also for other programming languages such as OIL [NWP<sup>+</sup>03].

The system could be improved further by designing a specific language that facilitates the definition of control mechanisms such as the context aggregation and the triggering of events. For this purpose, the model could be extended in order to allow to parametrise the control procedures and thus allow to minimise the need for writing application code, extending its modularity and reusability.

An important issue, which is worth investigating in future works concerns

---

<sup>1</sup><http://www.w3.org/Voice/>, February 16, 2004

the information security [BPR01], and privacy in particular. Considering the information delivery on public communication channels and private information, the ability to choose between different security policies is very important. For the SOPHIE project, it will be relevant to decide whether a certain information can be published on a public display or not. The system should know under which condition it is allowed to deliver a piece of information on a specific communication channel. PDAs play an important role for ubiquitous computing, but also our environment could exploit them in order to deliver private information that can't be publicly disclosed. Thus we may also consider to extend our smart information environment in the direction of personal devices, enabling the information delivery also on personal information channels.

Our model provides a technique to describe the quality of the information producers, which is then used to define the confidence attribute of each instance. The quality profile of the context producers depends also on context. This is true in particular for physical sensors, which may produce better measurements under certain conditions, whereas in other situations they provide less reliable data. This behaviour is caused by their physical properties depends on external characteristics such as temperature, humidity and lighting. Therefore, if physical sensor has to be intensively used within the system, it would be worth investigating these relationships, in order to improve the system's reliability.

Our model already allows the subjects to be distributed in several applications and systems. The context information is also suited to be distributed throughout several applications within a system and not managed in a centralised way. Moreover, context is already distributed in nature, in fact each application has its own context. Therefore, an interesting development direction is to investigate how to distribute the context information and how to define the communication between the different applications and systems. One accredited possibility would be to achieve the communication by means of the *web services*<sup>2</sup> technology and the *SOAP*<sup>3</sup> protocol. *Peer-to-peer* systems are another interesting solution, which may be applicable in this field. A distributed solution has the benefit of dividing up the information management between several applications. Consequently, it abridges the complexity of the whole system and increases its performance.

The presented model has shown that it is possible to design a general mechanism to describe and control the context information. Since there are several benefits for a system, which is able to exploit context-awareness, it is worth thinking about how to integrate a similar model into the core system, for instance within the OMS Pro database management system. In order to provide such mechanisms at system level it is important to design a meta-model, which is placed even at a higher abstraction level. We propose in figure 6.1 a meta-model, which can be considered as a starting point in this direction. The meta-model allows to define with high flexibility but also strong formalism the various operations in terms of **sensors** and **notifiers**. The model in figure 6.1 defines the physical sensors as **sensors** that do not have any input parameters,

---

<sup>2</sup><http://www.w3.org/2002/ws/>, February 16, 2004

<sup>3</sup><http://www.w3.org/2000/xml/Group/>, February 16, 2004





*Computers are useless. They can only give you answers.*

—Pablo Picasso

# 7

## Conclusion

By considering the existing research work in the field of context-aware computing, we have realised that several approaches lack a general infrastructure definition, especially in terms of a general data model describing context. The existing abstractions are offered in terms of frameworks facilitating the integration of different sensor systems for the collection of context data. However, they do not define good semantic interpretation of the collected data. Therefore, the data are passed to the application, which is responsible to interpret and process the incoming data.

We believe that a good general data model, which describes context as information about other information, i.e. as meta-information, can be a great improvement on the developing of context-aware applications. The extended use of context in computer applications can provide new interaction mechanisms and facilitate the user's tasks. Moreover, it can produce several benefits, for instance increasing the quality of collaborative work, by providing to the user the relevant information concerning a project team already in the appropriate context.

Furthermore, the computer applications may use the context information in order to gain important knowledge without the direct intervention of the user, enabling new human-computer interaction paradigms and especially opening new interaction dimensions that render the computing process invisible.

In the field of information systems, context can be employed as the driving technology that allows to manage a huge quantity of information and it empowers the system to deliver the right information to the right people at the right moment. The information may be delivered on various communication channels, which are also chosen according to the current situation appropriately.

In this thesis, we have demonstrated the use of a context model, which can be further integrated within existing systems and can provide several control mechanisms for context information. Consequently, the context information

is used to enrich the content, enabling the the whole system to adapt itself to several situations. Moreover, the proposed model allows to select specific information or resources depending on the context information by means of appropriate queries.

# A

## Glossary

**Context Element** It describes the abstract context information such as the temperature or the person identity. The corresponding values are represented by the context instances. A context element may be associated to several values, that represent the same context information at different time, or different subjects.

**Context Instance** The collection of context instances represents the set of context values of a given context element. Each instance belongs to exactly one context element, and is provided by a single source.

**Context Producer** The context producer is an abstraction over the devices that can generate context values. They represent the SOPHIE driver for the various context sensors. They can be both hardware and software devices.

**Context Type** The type information allows to define similar classes of context elements and associate them to similar properties and operations.



# B

## Method Description

In this chapter we describe briefly the functionality provided by the methods of each type in the model and their relationships. The operations for the context model are implemented by means of the Operation Specification Language (OPL) [NWP<sup>+</sup>03].

### B.1 Context Producers

`ctx_producer::push_value(+value)`

Creates a new context instance from the measured sensor data for each associated context element. The sensor data is transformed by means of the optional associated `mapper` into an appropriate format conform to the context type.

Furthermore, it sets the instance's `timestamp` and `confidence` attributes. This method is responsible to relate each created instance to their producer, referencing them in the `producedValues` association. The instances are added by calling the methods `ctx_element::add_instance` and `ctx_element::notify`.

### B.2 Mappers

`ctx_mapper::map_value(+value, -new value)`

This method fetches the code defined in the `proc` field, compiles and executes it. The code is defined in the `proc` field and must take one parameter as input and return one as output. It is called by `ctx_producer::push_value(value)`, in order to transform the sensor values into the adequate representation.

### B.3 Context Elements

`ctx_element::add_instance(+ctx_instance)`

Adds the new `instance` and references it in the `subject` association, according to the registered entities expressed by the association registers.

`ctx_element::get_dependent_ctx(-set of ctx_element)`

Gets the closure of dependent contexts, i.e. both the directly and indirectly dependent contexts.

`ctx_element::get_events(-set of ctx_events)`

Gets all events triggered by the current context element.

`ctx_element::get_instances(-set of ctx_instance)`

Gets all instances of a given context element.

`ctx_element::get_last_instance(-ctx_instance)`

Gets the last instance according to its `timestamp`.

`ctx_element::get_providers(-set of ctx_producer)`

Gets all providers of a given context element.

`ctx_element::get_subjects(-set of app_dictionary)`

Gets the subjects of the context element.

`ctx_element::get_type(-ctx_type)`

Gets the type of the context element.

`ctx_element::notify()`

Checks if some event has to be triggered by means of the associated filter. Besides, it notifies the availability of new instances to the context elements that directly depend on it.

#### Aggregate Context

The `aggr_element` is a subtype of `ctx_element`, hence it provides all the aforementioned functionality. Moreover, it implements the following methods that allow other context instances to be aggregated.

`aggr_context::aggregate(-ctx_instance)`

Executes the code stored in the `proc` field, which selects a list of instances to pass to the `list_operator` that performs the specific operation and returns a new instance. This method does not associate the instance with its subjects.

`aggr_context::get_operator(-list_operator)`

Gets the associated operator of type `list_operator`.

`aggr_context::new_instance()`

First, a new instance is created by calling the `aggregate` method. Finally, it announces the newly created instance by means of the `add_instance` method, which sets all the corresponding associations. This method is called whenever an aggregate element is notified of relevant changes, in order to update the current context.

## B.4 Context Operators

Both the `list_operator` and `filter_operator` types implement the method `execute_op`, which is responsible to fetch, compile and execute the code defined in the `proc` attribute. Their signature and behaviour are different, as explained in the following sections.

### List Operators

`list_operator::execute_op(+set of ctx_instances, -ctx_instance)`

Computes a new instance from a list of instances passed as input. It is responsible also to define the `timestamp` and `confidence` fields. It is called by the method `aggr_context::aggregate`.

### Filter Operators

`filter_operator::execute_op(-boolean)`

It returns a `boolean` indicating whether a given context is satisfied. It is called by the method `ctx_element::notify` in order to decide what events have to be triggered.

## B.5 Context Events

`ctx_event::fire(+ctx_element, +set of app_dictionary)`

This method is called by a context element by means of the `ctx_element::notify` method if the conditions expressed by the associated filter operator are fulfilled. The first parameter describes the context element that calls the method. The second one represents the set of recipients (`app_dictionary`), which are interested in the event that will be triggered as defined by the association `interestedIn`.

At the moment, it shows a pop-up message indicating the event's name, the context responsible for the event, its last instance value and the recipients of the message. The implementation of this method has to be changed accordingly, if the events has to be handled in different ways.

## B.6 Macros

`deleteInstances()`

Deletes all instances from the system. This affects the collection `ContextInstances` and the associations `instances`, `subject` and `providedValues`

`newSensorValue(+value, +ctx_producer oid)`

Inserts a new sensed value into the system by calling the method `push_value` of the appropriate context producer. This macro represents the socket where external sensors can be plugged in. It can be called from external applications when the OMS Pro OPL server is running.

`registerAsSubject(+ctx_element, +app_dictionary)`

Registers the input `app_dictionary` entity as subject of the input `ctx_element`. Consequently, from now on each context instance that is created for the mentioned context element is associated with its registered subject entity.

`registerForEvents(+ctx_element, +ctx_event, +filter_operator, +app_dictionary)`

Registers the input `app_dictionary` entity in the `interestedIn` association for a given `event`. The conditions under which the `event` has to be triggered are expressed by the `filter_operator`. It also sets the appropriate references in the associations `triggers` and `filterOp`.

# Bibliography

- [ART04] ARToolKit. [http://www.hitl.washington.edu/projects/shared\\_space/](http://www.hitl.washington.edu/projects/shared_space/), January 29, 2004
- [AS96] AKMAN, V., AND SURAV, M. Steps toward formalizing context. *AI Magazine*, 17(3):55–72, 1996
- [BBG01] BENERECETTI, M., BOUQUET, P., AND GHIDINI, C. On the dimensions of context dependence: Partiality, approximation, and perspective. In *CONTEXT 2001, LNAI 2116*, pages 59–72. Springer-Verlag, 2001
- [BD03] BARKHUUS, L., AND DEY, A. K. Is context-aware computing taking control away from the user? Three levels of interactivity examined. In *UbiComp 03 conference*, Seattle, October 2003
- [BL01] BAUER, T., AND LEAKE, D. B. WordSieve: A method for real-time context extraction. *Lecture Notes in Computer Science*, 2116:30–44, 2001
- [BLHL01] BERNERS-LEE, T., HENDLER, J., AND LASSILA, O. The semantic web. *Scientific American*, May 2001
- [BPR01] BURNETT, M., PREKOP, P., AND RAINSFORD, C. P. Intimate location modeling for context aware computing. In *Proceedings of the Workshop on Location Modeling for Ubiquitous Computing*, Atlanta, Georgia, 2001
- [CDM03] CERI, S., DANIEL, F., AND MATERA, M. 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
- [CDS04] CHALMERS, D., DULAY, N., AND SLOMAN, M. Meta data to support context aware mobile applications. In *IEEE Intl. Conference on Mobile Data Management (MDM2004)*. IEEE, January 2004
- [CFJ03] CHEN, H., FININ, T., AND JOSHI, A. An ontology for context-aware pervasive computing environments. In *IJCAI workshop on ontologies and distributed systems, IJCAI'03*, Acapulco, MX, August 2003

- [CFJ04] CHEN, H., FININ, T., AND JOSHI, A. A context broker for building smart meeting rooms. In *AAAI 2004 Spring Symposium on Knowledge Representation and Ontology for Autonomous Systems*, Stanford, 2004. Draft
- [Cha02] CHALMER, D. *Contextual Mediation to Support Ubiquitous Computing*. PhD thesis, Faculty of Engineering of the University of London, August 2002
- [CK00] CHEN, G., AND KOTZ, D. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000
- [CMR90] CARD, S. K., MACKINLAY, J. D., AND ROBERTSON, G. G. The design space of input devices. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people*, pages 117–124, Seattle, Washington, United States, 1990
- [DA00] DEY, A. K., AND ABOWD, G. D. 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, April 2000
- [Dey00] DEY, A. K. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, November 2000
- [Dey01] DEY, A. K. Understanding and using context. *Personal and Ubiquitous Computing Journal*, 5(1):4–7, 2001
- [DMAC02] DEY, A. K., MANKOFF, J., ABOWD, G. D., AND CARTER, S. Distributed mediation of ambiguous context in aware environments. In *Proceedings of the 15th Annual Symposium on User Interface Software and Technology (UIST 2002)*, pages 121–130, Paris, France, October 2002
- [DSFA99] DEY, A. K., SALBER, D., FUTAKAWA, M., AND ABOWD, G. D. An architecture to support context-aware applications. Technical report, Georgia Institute of Technology, June 1999
- [EBDN03] EDWARDS, K., BELLOTTI, V., DEY, A. K., AND NEWMAN, M. Stuck in the middle: The challenges of user-centered design and evaluation for middleware. In *Proceedings of the 2003 Conference on Human Factors in Computing Systems (CHI 2003)*, Fort Lauderdale, FL, April 2003
- [GBSV03] GOSLAR, K., BUCHHOLZ, S., SCHILL, A., AND VOGLER, H. A multidimensional approach to context-awareness. In *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI2003)*, 2003

- [GM03] GUHA, R., AND MCCARTHY, J. Varieties of contexts. In *4th International and Interdisciplinary Conference, CONTEXT 2003, LNAI 2680*, pages 164–177. Springer-Verlag, January 2003
- [Gru01] GRUDIN, J. Desituating action: Digital representation of context. *Human-Computer Interaction*, 16:269–286, 2001
- [GS01] GRAY, P., AND SALBER, D. Modelling and using sensed context information in the design of interactive applications. In *Engineering for Human-Computer Interaction: 8th IFIP International Conference, EHCI 2001, (LNCS)*, volume 2254, 2001
- [HHS<sup>+</sup>99] HARTER, A., HOPPER, A., STEGGLES, P., WARD, A., AND WEBSTER, P. The anatomy of a context-aware application. In *Mobile Computing and Networking*, pages 59–68, 1999
- [HIR02] HENRICKSEN, K., INDULSKA, J., AND RAKOTONIRAINY, A. Modeling context information in pervasive computing systems. In *Proceedings Pervasive 02*, Zurich, August 2002. Springer Verlag
- [HJ03] HAWICK, K., AND JAMES, H. Middleware for context sensitive mobile applications. In JOHNSON, C., MONTAGUE, P., AND STEKETEE, C., editors, *Workshop on Wearable, Invisible, Context-Aware, Ambient, Pervasive and Ubiquitous Computing*, volume 21 of *Conferences in Research and Practice in Information Technology*, pages 133–141, Adelaide, Australia, 2003. ACS
- [HL01a] HONG, J. I., AND LANDAY, J. A. A context/communication information agent. In *Personal and Ubiquitous Computing: Special Issue on Situated Interaction and Context-Aware Computing*, 2001
- [HL01b] HONG, J. I., AND LANDAY, J. A. An infrastructure approach to context-aware computing. *Human-Computer Interaction*, 16, 2001
- [JAR04] JARToolkit. <http://www.c-lab.de/jartoolkit/>, January 29, 2004
- [KM03] KORPIPÄÄ, P., AND MÄNTYJÄRVI, J. An ontology for mobile device sensor-based context awareness. In *CONTEXT 2003, LNAI*, volume 2608, pages 451–458, Berlin, 2003. Springer-Verlag
- [KOA<sup>+</sup>99] KIDD, C. D., ORR, R. J., ABOWD, G. D., ATKESON, C. G., ESSA, I. A., MACINTYRE, B., MYNATT, E., STARNER, T. E., AND NEWSTETTER, W. The Aware Home: A living laboratory for ubiquitous computing research. In *Proceedings of the Second International Workshop on Cooperative Buildings - CoBuild'99*, October 1999
- [Kok95] KOKINOV, B. A dynamic approach to context modeling. In *Proceedings of the IJCAI-95 Workshop on Modeling Context in Knowledge Representation and Reasoning*, 1995

- [MJ03] MANTORO, T., AND JOHNSON, C. Location history in a low-cost context awareness environment. In JOHNSON, C., MONTAGUE, P., AND STEKETEE, C., editors, *Workshop on Wearable, Invisible, Context-Aware, Ambient, Pervasive and Ubiquitous Computing*, volume 21 of *Conferences in Research and Practice in Information Technology*, pages 153–158, Adelaide, Australia, 2003. ACS
- [MPN01] MOTSCHNIG-PITRIK, R., AND NYKL, L. The role and modeling of context in a cognitive model of rogers' person-centred approach. In *CONTEXT 2001, LNAI 2116*, pages 275–289. Springer-Verlag, 2001
- [MR03] MEYER, S., AND RAKOTONIRAINY, A. A survey of research on context-aware homes. In JOHNSON, C., MONTAGUE, P., AND STEKETEE, C., editors, *Workshop on Wearable, Invisible, Context-Aware, Ambient, Pervasive and Ubiquitous Computing*, volume 21 of *Conferences in Research and Practice in Information Technology*, pages 159–168, Adelaide, Australia, 2003. ACS
- [NP03a] NORRIE, M. C., AND PALINGINIS, A. Empowering databases for context-dependent information delivery. In *Ubiquitous Mobile Information and Collaboration Systems (UMICS 2003)*, Klagenfurt/Velden, Austria, June 2003
- [NP03b] NORRIE, M. C., AND PALINGINIS, A. Versions for context dependent information services. In *Conference on Cooperative Information Systems (COOPIS 2003)*, Catania-Sicily, Italy, November 2003
- [NW97] NORRIE, M. C., AND WÜRGLER, A. OM framework for object-oriented data management. *INFORMATIK Journal of the Swiss Informaticians Society*, June 1997
- [NWP<sup>+</sup>03] NORRIE, M. C., WÜRGLER, A., PALINGINIS, A., VON GUNTEN, K., AND GROSSNIKLAUS, M. *OMS Pro 2.0 Introductory Tutorial*. Institute for Information Systems, ETH Zürich, March 2003
- [NWT<sup>+</sup>02] NIXON, P., WANG, F., TERZIS, S., WALSH, T., AND DOBSON, S. Engineering context-aware enterprise systems. In *Workshop on Engineering Context-Aware Object-Oriented Systems and Environments (ECOOSE)*, 2002
- [OMS04] OMS Pro. <http://www.oms.ethz.ch/omspro/>, February 6, 2004
- [Pas98] PASCOE, J. Adding generic contextual capabilities to wearable computers. In *ISWC*, pages 92–99, 1998
- [Pas01] PASCOE, J. *Context-Aware Software*. PhD thesis, Computing Laboratory, University of Kent at Canterbury, August 2001

- [PNZ<sup>+</sup>01] PETRELLI, D., NOT, E., ZANCANARO, M., STRAPPARAVA, C., AND STOCK, O. Modelling and adapting to context. In *Personal and Ubiquitous Computing*, volume 5(1), pages 20–24, London, UK, February 2001. Springer-Verlag
- [PRM99] PASCOE, J., RYAN, N. S., AND MORSE, D. R. Issues in developing context-aware computing. In H-W.GELLERSEN, editor, *Handheld and Ubiquitous Computing*, number 1707 in Lecture Notes in Computer Science, pages 208–221, Heidelberg, Germany, September 1999. Springer-Verlag
- [RCG<sup>+</sup>03] RIOS, D., COSTA, P. D., GUIZZARDI, G., PIRES, L. F., FILHO, J. G. P., AND VAN SINDEREN, M. Using ontologies for modeling context-aware services platforms. In *OOPSLA2003*, 2003
- [RJB99] RUMBAUGH, J., JACOBSON, I., AND BOOCH, G. *The Unified Modeling Language Reference Manual*. Addison Wesley Longman, Inc., 1999
- [SAT<sup>+</sup>99] SCHMIDT, A., AIDOO, K. A., TAKALUOMA, A., TUOMELA, U., LAERHOVEN, K. V., AND DE VELDE, W. V. Advanced interaction in context. *Lecture Notes in Computer Science*, 1707:89–??, 1999
- [SAW94] SCHLIT, B., ADAMS, N., AND WANT, R. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, 1994
- [SB00] SCHIRMER, J., AND BACH, H. Context management in an agent-based approach for service assistance in the domain of consumer electronics, 2000
- [SBG99] SCHMIDT, A., BEIGL, M., AND GELLERSEN, H.-W. There is more to context than location. *Computers and Graphics*, 23(6):893–901, 1999
- [Sch00] SCHMIDT, A. Implicit human computer interaction through context. In *Personal Technologies*, volume 4(2&3), pages 191–199, June 2000
- [SDA99] SALBER, D., DEY, A. K., AND ABOWD, G. D. 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
- [SG02] STAVRAKAS, Y., AND GERGATSOULIS, M. Multidimensional semi-structured data: Representing context-dependent information on the web. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE 2002)*, Toronto, Canada, May 2002

- [SGM00] STAVRAKAS, Y., GERGATSOULIS, M., AND MITAKOS, T. Representing context-dependent information using multidimensional XML. *Lecture Notes in Computer Science*, 1923:368–371, 2000
- [TACS98] THEODORAKIS, M., ANALYTI, A., CONSTANTOPOULOS, P., AND SPYRATOS, N. Context in information bases. In *Conference on Cooperative Information Systems*, pages 260–270, 1998
- [TACS02] THEODORAKIS, M., ANALYTI, A., CONSTANTOPOULOS, P., AND SPYRATOS, N. A theory of contexts in information bases. *Information Systems*, 27(3):151–191, 2002
- [TGF03] TAZARI, M.-R., GRIMM, M., AND FINKE, M. Modelling user context. In *The 10th International Conference on Human-Computer Interaction (HCI2003)*, Crete (Greece), July 2003. Lawrence Erlbaum Associates
- [Tur93] TURNER, R. M. Context-sensitive reasoning for autonomous agents and cooperative distributed problem solving. In *Proceedings of the 1993 IJCAI Workshop on Using Knowledge in Its Context*, Chambry, France, 1993
- [Wei91] WEISER, M. The computer for the twenty-first century. *Scientific American*, pages 94–100, September 1991
- [WHFG92] WANT, R., HOPPER, A., FALCÃO, V., AND GIBBONS, J. The active badge location system. Technical Report 92.1, Olivetti Research Ltd. (ORL), 24a Trumpington Street, Cambridge CB2 1QA, 1992
- [Wid03] WIDDOWS, D. A mathematical model for context and word-meaning. In *4th International and Interdisciplinary Conference, CONTEXT 2003, LNAI 2680*, pages 369–382. Springer-Verlag, January 2003
- [Win01] WINOGRAD, T. Architectures for context. *Human-Computer Interaction*, 16, 2001
- [WS01] WALCHE, H., AND STUCKENSCHMIDT, H. Practical context transformation for information system interoperability. In *CONTEXT 2001, LNAI 2116*, pages 367–380. Springer-Verlag, 2001
- [Zel97] ZELLER, A. *Configuration Management with Version Sets - A Unified Software Versioning Model and its Applications*. PhD thesis, TU Braunschweig, April 1997

# List of Figures

2.1	Several dimensions of context . . . . .	8
3.1	Context architecture . . . . .	12
3.2	Context-aware information system . . . . .	13
4.1	An application specific context data model . . . . .	16
4.2	Core model of context . . . . .	17
4.3	Context acquisition model . . . . .	19
4.4	Context aggregation model . . . . .	20
4.5	Event notification model . . . . .	21
4.6	UML static diagram of the the model types . . . . .	22
4.7	Overview of the full model . . . . .	23
5.1	Instance of list operator . . . . .	27
5.2	Event notification example . . . . .	31
5.3	Context producer's graphical user interface . . . . .	32
5.4	ARToolKit pattern example . . . . .	34
5.5	UML sequence diagram of the prototype . . . . .	36
6.1	Meta-model overview . . . . .	39



# Listings

5.1	Intermediate Representation Check Hook . . . . .	26
5.2	Trigger on update of the type list_operator . . . . .	26
5.3	execute_op code of the list_operator type . . . . .	27
5.4	list_operator implementation example . . . . .	28
5.5	Implementation of an aggregate_element . . . . .	28
5.6	aggregate_element selecting the instances of the last minute . . .	29
5.7	list_operator responsible to produce a list of users . . . . .	29
5.8	Implementation of a filter_operator . . . . .	30
5.9	Converting sensor values from Celsius to Kelvin . . . . .	32
5.10	Java code to bind an external sensor . . . . .	33



# Index

- Action, 21
- Active Badge, 7
- Adaptation, 12, 30
- Aggregate Operator, 29
- Artificial Intelligence, 2
- ARToolKit, 33
  
- Class Methods, 25
- Computing Context, 7
- Context, 5
  - aggregate, 19, 28
  - category of, 7
  - definition of, 6
  - element, 17, 43
  - instance, 18, 43
  - producer, 11, 18, 43
  - type, 17, 43
- Context Toolkit, 9, 11
- Context-Aware Computing, 1, 15
- Context-Awareness
  - active, 21
  - passive, 20
  
- Enterprise Application, 8
- Event, 21, 30
  
- Filter Operator, 31
- Framework, 9, 16, 41
  
- Hard Context-Awareness, 8
- Human-Computer Interaction, 2
  
- Instance Method, 25
  
- Linguistics, 2
- Location, 7
  
- Mobile Phones, 2
- Museum tour guide, 15
  
- Ontology, 17
  
- Person Recogniser, 33
- Pervasive Computing, 2, 8
- Philosophy, 2
- Physical Context, 7
- Polysemous words, 1
- Privacy, 13
- Public information display, 10
  
- Quality, 18, 34
  
- Resource Discovery, 12
  
- Security, 13
- Sensor, 11, 33
- Smart Environment, 8
- Soft Context-Awareness, 8
- SOPHIE, 9, 13, 15, 38, 43
- Speech Recognition, 19
- Subjective context, 8
  
- Taxonomy, 7
  
- Ubiquitous Computing, 2, 7, 15
- User Context, 7