Context-Awareness on Mobile Devices - the Hydrogen Approach

Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann
{firstname.surname}@scch.at
tel: +43 (7236) 3343 - 836
Software Competence Center Hagenberg
Hauptstraße 99, A-4232 Hagenberg

Werner Retschitzegger
werner@ifs.uni-linz.ac.at
tel: +43 (732) 2468 - 8883
Department of Information Systems
Johannes Kepler University Linz
Altenbergerstr. 69, A-4040 Linz

Abstract

Information about the user's environment offers new opportunities and exposes new challenges in terms of time-aware, location-aware, device-aware and personalized applications. Such applications constantly need to monitor the environment – called context - to allow the application to react accordingly to this context. Context-awareness is especially interesting in mobile scenarios where the context of the application is highly dynamic and allows the application to deal with the constraints of mobile devices in terms of presentation and interaction abilities and communication restrictions.

Current context-aware applications often realize sensing of context information in an ad hoc manner. The application programmer needs to deal with the supply of the context information including the sensing of the environment, its interpretation and its disposal for further processing in addition to the primary purpose of the application. The close interweavement of device specific context handling with the application obstructs its reuse with other hardware configurations.

Recently, architectures providing support for context-aware applications have been developed. Up to now such architectures are not trimmed to the special requirements of mobile devices regarding particularly the limitations of network connections, limited computing power and the characteristics of mobile users.

This paper proposes an architecture and a software framework - the Hydrogen Context-Framework - which support context-awareness for considering these constraints. It is extensible to consider all kind of context information and comprises a layered architecture.

To prove the feasibility the framework has been implemented to run on mobile devices. A context-aware postbox is realized to demonstrate the capabilities of the framework.

1 Introduction

Ubiquitous computing was first stressed by Marc Weiser [31], envisioning a scenario where computational power would be available everywhere embedded in walls, chairs, clothing etc. Weiser's goal is to achieve the most effective kind of technology, which is available throughout the physical environment, while making them effectively invisible to the user. There is still a long way to go, but miniaturized devices like PDAs, micro computers and embedded devices offer already existing technology as a first step towards ubiquity of applications.

In general, ubiquity enables new opportunities and challenges for applications in terms of time-aware [18], location-aware [12], device-aware [23] and personalized applications [19]. This implies that ubiquitous applications need to be context-aware, thus taking into account - individually for each user - time and location of access, together with the different capabilities of devices comprising display resolution, memory, computational capabilities as well as network capacity. Consequently, the fundamental objective of context-awareness is to provide services not only to people at any time, any where, with any media but specifically to communicate the right thing at the right time in the right way. The pre-requisite for this is that the application is aware of its context [1], [2], [31]. Therefore the application has to have a representation of the information about its environment (i.e. context) and react accordingly. From our point of view, context is all relevant information about an application's environment.

From a software-engineering point of view the application should not be intermingled with the management and storage of its context. To let an application concentrate on its very own purpose, an important architectural principle is to decouple mechanism for sensing context information and its interpretation from the provision of context-information to the application. Finally the context representation should be generic and thus extensible to cope with future needs.

1.1 Context Characteristics

Context can be used in two ways. On the one hand applications can customize themselves according to the context for better usage. Imagine a calendar reminder, which customizes itself according to location information, e.g. when the position of a device indicates a meeting room, a vibrating alarm is used instead of an alarm bell. On the other hand context information can be used for creating a new type of applications, e.g. a location aware reminder tool.

Context can be separated into physical context representing the level of environment sensors and logical context representing more abstract information about the environment. Physical context properties are at a very low level of abstraction and are continuously updated to take into account the fact that the environment and the application state itself continuously changes.

Logical context information is needed to enrich the semantics of physical context information (e.g., GPS-coordinates), thus making it meaningful for high-level purposes (e.g., street name). [16].

1.2 Context-Awareness for Mobile Devices

In the following issues different to non-mobile scenarios are outlined.

Context-awareness is especially interesting for mobile devices where the context of the application is highly dynamic allowing the application to deal with the constraints of mobile devices in terms of presentation and interaction abilities and communication restrictions.

At the same time, mobile devices such as PDAs suffer from a lack of system resources like computing power, memory and power supply.

This implies special restrictions on the architecture of a framework supporting context-awareness on mobile devices, and of context-aware applications in particular.

Furthermore a user, who is moving (with his/her mobile device) is not permanently connected to a network. In case that Wireless LAN, Bluetooth or other wireless connections are used, a user will get out of range of access points, switches to other access points or simply just moves out of network coverage. At the moment permanent connections are not guaranteed, an application cannot rely on remote servers. This discontinuity of network connections has to be taken into account when designing an framework architecture for mobile devices.

But the network connection when using mobile devices is not the only difference to a traditional use of applications. Mobile devices are much more personal meaning that a user of a mobile device normally does not change. Furthermore personal devices often move with their users.

Regarding the need to save energy a mobile device can be turned off or simply some extensions such as network cards can be disabled when they are not absolutely required. A deactivated sensor cannot sense any information about the context and therefore the current context cannot be determined until the sensor is turned on again.

On the basis of these special characteristics of a mobile scenario we identified some requirements for an architecture of a framework to support context-awareness on mobile devices

- Lightweightness: The architecture has to take into account the restrictions of limited processing power.
- Extensibility: Since available sensors and extension slots are limited, it is not possible for a single device to sense all context information. Therefore the architecture should support connections to remote sensors.
- Robustness: The architecture has to be robust against disconnections of remote sensors.
- Meta-Information: Due to the fact that remote sensors could provide controversial information, the context model has to contain meta-information about the distance of the device to the sensor, its preciseness and many more.
- Context-Sharing: In addition to the above-mentioned requirements we notice that when using mobile devices the sensed context is almost never complete. This incompleteness rests upon limited capabilities as well as the purpose to be mobile, which means not to attach any available hardware. We claim a mechanism to share the sensed context with other devices.

Therefore an architecture separating the concerns of context sensing from the application is needed, a software framework realizing this architecture would help to simplify work for an application programmer and would let an application concentrate on its application purpose.

The rest of this paper structures as follows: Section 2 reflects the state of the art in developing context-aware applications for mobile devices. For overcoming the shortcomings of existing approaches the *Hydrogen Context-Framework* is proposed in Section 3. Implementation aspects are outlined in Section 4. Finally, open issues and future work regarding context-awareness in mobile scenarios are discussed.

2 Related Work

Current context-aware applications often realize the sensing of context information in an ad hoc manner. Application programmers need to deal with the supply of the context information including the sensing of the environment, its interpretation and its disposal for further processing in addition to the primary purpose of the application. The close interweavement of the device specific context handling with the application obstructs its reuse with other hardware configurations (e.g. other sensors, other devices).

Up to now such architectures are not trimmed to the special requirements of mobile scenarios regarding particularly the limitations of network connections, power supply, CPU-speed, and memory.

Recently some architectures were proposed and frameworks were developed, which provide support for context-aware applications in general and mobile scenarios in particular.

The *ContextToolkit* [6], [7], [8], [9] provides a basic architecture for platform-independent supply of context information to the application. One of the requirements, which heavily influences the design of the *ContextToolkit*, is that "context can come from many, distributed machines" [5]. So-called widgets read sensors and deliver it to a server that is responsible for aggregating the context. Based upon the architecture a framework was developed.

Using the *ContextToolkit* the application can subscribe to a widget remotely but needs to re-subscribe after a disconnection [9]. Dynamic detection of remote sensors is not supported, applications must know both, the hostname and the port the component is being executed on [5]. Applications using this framework are fully depending on (remote) servers and widgets. Whilst there is no route to the server, an application cannot access any context, not even context that has been sensed before.

GeoNotes by Espinoza et al. is a system for abstracting location information for location-aware applications [10]. The system architecture is constructed to support shared information for mobile devices, exactly to leave notes at specific places for other users to be read. A user creates notes and sticks it to certain places, where other users can read them. Notes can be targeted at a single user or a whole user group. Vice versa a specific user can create and apply filters to perceive only a subset of the notes associated to a certain place.

GeoNotes considers context very limited in that they consider location context only. *GeoNotes* makes a central server necessary for the storage of the notes and the transmission to users reaching the associated places, because creator and receiver of a note do not have any direct connection. They are not at the same place at the same time.

Ferscha et al. [11] present a *multi-user team awareness framework* called *CampusSpace* for gathering the geographical position using WLAN access points. The idea is based on using signal strength and signal quality to determine spatial proximity either to an access point or another device also equipped with WLAN.

Nord et al. present an architecture for location aware applications [22]. The architecture combines the diffent ways of reading the location information with peer-to-peer position sharing.

Couderc et al. [4] propose an architecture to improve the level of quality of a service for mobile users using context-awareness. To handle changing context like bandwidth or available services the proposed general architecture bases on contextual objects to design and develop adaptive distributed information systems. These contextual objects are objects, which can adapt themselves according to a certain context. A first part of this architecture has been already implemented realizing a location aware web service.

The framework by Couderc et al. reads context information from a client and delivers it to a (remote) server, responsible for certain adaptations before responding to the client. The content management and the business logic are situated on this remote server. This approach fits for client/server architectures like web-applications, but does not support stand-alone applications or disconnected devices.

In general architectures, which require vast memory or system resources, are not feasible for mobile devices. This has to be regarded already while defining the architecture of a framework. Features like the storage of vast historical context information or the assumption of future context should be handled with care.

3 Hydrogen Context-Framework

In the following we propose a three-layered architecture for a context framework called *Hydrogen Context-Framework* in order to overcome the shortcomings of existing approaches in dealing with mobile scenarios. It is trimmed to the special needs of mobile devices as and thus taking care of the requirements discussed in Section 1.2

3.1 Framework Architecture

As shown in Figure 1 the proposed architecture comprises three layers to separate the concerns of interacting with the physical sensors, storing and maintaining the context from the applications itself. These layers are the *Adaptor Layer*, the *Management Layer*, and the *Application Layer*.

The core component is a storage for the context, which can be queried by applications, called *ContextServer*. Context is stored once for the whole device and provided to all applications.

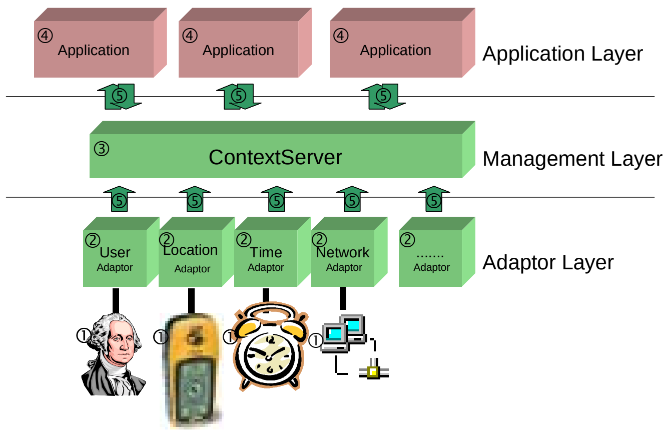

Figure 1: Overview of the Architecture

The *Adaptor Layer* ② is responsible to get information from sensors ① about the physical context [16], possibly enriches this information with logical context information [16] and delivers it to the Management Layer. Additionally the *ContextServer* ③ embedded in the *Management Layer* provides simple methods for the application for retrieving or subscribing to a context.

In addition to issues concerning the aimed devices application development is simplified and reusability and exchangeability of sensors and adaptors increases.

3.1.1 Adaptor Layer. Another important reason for separating context sensing, storing and the use in an application is the simultaneous access of two or more applications to one context. This often would not be possible when accessing the sensors directly because of exclusive locking of system resources, e.g. a serial interface can only be accessed from one application at the same time. Using this special layer means accessing sensors only once and delivering the gained data to the management layer, which solves the problem of multiple applications reading data from the same sensor(s).

3.1.2 Management Layer. The context server stores all contextual information about the current environment of the device. Furthermore the context server has the possibility to share its information with other devices in range. This enables devices to use information, which they cannot retrieve themselves by working with values sensed by others. As we will outline in Section 5.3 this context sharing is an open issue including the problems, such as a hop-dependent degree of exactness (e.g. for location information), which are posed by sharing context information.

The *Management Layer* is responsible for providing and retrieving these contexts and shares this information with other devices using peer-to-peer communication.

The *ContextServer* offers two ways of usage for applications referring to the context.

The first one is asynchronous and simply allows querying a specific context from the context server in a pull-based manner.

Due to the nature of context-aware applications, which often react to changes of the context during their execution, a second way is provided. A subscription-based push mechanism provides synchronous access to the context. Using this possibility the application is informed about changes or the invalidation of the subscribed context.

3.1.3 Application Layer. Applications, which use the context provided by the underneath layers, are part of the *Application Layer* ④.

3.1.4 Context Architecture. In addition to this architecture we propose an architecture for the context. Basically we distinguish between local and remote context. Because devices in range can communicate with each other, a device can store additional information about those other devices. This context of the connected devices can be used to approximate values for the own context, e.g. for location or temperature, or to provide additional information about remote, connected devices for applications, e.g. which users are in range.

The main benefit of this architecture in comparison to other approaches is that the three layers are located on one device thus making the approach robust against network disconnections

The three layered architecture makes it possible, that all applications have access to all context data by querying the *ContextServer*. Without this partition each application would have to read the sensors directly.

3.2 Benefits of the Framework Architecture

As already pointed out in Section 1.2 mobile scenarios including limited devices like PDAs influence the requirements for the Hydrogen Context-Framework.

Due to the fact that applications do not deal with remote servers but only with a local server, which provides any kind of available contextual information, the architecture is robust with respect to frequent disconnections.

For these reasons the Hydrogen Context-Framework abandons some concepts of other approaches like storing a vast amount of history of the context to meet the requirement of lightweightness identified in Section 1.2.

Context Sharing is supported on basis of a peer-to-peer connection without the need of a centralized server to serve dozens of clients.

4 Implementation Aspects

To prove the feasibility of the architecture presented in Section 3, a prototypical implementation has been developed using the PersonalJava [25] virtual machines Jeode [13] and the J2ME [26] J9 [14] on iPAQs (3660 and 3870) [3] with the PocketPC 2002 operating system.

Due to the fact that Java virtual machines for mobile devices are at an early stage of development we had to overcome some technical challenges caused by insufficient documentation and unforeseeable behaviour of our beta-releases.

To simplify the work of an application programmer the framework provides a set of predefined context classes and a so-called ContextClient [cf. ④ in Figure 2], which is responsible for communication with the Context Server.

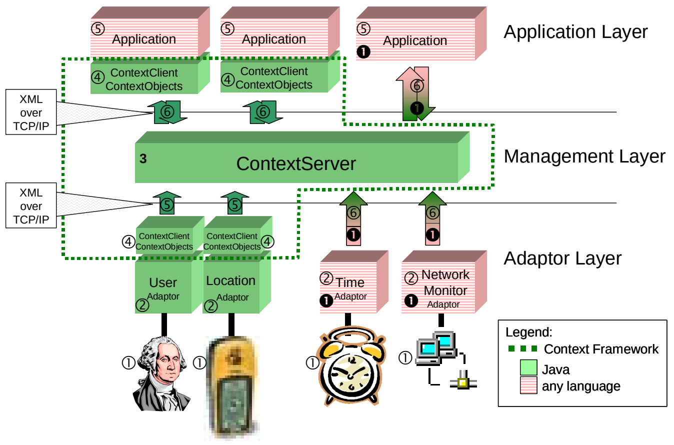

Figure 2: Framework Implementing Architecture

Inter-layer communication (cf. ⑥ and ⑥❶) is based on an XML-protocol and can be performed either by using the *ContextClient* ④ or directly (cf. ⑥❶) by using the XML-protocol. But note that in the second case all communication issues like opening a port, sending data and listening on ports to receive answers must be handled by the application or adaptor itself, while the *ContextClient* provides this functionality.

4.1 Context

As already mentioned, applications cannot only consider context of the executing device, but also should be able to consider the context of devices it is communicating with.

We propose an architecture, which not only contains the context of the location device, but is open to represent the context of remote devices, too.

In case of encountering another device, context information can be mutually exchanged enabling a local representation of the remote device's context. The context, which is based on this architecture, is depicted in an UML class-diagram in Figure 3.

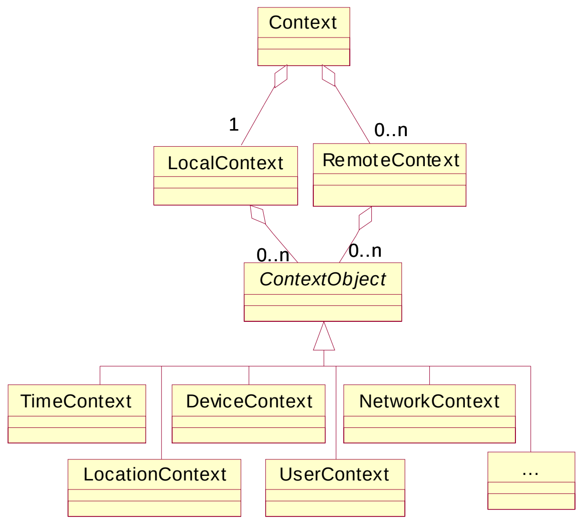

Figure 3: Context Architecture

The *local context* contains several *ContextObjects*, holding context information as provided by any attached sensor. Additionally any device can store *remote contexts* of devices, which are accessible over the network.

The Context-Framework comprises five types of context, but is extensible by specializing the *ContextObject*. The implemented contexts in the current version are in specific:

- Time: Time comprises the current time, as provided by the system clock of the used device
- Location: Location represents the current (physical) position of the device using GPS-coordinates and a value for the altitude. This context is typically set by a adaptor, which reads a GPS-receiver.
- Device: Device consists of an identifier, which should be unique, and a device type, which can be used to distinguish between different types of devices such as Desktop PCs, Laptops, and PDAs.
- User: User contains information about the current user of the device, currently holding only the name of the user.
- Network: Network context contains information about the available network connection types of the device. This context can be used for additional information about the likeliness of an abort of the connections or the available bandwidth.

More specialized types of context can be added to the framework by specializing the *ContextObject* class, which is the base for all context objects. It provides a standardized interface and a possibility to convert it into XML.

4.2 Context Server

The *ContextServer* itself is a Java-executable object, which stores all information about the current context. It is accessible through a user-defined port and waits for commands by adaptors and applications. Adaptors supply information to the *ContextServer*, which can be queried or subscribed by applications.

Subscribed applications can specify the interval for pushing updates in order to keep communication overhead and system resource usage low.

This communication is possible in two forms, using either XML-streams [⑥❶ in Figure 2], which ensures compatibility and interoperability, or serialized Java objects [⑥ in Figure 2].

4.3 Adaptors and Applications

As already illustrated in Figure 2 the Context-Framework comprises mechanisms for handling the communication between adaptors and applications. When developing applications in Java these tools can be used, otherwise pure XML communication can be used by non-Java applications.

The *ContextClient* does all the communication issues, opens ports, queries data from and sends data to the *ContextServer*. It provides methods for querying a context once, or subscribing to a context and waiting for updates provided by the *ContextServer*. This class can be simply used by any *Adaptor* or *Application*, as depicted in Figure 4.

Figure 4 illustrates the use of these classes and characterizes a typical usage scenario. The shown code fragment is part of a context-aware application and enables this application to react to the change of a location by calling the method `doSomething()`.

The call of the method `doSomething()` is linked to an event; by subscribing a listener to it the application expresses its interest to receive updates.

```
LocationContext lc;
lc = new LocationContext();①
lc.addContextChangedListener(new
  ContextChangedListener()②
  {
    public void
      contextChanged(ContextChangedEvent e)
      {
        doSomething();    ③
      }
   });
ContextClient cc;
cc = new ContextClient();④
cc.subscribeTo(lc); ⑤
```

Figure 4: Code Fragment

In our example the application subscribes a listener to changes of the context (c.f. `ContextObject`) utilizing the Java event subscription mechanism. If the change of the context is detected the appropriate adaptation is triggered (c.f. `doSomething()`).

This context subscribed to the ContextServer using the ContextClient (cf. Class `ContextClient`), which is part of the framework and simplifies communication with the ContextServer. Otherwise opening a socket to the server and sending the XML-formed command directly to it could also do this subscription to the ContextServer.

Explaining the code into detail, a developer has to instantiate a new object ①, and add a listener to it ②. It waits for *ContextChangedEvents*, which are fired when a change of a context occurs.

Listeners express their interest in receiving updates by subscribing to one or more classes of events.

They execute some code ③, when they catch an event, which is, in our case, fired by the *ContextObject*s.

Finally a C*ontextClient* is created ④ and told to subscribe the *LocationContext* ⑤.

Setting and retrieving a context is similar to the code fragment above.

4.4 Extending the Framework

Extending the framework with new types of context is simple. A new context just has to extend a class called *ContextObject* and to implement two abstract methods, namely *toXML()* and *fromXML()*. These two methods ensure compatibility with non-Java applications, because *toXML()* converts the values of the object into an XML-stream, which can be parsed by *fromXML()* and filled into an object. Furthermore, due to the fact that *ContextObject* implements the interface *Serializable*, a new, user-defined class may only contain data, which is serializable, or has to overwrite the methods *readObject* and *writeObject*. These methods are called, when an object is transmitted over a connection.

4.5 Exemplary Application

The feasibility of the architecture and the framework are demonstrated realizing a context-aware postbox. It sends and receives multimedia data such as images, text, and business cards and adapts it accordingly with respect to the network and device context.

The context-aware post box has to be executed on both devices, the sending device and the receiving device. The receiver is permanently listening for incoming data, while the sender can send data to any reachable device. The target has to be specified on application startup, but can be changed at any time during execution.

The context-aware postbox contains two different kinds of multimedia objects for demonstration purposes. They are adapted according to the network context of the transmission channel from one device to another, and alternatively the device properties of the target device are used, where the item should be sent.

The first kind of multimedia data is an image. The resolution of this image is adapted according to the available bandwidth or the device type of the recipient.

The second implemented object for demonstration is an intelligent business card. It decides dependent on the context, if it contains an image of the person or not.

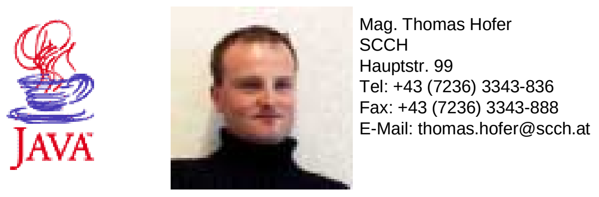
Mag. Thomas Hofer
SCCH
Hauptstr. 99
Tel: +43 (7236) 3343-836
Fax: +43 (7236) 3343-888
E-Mail: thomas.hofer@scch.at

Figure 5: Objects to be Transmitted using the Context Aware Postbox

Figure 5 shows the original data, which is used by our prototype. The first object is an image displaying the Java-logo. The business card shows the details of one of the authors including his image.

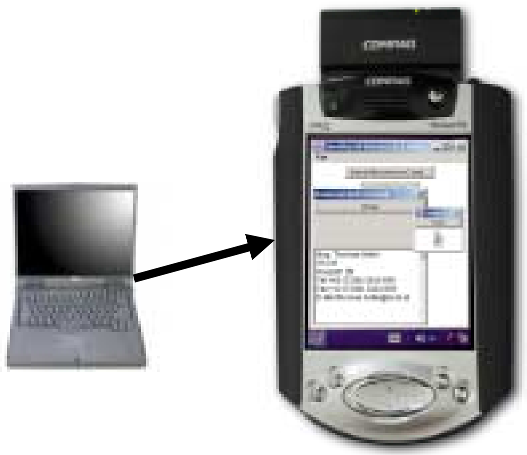

Figure 6: Image and Business Card received on a PDA

In the case that the data shown in Figure 5 is transmitted to a PDA [3] using a wireless connections, the data is adapted to the needs of a handheld device with its slow network connections. The resolution of the image is reduced, and it is converted to black-and-white, while the image of the business card is totally omitted.

Figure 6 shows the data received on the PDA.

Figure 7: Image and Business Card received on a Desktop Computer

Figure 7 shows same data transmitted to a Desktop PC. The data is not reduced because of a fast connection and much better device capabilities. This version contains an image within the business card and the Java-logo is colored and at a higher resolution.

4.6 Other Prototypical Applications

Other prototypical applications utilizing the framework like a location aware reminder have been realized. It reminds the user on predefined notes when reaching the places associated to the notes.

5 Open Issues and Future Work

This section discusses open issues and future work comprising the need for a comprehensive context model, an open XML-based interface and context sharing.

5.1 Comprehensive Context Model

At the moment the context-framework comprises some basic context classes only. One of the future tasks will be the integration of comprehensive context models like presented in [16]. The World Wide Web Consortium (W3C) is already working on a framework for the management of device and user profile information called "Composite Capabilities / Preference Profiles" (CC/PP) [29], which seems to be applicable to approach.

CC/PP is a framework for the management of device profile information, which is based on the Resource Description Framework (RDF) [28]. At the moment CC/PP tends to pay increased attention to well know contexts like device properties and user profiles, but new properties can be defined and included into the description.

5.2 XML-Protocol

Currently the framework provides a TCP/IP communication interface for applications, which are not implemented in Java. Therefore we use XML. All functionality, which is available for Java applications, is also available for XML-based commands. Furthermore all *Context Object*s can be converted into an XML-representation, which ensures that the *Context Object* can be processed by every application.

In further releases it is planned to use an XML-Schema [27] thus being able to validate commands and responses according to their syntactical correctness.

5.3 Context Sharing

Context-Sharing will offer the opportunity to gain information about the context of other context-aware devices, which are in physical proximity. This enriches any device with more context information than it can retrieve by itself.

The idea implies that two (or more) devices encounter each other, both with a *Context Server* executed. These two devices can have complementary context information, e.g. one device knows about the current location, the other device knows the temperature.

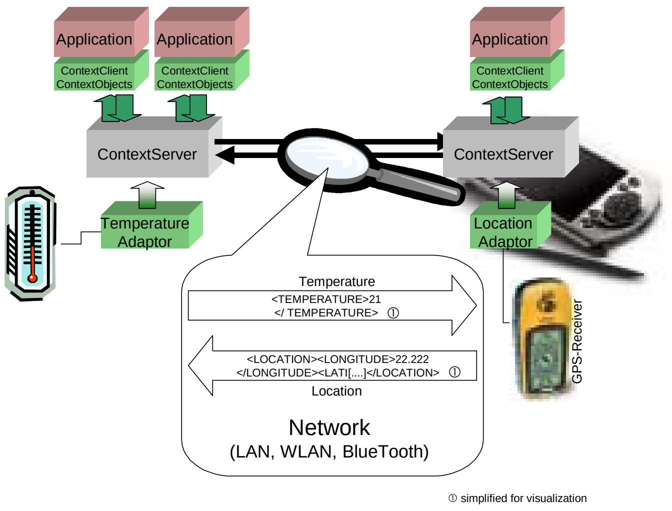
Application
Application
Application
ContextClient ContextObjects
ContextClient ContextObjects
ContextClient ContextObjects
ContextServer
ContextServer
Temperature Adaptor
Location Adaptor
GPS-Receiver
Temperature
<TEMPERATURE>21 </ TEMPERATURE> ①
<LOCATION><LONGITUDE>22.222 </LONGITUDE><LATI[....]</LOCATION> ①
Location
Network
(LAN, WLAN, BlueTooth)
① simplified for visualization

Figure 8: Context-Sharing on Multiple Devices

Context Sharing enables, as seen in Figure 8, both devices to have information about both kinds of context information. Simply to exchange all available information, however, is not enough. This idea raises additional open issues about security and reliability of this shared information.

Context, e.g. the user, must not be provided to all devices in range, in order to respect privacy and anonymity. The user has to keep in control which context information is allowed to be shared and which has to remain private.

Concerning reliability two (or even more) devices can provide contradictory information. Reasons for inconsistencies of shared data can be the use of sensors of different precision, or simply the scanning of "real" different values.

The Context-model should be extended to cover predefined testimonies about reliability of certain kinds of context in conjunction with device categories, possibilities to state the reliability in the runtime system and, finally, an (extensible) set of rules how to deal with the problems discussed above.

6 Conclusion

Motivated by the new possibilities, which context-awareness offers for applications, we proposed an architecture trimmed to the special needs of users using mobile devices. We identified requirements for a context-aware architecture, such as lightweightness, extensibility, robustness and the possibility to add meta-information. To meet these requirements we proposed a-three layered architecture, which comprises an application layer, a management layer and an adaptor layer.

We presented a framework, which implements our architecture and demonstrated its feasibility by a usage scenario.

7 Acknowledgements

The authors gratefully acknowledge support of the K plus Competence Center Program, which is funded by the Austrian Government, the Province of Upper Austria, and the Chamber of Commerce of Upper Austria.

8 References

[1] G. D. Abowd: “Software Engineering Issues for Ubiquitous Computing” Int. Conf. on Software Engineering, Los Angeles, 1999.

[2] G. D. Abowd and E. D. Mynatt, “Charting past, present, and future research in ubiquitous computing”, ACM Trans. Comput.-Hum. Interact. 7, 1 (Mar. 2000), Pages 29 – 58

[3] http://athome.compaq.com/showroom/static/ iPAQ/handheld_jumppage.asp

[4] P. Couderc, A.M. Kermarrec, “Improving Level of Service for Mobile Users Using Context-Awareness”, 18th IEEE Symposium on Reliable Distributed Systems, Lausanne, Switzerland, October 18 - 21, 1999

[5] A. K. Dey, D. Salber, G. D. Abowd, M. Futakawa , “An Architecture To Support Context-Aware Applications”, 12th Annual ACM Symposium on User Interface Software and Technology, 1999

[6] A. K. Dey and G. D. Abowd, “The Context Toolkit: Aiding the Development of Context-Aware Applications”, In the Workshop on Software Engineering for Wearable and Pervasive Computing, Limerick, Ireland, June 6, 2000.

[7] A. K. Dey and G. D. Abowd, “CybreMinder: A Context-Aware System for Supporting Reminders”, Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K), Bristol, UK, September 25-27, 2000. pp. 172-186.

[8] A. K. Dey, “Understanding and Using Context”, Personal and Ubiquitous Computing, http://www.personal-ubicomp.com/, ISSN: 0949-2054, Volume 5 Issue 1 (2001) pp 4-7, Springer

[9] A. K. Dey, D. Salber and G. D. Abowd, “A Context-based Infrastructure for Smart Environments”, In Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE '99) , Dublin, Ireland, December 1999.

[10] F. Espinoza, P. Persson, A. Sandin, H. Nyström, E. Cacciatore, M. Bylund, “GeoNotes: Social and Navigational Aspects of Location-Based Information Systems”, HUMLE Lab, Swedish Institute of Computer Science (SICS), in Abowd, Btummit & Shafer (eds): Ubicomp 2001, Ubiquitous Bcomputing, Internetation Conference Atlanta, Sep. 30 – Oct. 2, Berlin, Springer, p. 2-17

[11] A. Ferscha, W. Beer, W. Narzt, “Location Awareness in Community Wireless LANs”, 31. Jahrestagung der deutschen Gesellschaft für Informatik, Jahrestagung der Österreichischen Computer Gesellschaft, Mobile internet based services and information logistics, Workshop, GI/ÖCG-Jahrestagung 2001, Vienna, Austria, September 2001.

[12] M. Großmann, A. Leonhardi, B. Mitschang, K. Rothermel: “A World Model for Location-Aware Systems”. Informatik, 8(5), 2001.

[13] www.insignia.com

[14] www.ibm.com

[15] G. Kappel, W. Schwinger, B. Pröll, W. Retschitzegger, T. Hofer, “Modeling Ubiquitous Web Applications - A Comparison of Approaches”, In: W. Winiwarter, St. Bressan, I.K. Ibrahim (editors), Third International Conference on Information Integration and Web-Based Applications & Services (iiWAS 2001), 10.-12.09.2001, ISBN 3-85403-150-5, pp. 163-174

[16] G. Kappel, W. Retschitzegger, E. Kimmerstorfer, B. Pröll, W. Schwinger, T. Hofer, “Towards a Generic Customisation Model for Ubiquitous Web Applications”, 2nd International Workshop on Web Oriented Software Technology; IWWOST'2002; Málaga, Spain; 10 June, 2002

[17] G. Kappel et al, “Customising Web Applications Towards Ubiquity - The Notion and the Issues”, W3C Workshop on Delivery Context, Sophia-Antipolis, France, March 2002

[18] L. Kleinrock: “Nomadicity: Anytime, Anywhere In A Disconnected World”, Mobile Networks and Applications, Jan. 1996

[19] A. Kobsa: “Generic User Modeling Systems”, User Modeling and User-Adapted Interaction, Vol. 11, 2001.

[22] J. Nord et al., “An Architecture for Location Aware Applications ", Department of Computer Science,Lulea University of Technology, Sweden, Proceedings of the Hawai'i International Conference on System Sciences (HICSS35), Big Island, Hawaii, January 7 – 10, 2002

[23] J. R. Rodriguez et al. :”Extending e-business to Pervasive Computing Devices - Using WebSphere Everplace Suite Version 1.1.2”, IBM Redbooks, International Technical Support Organisation, SG24-5996-00, 2001.

[24] A. Schmidt, M. Beigl and H. W. Gellersen. “There is more to Context than Location”. In: Computers & Graphics Journal, Elsevier, Volume 23, No.6, December 1999, pp 893-902.

[25] http://Java.sun.com/products/personalJava/

[26] http://Java.sun.com/j2me

[27] WORLD WIDE WEB CONSORTIUM (W3C) 2000, XML Schema, http://www.w3.org/XML/Schema, 2000

[28] World Wide Web Consortium (W3C), Resource Description Framework (RDF), http://www.w3.org/RDF, 2000.

[29] WORLD WIDE WEB CONSORTIUM (W3C) 2001a. Composite Capabilities/Preference Profiles, http://www.w3.org/Mobile/CCPP/, 2001

[30] S. Weber, J. Jennings, “Architectural Issues for Pervasive Computing”, IBM TJ Watson Research, Workshop on Software Engineering for Wearable and Pervasive Computing, ICSE 2000

[31] M. Weiser: "The Computer for the 21st Century", Scientific American, 265, 3, September 1991.