Technical University of Vienna
Information Systems Institute
Distributed Systems Group

A Survey on Context-aware systems

Matthias Baldauf and
Schahram Dustdar
e9902330@student.tuwien.ac.at
dustdar@infosys.tuwien.ac.at

TUV-1841-2004-24 November 30, 2004

Context-aware systems offer entirely new opportunities for application developers and for end users by gathering context data and adapting systems' behavior accordingly. Especially in combination with mobile devices such mechanisms are of great value and claim to increase usability tremendously. In this paper, we present a layered architectural framework for context-aware systems. Based on our suggested framework for analysis, we introduce various existing context-aware systems focusing on context-aware middleware and frameworks, which ease the development of context-aware applications. We discuss various approaches and analyze important aspects in context-aware computing on the basis of the presented systems.

Keywords: Context-awareness, Layer architecture, Context framework, Context middleware

Argentinierstr. 8/184-1
A-1040 Vienna, Austria
phone: +43 1 58801-18402
fax: +43 1 58801-18491
URL: http://www.infosys.tuwien.ac.at/

A Survey on Context-aware systems

Matthias Baldauf, Schahram Dustdar
Distributed Systems Group, Institute of Information Systems,
Vienna University of Technology
{e9902330@student.tuwien.ac.at | dustdar@infosys.tuwien.ac.at}

Abstract. Context-aware systems offer entirely new opportunities for application developers and for end users by gathering context data and adapting systems' behavior accordingly. Especially in combination with mobile devices such mechanisms are of great value and claim to increase usability tremendously. In this paper, we present a layered architectural framework for context-aware systems. Based on our suggested framework for analysis, we introduce various existing context-aware systems focusing on context-aware middleware and frameworks, which ease the development of context-aware applications. We discuss various approaches and analyze important aspects in context-aware computing on the basis of the presented systems.

Keywords: Context-awareness, Layer architecture, Context framework, Context middleware

1 Introduction

With the appearance and penetration of mobile devices such as notebooks, PDAs, and smart phones, pervasive (ubiquitous) systems become increasingly more popular these days. The term 'pervasive' introduced first by Mark Weiser in 1991 [38] refers to the seamless integration of devices into the users' everyday life. Appliances should vanish in the background to make the user and his tasks the central focus rather than computing devices and technical issues.

One part in the wide range of pervasive computing are the so called context-aware (or sentient) systems. Context-aware systems are able to adapt their operations to the current context without explicit user intervention and thus aim at increasing usability and effectiveness by taking environmental context into account. Especially when using mobile devices it is desirable that programs and services react specifically to their current location, time and other environment attributes and adapt their behavior to changing circumstances as context data may change rapidly. The needed context information may be retrieved in a variety of ways like applying sensors, network information, device status, browsing user profiles and using other sources.

The history of context-aware systems started in 1992 when Want, Hopper et al introduced their 'Active Badge Location System' [39] which is considered to be one of the first context-aware applications. The infrared technology based system is able to determine a user's current location which was used to forward phone calls to a

telephone close to the user. In the middle of the 1990s a couple of location-aware tour guides [23, 40, 41] emerged which provided information according to the user's current location. While location is the most used attribute of context by far attempts to use other context information as well grew over the last few years as the examples in this paper show.

Hence, it is a challenging task to define the word 'context' and many researchers tried to find their own definition for what context actually includes. In the literature the term 'context-aware' first appeared in [1] where the authors describe context as location, identities of nearby people and objects and changes to those objects. Such enumerations of context examples were often used in the beginning of context-aware system history. In [2] Ryan, Pascoe and Morse referred to context as the user's location, the environment, the identity and the time. Dey [3] enumerates context as the user's emotional state, focus of attention, location and orientation, date and time, objects and people in the user's environment. Another common way of defining context was the use of synonyms. Hull, Neaves and Bedford-Roberts describe context as the aspects of the current situation [4]. Definitions like this one are often too wide, a good one is found in [5]: Brown defines context to be the elements of the user's environment that the computer knows about. One of the best topical definitions is found by Dey and Abowd [6]. The authors refer to context as 'any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves'.

One popular way to classify context instances is the distinction of different context dimensions. In [7, 8] these dimensions are called 'external' and 'internal', [9] refers to 'physical' and 'logical' context. The 'external' ('physical') dimension means context that can be measured by hardware sensors, i.e. location, light, sound, movement, touch, temperature, air pressure etc. Whereas the 'internal' ('logical') dimension is mostly specified by the user or captured monitoring the user's interaction, i.e. the user's goals, tasks, work context, business processes, the user's emotional state etc.

Most context-aware systems make use of external context factors as they provide useful data like location information etc. Furthermore external attributes are easy to sense due to off-the-shelf sensing technologies. Virtually all systems presented in this paper apply physical context information. Examples for the use of logical data are the Watson Project [10] and the IntelliZap Project [11] which support the user providing relevant information due to information read out of opened web pages, documents etc.

When dealing with context three entities can be distinguished [12]: *places* (rooms, buildings etc.), *people* (individuals, groups) and *things* (physical objects, computer components etc.). Each of these entities may be described by various attributes summarized to four main categories: *identity* (each entity has an unique identifier), *location* (an entity's position, co-location, proximity etc.), *status* (or activity, meaning the intrinsic properties of an entity, e.g. temperature and lightning for a room, processes running currently on a device etc.) and *time* (used for timestamps to accurately define situation, ordering events etc.).

The reminder of this paper is structured as follows. Section 2 introduces current design principles for context-aware systems describing requirements for their architecture and the used context model. In section 3 we present a comparison of

existent context-aware systems and explain the approaches' varieties and similarities. Finally section 4 draws some concluding remarks.

2 Design Principles

2.1 Architecture

Context-aware systems can be implemented in many ways. The approach depends on special requirements and conditions like e.g. the location of sensors (local or remote), the amount of possible users (one user or many), the available resources of the used devices (high-end-PCs or small mobile devices) or the facility of a further extension of the system. Based on these considerations three different approaches of context-aware system architectures can be distinguished [13]:

Direct sensor access: This approach is often used in devices with sensors locally built in. The client software gathers the desired information directly from these sensors, i.e. there is no additional layer for gaining and processing sensor data. Drivers for the sensors are hardwired into the application, so this tight coupled method is usable only in rare cases as it complicates extensibility. Also it is not suited for distributed systems due to its direct access nature without any component capable of managing multiple concurrent sensor accesses.

Middleware based: Modern software design uses methods of encapsulation to separate e.g. business logic and graphical user interfaces. The middleware based approach introduces a layered architecture to context-aware systems with the intention of hiding low-level sensing details. Compared to direct sensor access this technique eases extensibility since the client code has not be modified anymore and it simplifies the reusability of hardware dependent sensing code due to the strict encapsulation.

Context server: The next logical step is to permit multiple clients access to remote data sources. This distributed approach extends the middleware based architecture by introducing an access managing remote component. Gathering sensor data is moved to this so called context server to facilitate concurrent multiple access. Beside the reuse of sensors the usage of a context server has the advantage of relieving clients of resource intensive operations. As probably the majority of end devices used in context-aware systems are mobile gadgets with limitations in computation power, disk space etc. this is an important aspect. Sadly there is no free lunch: in return one has to consider about appropriate protocols, network performance, quality of service parameters etc. when designing a context-aware system based on client-server architecture.

In a similar manner Winograd [17] describes three different context management models for coordinating multiple processes and components:

Widgets: Derived from the homonymous GUI elements a widget is a software component that provides a public interface for a hardware sensor [12]. They hide low-level details of sensing and ease application development due to their reusability. Because of the encapsulation in widgets it is possible to exchange widgets which

provide the same kind of context data (e.g. exchange a radio frequency widget by a camera widget to collect location data). Widgets are usually controlled by some kind of a widget manager. The tight coupled widget approach increases efficiency but is not robust to component failures.
Networked services: This more flexible approach, argued for e.g. in [42], resembles the context server architecture. Instead of a global widget manager discovery techniques are used to find networked services. This service based approach is not as efficient as a widget architecture due to complex network based components but provides robustness.

Blackboard model: In contrast to the process-centric view of the widget and the service-oriented model the blackboard model represents a data-centric view. In this asymmetric approach processes post messages to a shared media, the so called blackboard, and subscribe to it to be notified when some specified event occurs. Advantages of this model are the simplicity of adding new context sources and the easy configuration. Unfavorable is the need of a centralized server to host the blackboard and the lacks in communication efficiency as two hops per communication are needed.

In this paper we will focus on middleware based and context-server based systems due to their usability in distributed systems. Many layered context-aware systems and frameworks have evolved during the last years. Most of them differ in the functional range, location and naming of the layers, the use of optional agents etc. Beside these adaptations and modifications a common architecture in modern context-aware applications is identifiable when analyzing their design.

As mentioned above a separation of detecting and using context is necessary to improve extensibility and reusability of systems. The following abstract architecture augments layers for detecting and using context by adding interpreting and reasoning functionality [12, 14].

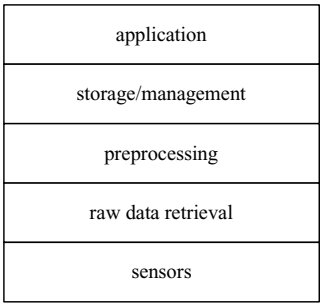

Fig. 1. Abstract layer architecture for context-aware systems

The first layer consists of the sensors. It is notable that the word ‘sensor’ not only refers to sensing hardware but to every data source which may provide usable context information. Concerning to the way data are captured sensors can be classified in three groups [15].

Physical sensors: The most frequently used type of sensors are physical sensors. Many hardware sensors are available nowadays capable of capturing almost any physical data. Table 1 shows some examples of physical sensors [16]:

Type of context	Available Sensors
Light	Photodiodes, color sensors, ir and uv-sensors etc.
Visual Context	Various cameras
Audio	Microphones
Motion, Acceleration	Mercury switches, angular sensors, accelerometers, motion detectors, magnetic fields
Location	Outdoor:Global Positioning System (GPS), Global System for Mobile Communications (GSM); Indoor: Active Badge system etc.
Touch	Touch sensors implemented in mobile devices
Temperature	Thermometers
Physical attributes	Biosensors to measure skin resistance, blood pressure

Table 1. List of different physical sensor types

Virtual sensors: Virtual sensors source context data from software. E.g. it is possible to determine an employee's location not only by using tracking systems (physical sensors) but also by a virtual sensor, e.g. by browsing an electronic calendar, a travel-booking system, emails etc. for location information. Other context attributes that can be sensed by virtual sensors include e.g. the user's activity by checking for mouse-movement and keyboard input.

Logical sensors: These sensors make use of a couple of information sources, they combine physical and virtual sensors with additional information from databases etc. to solve a higher task. E.g. by analyzing logins at desktop pcs and a database mapping fixed devices to location information a logical sensor can be constructed to detect an employee's current position.

The second layer is responsible for the retrieval of raw context data. It makes use of appropriate drivers for physical sensors and APIs for virtual and logical sensors. The query functionality if often implemented in reusable software components which make low-level details of hardware access transparent by providing more abstract methods like getPosition() etc. By using interfaces for components responsible for equal types of context these components become exchangeable. So it is possible e.g. to replace a RFID system by a GPS system without any major modifications.

The next layer is not implemented in every context-aware system but may offer useful information if the raw data are too coarse grained: this preprocessing layer is responsible for reasoning and interpreting. The sensors queried in the underlying layer most often return technical data that are not appropriate to use by application designers, hence this layer raises the results of layer two to a higher abstraction level. The transformations include extraction and quantization operations. E.g. the exact GPS position of a person might not be of value for an application but the name of the room the person is in is needed.

In context-aware systems consisting of several different context data sources the single context atoms can be combined to high-level information in this layer. This

process is also called aggregation or compositing. A single sensor value is often not important to an application; combined information might be more precious. In this vein a system is able to determine e.g. whether a client is situated indoor or outdoor by analyzing various physical data like temperature and light or whether a person is currently attending a meeting by capturing noise level and location etc. To make this analysis work correctly a lot of statistical methods are involved and often some kind of training phase is required.

Obviously, this abstraction functionality could also be implemented directly by the application. But due to a couple of reasons this task should be encapsulated and better moved to the context server. The encapsulation advances the reusability and hence eases the development of client applications. And by making such aggregators remote accessible the network performance increases (as clients have to send only one request to gain high-level data instead of connecting to various sensors) and limited client resources are saved.

The problem of sensing conflicts that might occur when using several data sources has also to be solved in this layer. E.g. when a system is notified about a person's location by the coordinates of her mobile phone and a camera spotting this person it might be difficult to decide what information to use. Often this conflict is approached by using additional data like time stamps and resolution information.

The fourth layer organizes the gathered data and offers them via a public interface to the client. Access by clients may happen in two different ways: synchronous and asynchronous. In the synchronous manner the client is polling the server for changes via remote method calls: it sends a message requesting some kind of offered data and pauses until it receives the server's answer. The asynchronous mode works via subscriptions: at the program's start the client subscribes to specific events it is interested in. When one of these events occurs the client is either simply notified or a client's method is directly involved using a callback.

In the majority of cases the asynchronous approach is more suitable due to rapid changes in the underlying context. The polling technique is more resource intensive as context data has to be requested quite often and the application has to prove for changes itself using some kind of context history.

The client is realized in the fifth layer, the application layer. The actual reaction on different events and context-instances is implemented here. Sometimes information retrieval and application specific context management and reasoning is encapsulated in form of agents which communicate with the context server and act as an additional layer between the preprocessing and the application layer [13]. An example for context logic at the client side is the display on mobile devices: as a light sensor detects bad illumination text may be displayed in higher color contrast.

2.2 Context Models

A context model is needed to define and store context in a machine processible form. To develop flexible and efficient context ontologies that cover the wide range of possible contexts is a challenging task. The most important goals when designing a context ontology include [20]:

Simplicity: The used expressions and relations should be as simple as possible to simplify the work of applications developers.

Flexibility and extensibility: The ontology should support the simple addition of new context elements and relations.

Genericity: The ontology should not be limited to special kind of context atoms but rather support different types of context.

Expressiveness: The ontology should allow to describe as much context states as possible in arbitrary detail.

There are tools available to define declarative representations and to publish and share ontologies developed by the World Wide Web Consortium, e.g. the Resource Description Language (RDF) [19, 20, 21] and the Web Ontology Language OWL [22, 32, 34, 35]. A single context atom can be described with a couple of attributes. The two most obvious are

Context type: The context type refers to the category of context like temperature, time, speed etc. This type information may be used a parameter for a context query or a subscription, e.g. subscribeToChanges("temperature"). It is important to use meaningful type names, hence as the system grows some names might not be unique anymore. For example the type 'position' may belong to a mobile device or a user. One solution to create a well-structured type names is the use of cascaded names [20] as shown in table 2.

Context value: Context value means the raw data gathered by a sensor. The unit depends on the context type and the applied sensor, e.g. degree Celsius, miles per hour etc.

In most cases context type and context value are not enough information to build a working context-aware system. Additional attributes that might be useful include

Description: A literal description containing details about the context atom. The attribute is especially helpful to application developers when new sensors can be dynamically added to the system.

Time stamp: This attribute contains a date/time-value describing when the context was sensed. It is needed e.g. to create a context history and deal with sensing conflicts.

Source: A field containing information how the information was gathered. In case of an hardware sensor it might hold the ID of the sensor and allow an application to prefer data from this sensor.

Confidence: The confidence attribute describes the uncertainty of this context type. Not every data source delivers accurate information. E.g. location data suffers inaccuracy dependent on the used tracking tool.

Part of a flexible context model is an extendable context vocabulary to deal with abstract descriptions rather than technical data. It simplifies the description of various context atoms and context instances. These verbal descriptions are often based on subjective impressions and mostly implemented using fuzzy sets. In [20] 'context' refers to this attribute.

Table 2 shows a small part of an example vocabulary. Notice that not all contexts have to be available at a time. In contrast to temperature a light source is not always measurable.

Context type	Context
Environment:Temperature	Cold
Environment:Temperature	Normal
Environment:Temperature	Hot
Environment:Light:Source	50Hz
Environment:Light:Source	60Hz
Environment:Light:Source	NotAvailable
Device:Activity:Placement	AtHand
Device:Activity:Placement	NotAtHand

Table 2. Example context vocabulary [20]

Based on this vocabulary above instances of context atoms can be created (Table 3).

Context type	Context value	Context	Confidence	Source	Timestamp
Environment:Temperature	21 °C	Normal	0.9	Sensor #2	05-25-04 13:36:14
Device:Activity:Placement	-	AtHand	1	Sensor #5	05-25-04 15:12:57

Table 3. Example context atoms

3 Existent systems and frameworks

3.1 Location-aware systems

Context-aware systems dealing with location information are widespread and the demand for them grows due to the increasing spread of mobile devices. Although by location mostly a user's whereabouts is meant the term also refers to the location of devices and services.

Famous examples for location-aware systems are various tourist guide projects where information dependent to the current location is displayed, other examples can be found in [25, 26, 27, 28]. A couple of different location aware infrastructures are available to collect position data: GPS satellites, mobile phone towers, badge proximity detectors, cameras, magnetic card readers, barcode readers etc. These sensors can provide either position or proximity information, the appropriate sensor depends on the use: they differ in price, accuracy, some need a clear line of sight, other signals may travel through walls etc.

As a detailed example we introduce an indoor location sensing system: In [24] Harter et al present a location-aware system using ultrasonic technique. To each entity (person or equipment) that should be detectable a small sending unit called bat is

attached. These bats have globally unique identifiers and contain ultrasonic transducers. To monitor the signals sent by the bats receivers are installed at the rooms' ceilings and connected by a wired network. The third needed hardware type is a base station. It periodically sends radio messages with specific bat ids and resets the receivers. A corresponding bat reacts emitting an ultrasonic impulse which is caught by the receivers. By recording the time of arrival of signals the distance between the bat and the receiver can be calculated. The bat's exact position is then determined using multilateration (an extension of trilateration). A challenge the authors where confronted with due to the use of ultrasonic technique was the incorrect measurement because of unwanted reflections of the signals. The problem could be solved by using a statistical outlier rejection algorithm to improve the accuracy of the calculated positions.

3.2 Context-aware systems

The systems named in the prior chapter use only one aspect of context, namely location information. The use of different types of context atoms like e.g. noise, light and location allows the combination to high-level context objects. These elements are necessary to build more adaptive, useful and user-friendly systems.

As example for this kind of context-aware infrastructures serves the system presented by Muñoz et al [29] which extends the instant messaging paradigm by adding context-awareness to support information management within a hospital setting. All users (in this case physicians, nurses etc.) are equipped with mobile devices to write messages that are sent when a specified set of circumstances is satisfied. For example a user can formulate a message that should be delivered to the first doctor that enters room number 108 after 8 a.m. The contextual elements this system is aware of include location, time, roles and device state.

Its context functionality is moved to agents which include three modules (layers). The perception module gathers raw context information from data sources (sensors, users, other agents, the server). The reasoning module governs the agent's actions and finally the action module triggers a user-specified event. All messages between agents are XML encoded.

3.3 Context-aware frameworks

Context-aware systems capable of dealing with special types of context are well-suited for specific conditions, e.g., the hospital scenario. These systems can be optimized for the situations they are used in, they do not have to be flexible and extensible. To really simplify the developing of context-aware applications rather an abstract framework is needed. Such a generic infrastructure not only provides client access to retrieve context data, it also permits the simple registration of new distributed heterogeneous data sources.

In this section different context-aware frameworks are introduced and compared based on various design decisions.

Architecture

The most common design approach for distributed context-aware frameworks is a classical hierarchical infrastructure with one or many centralized components using a layered architecture as presented in Section 2. This approach is useful to overcome memory and processor constraints of small mobile devices but provides one single point of failure and thereby lacks of robustness.

The architecture of the Context Managing Framework presented by Korpipää et al in [19] is depicted in figure 2. Four main functional entities comprise this context framework: the context manager, the resource servers, the context recognition services and the application.

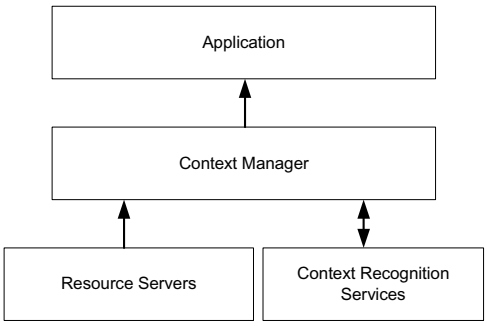

Fig. 2. Architecture of the Context Managing Framework

Whereas the resource servers and the context recognition services are distributed components, the so-called context manager represents a centralized server managing a blackboard: it stores context data and serves information to the clients applications.

The SOCAM (Service-oriented Context-Aware Middleware) project introduced by Gu et al [32] is another architecture for the building and the rapid prototyping of context-aware mobile services. It uses a central server as well, here called context interpreter, which gains context data through distributed context providers and offers it in mostly processed form to the clients. The context-aware mobile services are located on top of the architecture: they make use of the different levels of context and adapt their behavior according to the current context.

One further extensible centralized middleware approach designed for context-aware mobile applications is a project called CASS (Context-awareness sub-structure) [43]. Figure 3 shows the system's composition.

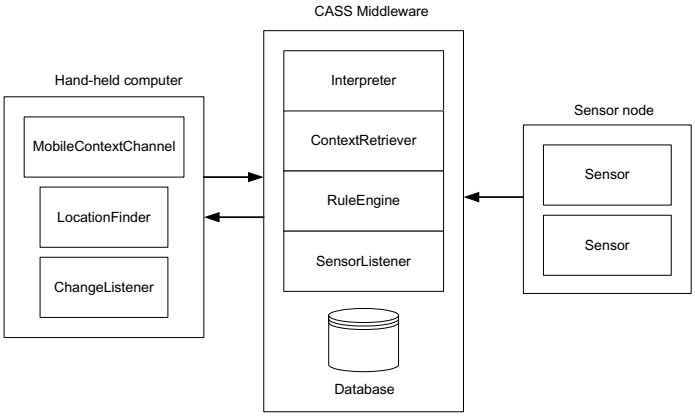

Fig. 3. Architecture of the CASS system

The middleware contains an Interpreter, a ContextRetriever, a Rule Engine and a SensorListener. The SensorListener listens for updates from sensors which are located on distributed computers called sensor nodes. Then the gathered information is stored in the database by the SensorListener. The ContextRetriever is responsible for retrieving stored context. Both of these classes may use the services of an interpreter. The ChangeListener is a component with communications capabilities, that allows a mobile computer to listen for notification of context change events. Sensor and LocationFinder classes also have built-in communications capabilities. Mobile clients connect to the server over wireless networks. To reduce the impact of intermittent connections local caching on the client side is supported.

CoBrA (Context Broker Architecture) [34] is an agent based architecture for supporting context-aware computing in so called intelligent spaces. Intelligent spaces are physical spaces (e.g. living rooms, vehicles, corporate offices and meeting rooms) that are populated with intelligent systems that provide pervasive computing services to users. Central to CoBrA is the presence of an intelligent context broker that maintains and manages a shared contextual model on the behalf of a community of agents. These agents can be applications hosted by mobile devices that a user carries or wears (e.g. cell phones, PDAs and headphones), services that are provided by devices in a room (e.g. projector service, light controller and room temperature controller) and web services that provide a web presence for people, places and things in the physical world (e.g. services keeping track of people's and objects' whereabouts). The context broker consists of four functional main components: the Context Knowledge Base, the Context Inference Engine, the Context Acquisition Module and the Privacy Management Module. To avoid the bottle neck problem CoBrA offers the possibility of creating broker federations.

The Context Toolkit [12, 30], another context-aware framework, takes a step towards a peer-to-peer architecture but it still needs a centralized discoverer where distributed sensor units (called widgets), interpreters and aggregators are registered in order to be found by client applications. The toolkit's object-oriented API provides a

superclass called BaseObject which provides generic communications abilities to ease the creation of own components.

Another framework based on a layered architecture is built in the Hydrogen project [9]. Its context acquisition approach is specializing in mobile devices. While in the majority of existent distributed content-aware systems the working of a centralized component is essential, the Hydrogen system tries to avoid this dependency. It distinguishes between a remote and a local context: remote context is information another device knows about, local context is knowledge our own device is aware of. When the devices are in physical proximity they are able to exchange these contexts in a peer-to-peer manner via WLAN, Bluetooth etc. This exchange of context information among client devices is called "context sharing". Figure 4 shows the management of a device's context which consists of its own local context and a set of remote contexts gathered from other devices. Both local and remote context are made up of context objects. The superclass ContextObject is extended by different context types, e.g. LocationContext, DeviceContext etc. This approach allows the simple addition of new context type by specializing ContextObject. A context type has to implement ContextObject's toXML and fromXML methods to convert the data to a XML stream.

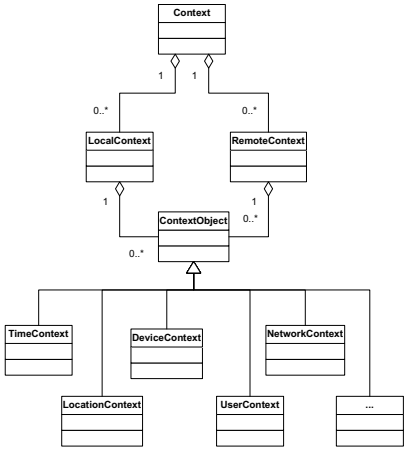

Fig. 4. Hydrogen's object oriented approach to manage local and remote contexts

The architecture consists of three layers which are all located on the same device (figure 5). The Adaptor layer is responsible for retrieving raw context data by querying sensors. This layer permits a sensor's concurrent use by different applications. The second layer, the Management layer, makes use of the Adaptor layer to gain sensor data and is responsible for providing and retrieving contexts. The so called 'context server' offers the stored information via synchronous and asynchronous methods to the client applications. On top of the architecture is the Application layer where appliance code is implemented to react on specific context changes reported by the context manager. Due to platform and language independency all inter-layer communication is based on a XML-protocol.

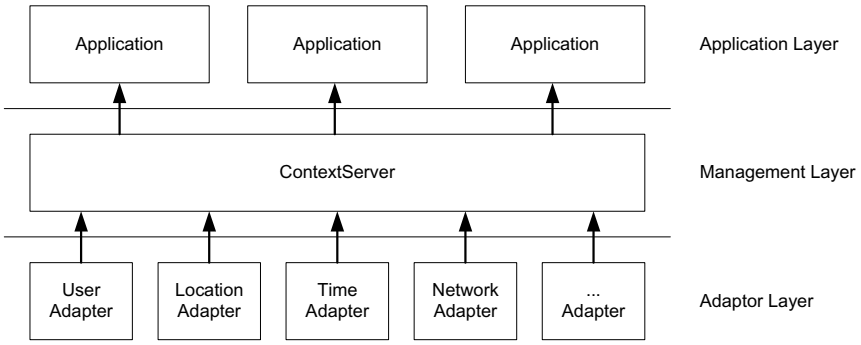

Fig. 5. Architecture of the Hydrogen project

The CORTEX system is an example for a context-aware middleware approach. Its architecture is based on the Sentient Object Model [31] which was designed for the development of context-aware applications in an ad-hoc mobile environment. The model's special suitability for mobile applications depends on the use of STEAM, a location-aware event-based middleware service designed specifically for ad-hoc wireless networking environments.

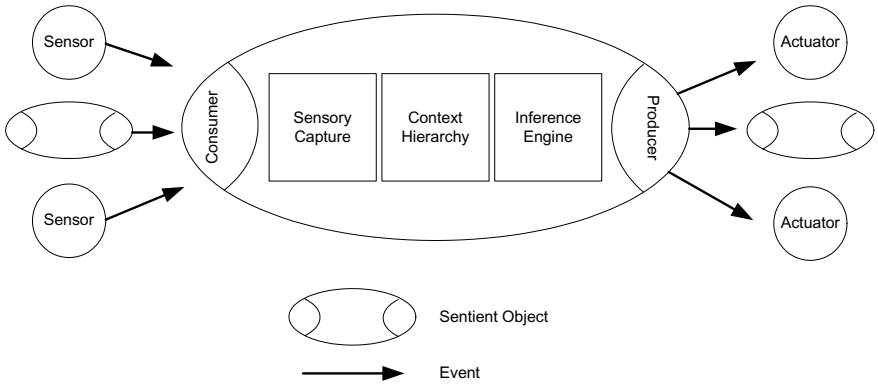

Fig. 6. The Sentient Object Model

A sentient object is an encapsulated entity consisting of three main parts (figure 6): sensory capture, context hierarchy and inference engine. Via interfaces a sentient objects communicates with sensors which produce software events and actuators which consume software events. As figure 6 shows: sentient objects can be both producer and consumer of another sentient object. Own sensors and actuators are programmed using STEAM. For building sentient objects a graphical development tool is available which allows developers to specify relevant sensors and actuators, define fusion networks, specify context hierarchies and production rules, without the need to write any code.

The Gaia project [36, 37], another middleware infrastructure, extends typical operating system concepts to include context-awareness. It aims at supporting the development and execution of portable applications for active spaces. Gaia exports services to query and utilize existing resources, to access and use current context, and provides a framework to develop user-centric, resource-aware, multi-device, context-sensitive and mobile applications. The current system consists of the Gaia kernel and the application framework (figure 7).

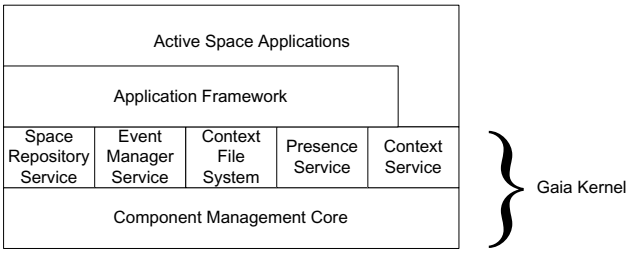

Fig. 7. Architecture of the Gaia system

In this paper, we focus on Gaia's parts concerning context-awareness, namely the Event Manager, the Context Service and the Context File System. The Event Manager service is responsible for event distribution in the active space and implements a decoupled communication model based on suppliers, consumers, and channels. Each channel has one or more suppliers that provide information to the channel and one or more consumers that receive the information. The reliability is increased as suppliers are exchangeable. With the help of the Context Service applications may query and register for particular context information and higher level context objects. And finally the Context File System makes personal storage automatically available in the users' present location. It constructs a virtual directory hierarchy to represent context as directories, where path components represent context types and values. For example, to determine which files have the context of location == RM2401 && situation == meeting associated with them, one may enter the /location:/RM2401/situation:/meeting directory.

Resource Discovery

As sensors in a distributed network may fail or new ones may be added, a discovery mechanism to search for and find appropriate sensors at runtime is important.

For these purposes the Context Toolkit offers the already mentioned discoverer. The discoverer works as registry component which interpreters, aggregators and widgets have to notify about their presence and their contact possibilities. After registration the components are pinged to ensure that they are operating. If a component does not respond to a specified number of consecutive pings, the discoverer determines that the component is unavailable and removes it from its registry list. Customers may find appropriate components querying the discoverer either via a white page lookup (a search for the component's name) or a yellow page

lookup (a search for specific attributes). In case the lookup was successful the discoverer returns a handle to contact the context component.

SCAM offers a discovery mechanism as well called service locating service. In Gaia different context providers are stored in a registry component. A pure peer-to-peer context-aware system like Hydrogen only uses local built-in sensors and does not connect to distributed sensors therefore no discovery mechanism is involved.

Sensing

The Context Toolkit's authors presented a new approach to handle different data sources. Derived from the use of widgets in GUI development they introduced so called context widgets to separate applications from context acquisition concerns. In these widgets the complexity of sensing is hidden, further they abstract the gained context information (e.g. the accurate position of a person might not be of value but the application should be notified when this person enters another room) and as widgets are encapsulated software components they are reusable. Each widget owns some attributes that can be queried by applications, e.g. the IdentityPresence widget implemented by the authors offers attributes like its location, the last time a presence was detected and the identity of the last user detected. Beside the polling mechanism an asynchronous way of data retrieval is possible too: if an application subscribes to a widget it is notified when the widget's context changes. The IdentityPresence provides the callbacks PersonArrives(location, identity, timestamp) and PersonLeaves(location, identity, timestamp) which are triggered when a person either arrives or leaves a room. The separation of acquisition and use of context permits a simple exchange of widgets since e.g. identity may be sensed in various ways like Active Badges, video recognition etc.

This manner of building reusable sensor units that make the action of sensing transparent to the customer (whether it is a centralized server or a distributed client component) became widely accepted in distributed context-aware systems: CASS applies 'sensor nodes', SOCAM uses 'context providers', the Context Managing Framework refers to 'resource servers', CoBrA makes use of 'context acquisition components'.

Context Model

A efficient model for handling, sharing and storing context data is essential for a working context-aware system. The Context Toolkit handles context in simple attribute-value-tuples which are encoded using XML for transmission.

As already described above Hydrogen uses an object-oriented context model approach with a superclass called ContextObject which offers abstract methods to convert data streams from XML representations to context objects and vice versa.

More advanced ways of dealing with context data based on ontologies are found in SOCAM, CoBrA and the Context Managing Framework. SOCAM's authors divide a pervasive computing domain into several sub-domains, e.g. home domain, office domain etc., and define individual low-level ontologies in each sub-domain to reduce the complexity of context processing. Each of these ontologies implemented in OWL provides a special vocabulary used for representing and sharing context knowledge.

CoBrA also uses an own OWL-based ontology approach, namely COBRA-Ont [34, 35]. The following lines show a short part of an COBRA-Ont example:

```
<loc:LocationContext>

    <rdf:type rdf:resource="&tme;InstantThing"/>

    <loc:locationContextOf>

      <per:Person>

          <per:name rdf:datatype="&xsd;string">Harry
Chen</per:name>

      </per:Person>

    </loc:locationContextOf>

    <loc:boundedWithin rdf:resource="&ebgeo;Japan"/>

    <tme:at rdf:datatype="&xsd;dateTime">2004-02-
23T11:23:00</tme:at>

  </loc:LocationContext>
```

The ontology's structure and vocabulary applied in the Context Managing Toolkit are described in RDF. Parts of its vocabulary are used as example in Section 2, see table 2 and 3.

In Gaia context is represented in a special manner, namely through 4-ary predicates in the way Context(<ContextType>, <Subject>, <Relater>, <Object>) written in DAML+OIL. The Context Type refers to the type of context the predicate is describing, the Subject is the person, place or thing with which the context is concerned, and the Object is a value associated with the Subject. The Relater relates the Subject and the Object such as a comparison operator (=, >, or <), a verb, or preposition. An example for a context instance is Context(temperature, room 3231, is, 98 F). This syntax is used for both representing context and forming inference rules.

Context Processing

After raw context data was sensed by a data source, it has to be processed as its customers mostly are rather interested in already interpreted and aggregated information than in raw, fine-grained data. Whereas context aggregation means the composing of context atoms either to collect all context data concerning a specific entity or to build higher-level context objects, context interpretation refers to the transformation of context data including special knowledge. These forms of context data abstraction ease the application designer's work tremendously.

The Context Toolkit offers facilities for both context aggregation and context interpretation: the context aggregators (former called context servers) are responsible

for composing context about of particular entity by subscribing to relevant widgets, context interpreters provide the possibility of transforming context, e.g. in a simple case returning the corresponding email address to a passed name. Like widgets aggregators and interpreters inherit communication methods from the upperclass BaseObject and have to be registered at the discoverer in order to be found.

The Context Managing Framework presented by Korpipää et al. (Fig. Xx) offers various processing facilities as well. Its resource servers' task is complex: First they gather raw context information by connecting to various data sources. After the preprocessing and feature abstraction crip limits and fuzzy sets are used for quantization. But now the data are delivered by posting it to the context manager's blackboard. The context recognition services are used by the context manager to create higher-level context object out of context atoms. In this vein new recognition services are easy to add.

In SOCAM the Context Reasoning Engine reasons over the knowledge base, its tasks include inferring deduced contexts, resolving context conflicts and maintaining the consistency of the context knowledge base. Different inference rules used by the reasoning engine can be specified. The interpreter is implemented with the help of Jena2 [33], a semantic web toolkit.

In CoBrA's architecture the so-called Inference Engine processes context data. The engine contains the Context Reasoning Module responsible for aggregating context information: it reasons over the Context Knowledge Base and deduces additional knowledge from information acquired from external sources.

In CASS the deriving of high-level context is also based on an inference engine and a knowledge base. The knowledge base contains rules queried by the inference engine to find goals using the so called forward chaining technique. As these rules are stored in a database separated from the interpreter whether recompiling nor restarting of components is necessary when rules change. Table 4 shows an example for a rule.

Rain	Brightness	Temperature	Goal
wet	dull	cold	Indoor

Table 4. A rule's database entry containing criteria to display rather indoor than outdoor activities in a CASS based tour-guide application.

In CORTEX the whole context processing is encapsulated in Sentient Objects: the sensory capture unit performs sensor fusion to manage uncertainty of sensor data (sensing conflicts) and to build higher-level context objects. Different contexts are represented in a so called context hierarchy together with specific actions to be undertaken in each context. Since only one context is active at any point in time (concept of the 'active context') the number of rules that have to evaluated are limited which increases efficiency of the inference process. The inference engine component is based on CLIPS (C Language Integrated Production System). It is responsible for changing application behavior according to the current context by using conditional rules.

Gaia's context processing is hidden in the Context Service Module allowing the creation of high-level context objects by performing first order logic operations such as quantification, implication, conjunction, disjunction, and negation of context predicates. One example of a rule is Context(Number of people, Room 2401, >, 4)

AND Context(Application, Powerpoint, is, Running) => Context(Social Activity, Room 2401, Is, Presentation).

Almost all current context-aware frameworks permit the aggregation and interpretation of raw context data, only exceptions leave the higher-level abstractions for the applications' layer (e.g. Hydrogen, Owl [45]).

Historical context data

Sometimes it might be necessary to have access to historical context data. Such context histories may be used to establish trends and predict future context values. As most data sources constantly provide context data, the maintaining of a context history is mainly a memory concern, so a centralized high-resource storage component is needed. Since in a server-based architecture the context data provided by sensors has to be stored at the server-side anyway to offer it to customers, the majority of these systems has the facility to query historical context data.

The Context Toolkit, CoBrA, CASS, SOCAM, CORTEX and Owl save sensed context data persistently in a database. An further advantage of using a database is the use of the Structured Query Language (SQL) which enables read and manipulation operations at a high abstraction level.

In the CoBrA and the CASS architecture the persistent storage is called Context Knowledge Base, additionally a set of APIS is offered to assert, delete, modify and query the stored knowledge.

CASS uses its database not only to save context data but also to store domain knowledge and inference rules needed for creating high-level context.

Due to limited memory resources a peer-to-peer network of mobile devices like Hydrogen is not able to offer persistent storage possibilities.

Security and Privacy

As context may include sensitive information on people, e.g. their location and their activity, it is necessary to have the possibility of protecting privacy.

For these purposes the Context Toolkit introduces the concept of context ownership. Users are assigned to sensed context data as their respective owners who are allowed to control the other user's access. New components involved in this access control are the Mediated Widgets, Owner Permissions, a modified BaseObject and Authenticators. The MediatedWidget class is an extension of a basic widget which contains a so-called widget developer specifying who owns the data being sensed. The Owner Permission is the component that receives permission queries and determines to grant or to deny access based on stored situations. These situations include authorized users, time of access etc. The modified BaseObject contains all the original methods augmented with identification mechanisms. Applications and components now have to provide their identity along with the usual request for information. Finally the Authenticator is responsible for proofing the identity using a public-key infrastructure.

Owl's security concept is based on Role Based Access Control (RBAC). As the number of users is generally smaller than the number of roles the amount of managed associations between entities and privileges is reduced. The authors remind to consider the privacy of the context sources as well as that of the context subjects.

CoBrA includes an own flexible policy language to control context access, called Rei [44]. This policy language is modeled on deontic concepts of rights, prohibitions, obligations and dispensations and controls data access through dynamically modifiable domain dependent policy rules. The example for Rei's rule syntax shown below states that all employees of 'UBMC' can perform senseAction1.

```
has(Variable, right(senseAction1,
employee(Variable,'UMBC'))))
```

4 Conclusion and future work

In this survey paper we introduced an abstract layer architecture for context-aware systems and presented various existent middleware and server-based approaches to ease the development of context-aware applications. The direct comparison of the named systems and frameworks shows their similarity concerning the layered structure. Especially remarkable is the strict division of the context data's acquisition and use. Thus context sources become reusable and are able to serve a multitude of context clients.

Although most authors refer to abstract 'context sources' the current mainly used and tested sources are physical sensors. Virtual and logical sensors are capable of providing useful context data as well and should be more incorporated in the ongoing research. Other often disregarded aspects are security and privacy issues. These facets belong to the most important components of a context-aware system as the protection of sensitive context data must be guaranteed. Many systems totally lack security modules, others provide basic security mechanisms and only a few systems offer advanced and sufficient security options.

Probably the main problem in the presented approaches is the variety of used context encodings and ways to find and access context sources. Every system and framework uses its own format to describe context and its own communications mechanisms. We believe that standardized formats and protocols are important for the enhancements of context-aware systems to make the development of context services the focus rather than the communication between context sources and users. In our opinion Web services seem to be an appropriate solution to achieve that aim as they provide standardized methods for service description and access: The XML-based WSDL (Web Services Description Language) is suited to describe the functionality of and communication with a context service, whereas the UDDI (Universal Description, Discovery and Integration) can be used to search for and register context services. Hence we are going to continue our investigation on context-aware systems placing emphasis on service-oriented architectures, especially exploring advantageous opportunities provided by Web services.

References

[1] Bill N. Schilit and Marvin M. Theimer. Disseminating Active Map Information to Mobile Hosts. IEEE Network, 8(5). 1994

[2] Nick Ryan, Jason Pascoe and David Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. Gaffney, V., van Leusen, M., Exxon, S. (eds) Computer Applications in Archaeology 1997

[3] Anind K. Dey. Context-aware computing: The CyberDesk project. Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments. Menlo Park, CA: AAAI Press.

[4] Richard Hull, Philip Neaves, James Bedford-Roberts. Towards situated computing. In Proceedings of International Symposium on Wearable Computers 1997

[5] Peter J. Brown. The Stick-e Document: a framework for creating context-aware applications. In Electronic Publishing, Palo Alto, 1996

[6] Anind K. Dey, Gregory D. Abowd. Towards a better understanding of context and contextawareness. Proceedings of the Workshop on the What, Who, Where, When and How of Context-Awareness, affiliated with the CHI 2000 Conference on Human Factors in Computer Systems, New York, NY: ACM Press. 2000

[7] Paul Prekop, Mark Burnett. Activities, context and ubiquitous computing. Special Issue on Ubiquitous Computing Computer Communications, vol.26, no.11, 2003

[8] Richard Moe Gustavsen. Condor – An Application Framework for mobility-based context-aware Applications. 2002

[9] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann. Context-Awareness on Mobile Devices – the Hydrogen Approach. 2002

[10] Jay Budzik, Kristian J. Hammond. User interactions with everyday applications as context for just-in-time information access. Proceedings of Intelligent User Interfaces 2000. ACM Press, 2000

[11] Lev Finkelstein, Evgeniy Gabrilovich1, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman and Eytan Ruppin. Placing Search In Context: the Concept Revisited. Tenth International World Wide Web Conference WWW10, Hong Kong, 2001

[12] Anind K. Dey, Gregory D. Abowd. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction (HCI) Journal, Volume 16 (2-4)

[13] Harry Chen. An Intelligent Broker Architecture for Context-Aware Systems. PhD. Dissertation proposal. 2003

[14] Heikki Ailisto, Petteri Alahuhta, Ville Haataja, Vesa Kyllönen, Mikko Lindholm. Structuring Context Aware Applications: Five-Layer Model and Example Case. 2002

[15] Jadwiga Indulska, Peter Sutton. Location Management in Pervasive Systems. In Conferences in Research and Pratice in Information Technology series, Vol. 21. 2003

[16] Albrecht Schmidt, Kristof Van Laerhoven. How to Build Smart Appliances? IEEE Personal Communications 8(4). 2001

[17] Terry Winograd. Architectures for Context. Human-Computer-Interaction, vol.16, nos. 2-4) 2001

[18] Anand Ranganathan, Roy H. Campbell. An infrastructure for context-awareness based on first order logic. Personal and Ubiquitous Computing 7 2003

[19] Panu Korpipää, Jani Mäntyjärvi, Juha Kela, Heikki Keränen, Esko-Juhani Malm. Managing Context Information in Mobile Devices. IEEE Pervasive Computing. 2003

[20] Panu Korpipää, Jani Mäntyjärvi. An Ontology for Mobile Device Sensor-Based Context Awareness. Proc. Context '03, LNAI no. 2680, 2003
[21] Resource Description Framework (RDF). www.w3.org/RDF/
[22] OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/
[23] Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, Mike Pinkerton. Cyberguide: A mobile context-aware tour guide. Wireless Networks, 3(5). 1997
[24] Andy Harter, Andy Hopper, Pete Steggles, AndyWard, Paul Webster. The Anatomy of a Context-Aware Application. Wireless Networks, 8(2/3) 2002
[25] Fredrik Espinoza, Per Persson, Anna Sandin, Hanna Nyström, Elenor Cacciatore, Markus Bylund. GeoNotes: Social and Navigational Aspects of Location-Based Information Systems. SICS Technical Report T2001:08. 2001
[26] Nissanka B. Priyantha, Anit Chakraborty, Hari Balakrishnan. The Cricket Location-Support System. In Proceedings of the Sixth ACM Annual International Conference on Mobile Computing and Networking (Boston, MA). 2000
[27] Jenna Burrell, Geri K. Gay. E-graffiti: evaluating real-world use of a context-aware system. Interacting with Computers (Special Issue on Universal Usability) 14(4). 2002
[28] Clemens Kerer, Schahram Dustdar, Mehdi Jazayeri, Danilo Gomez, Akos Szego, Jose A. Burgos Caja. Presence-Aware Infrastructure using Web services and RFID technologies
[29] Miguel A. Muñoz, Marcela Rodríguez, Jesus Favela, Ana I. Martinez-Garcia, Victor M. González. Context-Aware Mobile Communication in Hospitals. 2003
[30] Daniel Salber, Anind K. Dey, Gregory D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In Proceedings of ACM CHI 99, Pittsburgh, PA. 1999
[31] Gregory Biegel, Vinny Cahill. A Framework for Developing Mobile, Context-aware Applications. In Proceedings of 2nd IEEE conference on Pervasive computing and Communications, Percom 2004
[32] Tao Gu, Xiao Hang Wang, Hung Keng Pung, Da Qing Zhang. A Middleware for Context-Aware Mobile Services. IEEE Vehicular Technology Conference. Milan, Italy, 2004
[33] Jena – A Semantic Web Framework for Java. http://jena.sourceforge.net.
[34] Harry Chen, Tim Finin, Anupam Joshi. Using OWL in a Pervasive Computing Broker. Workshop on Ontologies in Agent Systems, AAMAS 2003
[35] Harry Chen, Tim Finin, Anupam Joshi. An Ontology for Context-Aware Pervasive Computing Environments. Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review. 2004
[36] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganat, Roy H. Campbell, Klara Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. In IEEE Pervasive Computing, Oct-Dec 2002.
[37] Gaia Homepage. http://choices.cs.uiuc.edu/gaia.
[38] Mark Weiser. The Computer for the Twenty-First Century. Scientific American. 1991
[39] Roy Want, Andy Hopper, Veronica Falcão, Jonathan Gibbons. The Active Badge Location System. ACM Transactions on Information Systems, 10(1), 1992
[40] Sidney Fels, Yasuyuki Sumi, Tameyuki Etani, Nicolas Simonet, Kaoru Kobayashi, Kenji Mase. Building a context-aware mobile assistant for exhibition tours. The First Kyoto Meeting on Social Interaction and Communityware. 1998
[41] Nigel Davies, Keith Cheverst, Keith Mitchell, Alon Efrat. Developing a context sensitive tour guide. Proceedings of First Workshop on Human-Computer Interaction for Mobile Devices, Glasgow, UK. 1998
[42] Jason I. Hong and James A. Landay. An infrastructure approach to context-aware computing. Human-Computer Interaction, Vol. 16, 2001

[43] Patrick Fahy, Siobhan Clarke. CASS – Middleware for Mobile Context-Aware Applications. MobiSys 2004
[44] Lalana Kagal, Tim Finin, Anupam Joshi. A Policy Language for a Pervasive Computing Environment. IEEE 4th International Workshop on Policies for Distributed Systems and Networks, 2003.
[45] Maria R. Ebling, Guerney D. H. Hunt, Hui Lei. Issues for Context Services for Pervasive Computing. Workshop on Middleware for Mobile Computing, 2001