A Survey on Context-Aware Systems

Matthias Baldauf, Schahram Dustdar* and Florian Rosenberg
Distributed Systems Group
Information Systems Institute
Vienna University of Technology
Argentinierstrasse 8/184-1
1040 Vienna, Austria
E-mail: baldauf@par.univie.ac.at
E-mail: dustdar@infosys.tuwien.ac.at
E-mail: rosenberg@infosys.tuwien.ac.at
*Corresponding author

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 these mechanisms are of high value and are used to increase usability tremendously. In this paper, we present common architecture principles of context-aware systems and derive a layered conceptual design framework to explain the different elements common to most context-aware architectures. Based on these design principles, 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, context framework, context middleware, sensors, context model, context ontology, context-aware services

Reference to this paper should be made as follows: Baldauf, M., Dustdar, S., Rosenberg, F. (xxxx) 'A Survey on Context-Aware Systems', *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Mathias Baldauf is a PhD student at the Vienna University of Technology and a research assistant at the Institute of Scientific Computing, University of Vienna.

Schahram Dustdar is a Full Professor of Computer Science with a focus on Internet Technologies at the Distributed Systems Group, Information Systems Institute, Vienna University of Technology (TU Wien). In 1999 he co-founded Caramba Labs Software AG (CarambaLabs.com) in Vienna, a venture capital co-funded software company focused on software for collaborative processes in teams. Caramba Labs was nominated for several (international and national) awards. He has published some 100 scientific papers as conference-, journal-, and book contributions. He has written three academic books, one professional book, and co-edited six books/proceedings. More information can be found at: http://www.infosys.tuwien.ac.at/Staff/sd.

Florian Rosenberg is research assistant and PhD student at the Distributed Systems Group, Information Systems Institute, Vienna University of Technology. His research areas include context-aware and autonomic services, service-oriented architectures and Web service technologies. More information can be found at: http://www.infosys.tuwien.ac.at/Staff/rosenberg.

1 INTRODUCTION

With the appearance and penetration of mobile devices such as notebooks, PDAs, and smart phones, pervasive (or ubiquitous) systems are becoming increasingly popular these days. The term "pervasive" introduced first by Mark Weiser in 1991 (Weiser, 1991) refers to the seamless integration of devices into the users everyday life. Appliances should vanish into the background to make the user and his tasks the central focus rather than computing devices and technical issues. One field 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. Particularly when it comes to 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 according to the changing circumstances as context data may change rapidly. The needed context information may be retrieved in a variety of ways, such as applying sensors, network information, device status, browsing user profiles and using other sources. The history of context-aware systems started when Want et al. (1992) introduced their *Active Badge Location System* 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 (Abowd et al., 1997; Sumi et al., 1998; Cheverst et al., 2000) emerged which provided information according to the user's current location. While location information is by far the most frequently used attribute of context, attempts to use other context information as well have grown over the last few years as the examples in this paper will 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 literature the term *context-aware* appeared in Schilit and Theimer (1994) the first time. There the authors describe context as location, identities of nearby people, objects and changes to those objects. Such enumerations of context examples were often used in the beginning of context-aware systems research. Ryan et al. (1997) referred to context as the user's location, environment, identity and time. Dey (1998) defines context as the user's emotional state, focus of attention, location and orientation, date and time, as well as objects and people in the user's environment. Another common way of defining context was the use of synonyms. Hull et al. (1997) describe context as the aspects of the current situation. These kind of definitions are often too wide. However, a good one can be found in Brown (1996). Brown defines context to be the elements of the user's environment which the computer knows about. One of the most accurate definitions is given by Dey and Abowd (2000b). These 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. Prekop and Burnett (2003) and Gustavsen (2002) call these dimensions *external* and *internal*, and Hofer et al. (2002) refer to *physical* and *logical* context. The external (physical) dimension refers to context that can be measured by hardware sensors, i.e., location, light, sound, movement, touch, temperature or air pressure, whereas the internal (logical) dimension is mostly specified by the user or captured by monitoring user interactions, i.e., the user's goals, tasks, work context, business processes, the user's emotional state. Most context-aware systems make use of external context factors as they provide useful data, such as location information. Furthermore, external attributes are easy to sense by using 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* (Budzik and Hammond, 2000) and the *IntelliZap Project* (Finkelstein et al., 2001) which support the user by providing relevant information due to information read out of opened Web pages, documents, etc. When dealing with context, three entities can be distinguished (Dey and Abowd, 2001): *places* (rooms, buildings etc.), *people* (individuals, groups) and *things* (physical objects, computer components etc.). Each of these entities may be described by various attributes which can be classified into four categories: *identity* (each entity has a 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.).

This paper is structured as follows. Section 2 introduces current design principles for context-aware systems and common context models used in various context-aware systems. In Section 3, we present a comparison of existent context-aware systems and explain approaches, varieties and similarities. In Section 4, we discuss the presented approaches, and highlight advantages and disadvantages. Finally, Section 5 draws some concluding remarks and presents some future work in this area.

2 DESIGN PRINCIPLES

In this section, we describe basic design principles and introduce a conceptually layered framework, to associate the functionality implemented in existent frameworks to various layers. Furthermore, we depict different context models used for representing, storing and exchanging contextual information.

2.1 Architecture

Context-aware systems can be implemented in many ways. The approach depends on special requirements and conditions such as 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. Furthermore, the method of context-data acquisition is very important when designing context-aware systems because it predefines the architectural style of the system at least to some extent. Chen (2004) presentes three different approaches on how to acquire contextual information:

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 tightly coupled method is usable only in rare cases. Therefore, it is not suited for distributed systems due to its direct access nature which lacks a component capable of managing multiple concurrent sensor accesses.

Middleware infrastructure. 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. Besides 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. 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 (2001) 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 (Dey and Abowd, 2000a, 2001). Widgets 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 tightly coupled widget approach increases efficiency but is not robust to component failures.

Networked services. This more flexible approach, argued for example in Hong and Landay (2001), 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. One drawback is the need of a centralized server to host the blackboard and the lack in communication efficiency as two hops per communication are needed.

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

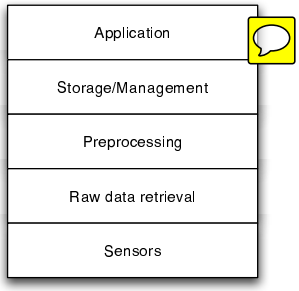

Figure 1: Layered conceptual framework for context-aware systems

As mentioned above, a separation of detecting and using context is necessary to improve extensibility and reusabil-

ity of systems. The following layered conceptual architecture, as depicted in Figure 1, augments layers for detecting and using context by adding interpreting and reasoning functionality (Ailisto et al., 2002; Dey and Abowd, 2001).

The first layer consists of a collection of different sensors. It is notable that the word "sensor" not only refers to sensing hardware but also to every data source which may provide usable context information. Concerning the way data is captured, sensors can be classified in three groups (Indulska and Sutton, 2003).

- *Physical sensors.* The most frequently used type of sensors are physical sensors. Many hardware sensors are available nowadays which are capable of capturing almost any physical data. Table 1 shows some examples of physical sensors (Schmidt and van Laerhoven, 2001).

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: Commonly used physical sensor types

- *Virtual sensors.* Virtual sensors source context data from software applications or services. For example, 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, and combine physical and virtual sensors with additional information from databases or various other sources in order to solve higher tasks. For example, a logical sensor can be constructed to detect an employee's current position by analyzing logins at desktop PCs and a database mapping of devices to location information.

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 is often implemented in reusable software components which make low-level details of hardware access transparent by providing more abstract methods such as `getPosition()`. By using interfaces for components responsible for equal types of context these components become exchangeable. Therefore, it is possible, for instance, to replace a RFID system by a GPS system without any major modification in the current and upper layers.

The *Preprocessing* layer is not implemented in every context-aware system but may offer useful information if the raw data are too coarse grained. The preprocessing layer is responsible for reasoning and interpreting contextual information. 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. For example, 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, may be.

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 "composition". A single sensor value is often not important to an application, whereas combined information might be more precious and accurate. 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. To make this analysis work correctly a multitude 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 better be encapsulated and moved to the context server. The encapsulation advances the reusability and, hence, eases the development of client applications. And by making such aggregators remotely 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 to be solved in this layer as well. For example, when a system is notified about a person's location by the coordinates of her mobile phone and by 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, *Storage and Management*, organizes the gathered data and offers them via a public interface to the client. Clients may gain access in two different ways,

synchronous and *asynchronous*. In the synchronous manner the client is polling the server for changes via remote method calls. Therefore, 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. Each client subscribes to specific events it is interested in. On occurrence of one of these events, 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 (Chen, 2004). 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.

All the systems, we analyzed in this paper implement most of the layers of the conceptual framework presented above.

2.2 Context Models

A context model is needed to define and store context data in a machine processable form. To develop flexible and useable context ontologies that cover the wide range of possible contexts is a challenging task. Strang and Linnhoff-Popien (2004) summarized the most relevant context modeling approaches which are based on the data structures used for representing and exchanging contextual information in the respective system:

Key-Value Models. These models represent the simplest data structure for context modeling. They are frequently used in various service frameworks, where the key-value pairs are used to describe the capabilities of a service. Service discovery is then applied by using matching algorithms which use these key-value pairs.

Markup Scheme Models. All markup based models use a hierarchical data structure consisting of markup tags with attributes and content. *Profiles* represent typical markup-scheme models. Typical examples for such profiles are the Composite Capabilities / Preference Profile (CC/PP) (W3C, 2004a) and User Agent Profile (UAProf) (Wapforum, 2001), which are encoded in RDF/S. Various other examples can be found in Strang and Linnhoff-Popien (2004).

Graphical Models. The Unified Modeling Language (UML) is also suitable for modeling context. Various approaches exist where contextual aspects are modeled in by using UML, e.g., Sheng and Benatallah (2005). Another modeling approach includes an extension to the ORM (Object-Role Modeling) by context information presented in Hendricksen et al. (2003).

Object Oriented Models. Modeling context by using object-oriented techniques offers to use the full power of object orientation (e.g., encapsulation, reusability, inheritance). Existing approaches use various objects to represent different context types (such as temperature, location, etc.), and encapsulate the details of context processing and representation. Access the context and the context processing logic is provided by well-defined interfaces. Hydrogen (Hofer et al., 2002) uses such an object-oriented example. We explain the system in more detail in Section 3.

Logic Based Models. Logic-based models have a high degree of formality. Typically, facts, expressions and rules are used to define a context model. A logic based system is then used to manage the aforementioned terms and allows to add, update or remove new facts. The inference (also called reasoning) process can be used to derive new facts based on existing rules in the systems. The contextual information needs to be represented in a formal way as facts. One of the first approaches was published by McCarthy and Buvač (1997).

Ontology Based Models. Ontologies represent a description of the concepts and relationships. Therefore, ontologies are a very promising instrument for modeling contextual information due to their high and formal expressiveness and the possibilities for applying ontology reasoning techniques. Various context-aware frameworks use ontologies as underlying context models. We describe some of them in Section 3.

The conclusion of the evaluation presented in Strang and Linnhoff-Popien (2004), based on six requirements, show that ontologies are the most expressive models and fulfill most of their requirements. Korpipää et al. (2003) present some requirements and goals having designed a context ontology:

- *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.

Numerous tools are 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 (W3C, 2000) and the Web Ontology Language OWL (W3C, 2004b). 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 such temperature, time, speed, etc. This type information may be used as 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 for creating well-structured type names is the use of cascaded names (Korpipää and Mäntyjärvi, 2003) 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:

- *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 a 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 depending 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. Table 2 shows a small part of an example vocabulary from Korpipää and Mäntyjärvi (2003). 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**
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

3 EXISTENT SYSTEMS AND FRAMEWORKS

In this section, we discuss three types of existing context-aware systems. Firstly, we briefly describe a special case of context-aware systems, the so-called location-aware systems. Secondly, we describe an existing context-aware system assisting hospital information management. Thirdly, we draw our main focus on context-aware frameworks, as basic building block for context-aware systems and applications.

3.1 Location-Aware Systems

Context-aware systems dealing with location information are widespread and the demand for them is growing due to the increasing spread of mobile devices. 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 (Espinoza et al., 2001; Priyantha et al., 2000; Burrell and Gay, 2002; Kerer et al., 2004). A couple of different location aware infrastructures are available in order to collect position data. These include GPS satellites, mobile phone towers, badge proximity detectors, cameras, magnetic card readers, barcode readers, etc. These sensors can provide either position or proximity information. In addition, they differ in price and accuracy. Some need a clear line of sight, other signals can travel through walls, etc. As a detailed example, we introduce an indoor location sensing system from Harter et al. (2002), who describes a location-aware system using ultrasonic technique. To each entity (person or equipment) which 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 hardware type needed is a *base station*. It periodically sends radio messages with specific bat ids and resets the receivers. A corresponding bat reacts by 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 by using multilateration (an extension of trilateration). A challenge the authors where confronted with due to the use of ultrasonic tech-

nique was the incorrect measurement because of unwanted reflections of the signals. The problem was 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 such as 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 an example for this kind of context-aware infrastructures serves the system presented by Munõz et al. (2003) 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 agents' 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., in hospital scenarios. These systems can be optimized for the situations they are used in. They do not have to be flexible and extensible. To actually simplify the development of context-aware applications an abstract framework is needed. Such a generic infrastructure not only provides a client with 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 criteria (architecture, resource discovery, sensing, context model, context processing historical context data, security and privacy).

Architectures

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 robustness. The architecture of the *Context Managing Framework* presented by Korpipää et al. (2003) 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*.

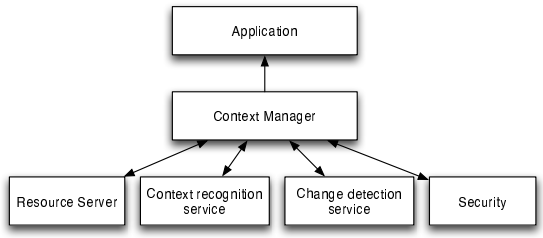

Figure 2: *Context Managing Framework* Architecture

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 provides this information to the client applications.

The *SOCAM* (Service-oriented Context-Aware Middleware) project introduced by Gu et al. (2004a,b) 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, thus, they make use of the different levels of context and adapt their behavior according to the current context.

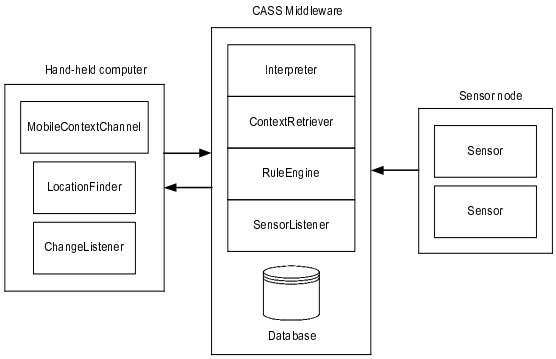

Figure 3: Architecture of the CASS system

One further extensible centralized middleware approach designed for context-aware mobile applications is a project called *CASS* (Context-awareness sub-structure) presented in Fahy and Clarke (2004). In Figure 3, the system architecture of *CASS* is presented. 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 con-

text data. Both of these classes may use the services of an interpreter. The `ChangeListener` is a component with communication 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 is supported on the client side.

CoBrA (Context Broker Architecture) (Chen et al., 2003) 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 peoples and object 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* (Salber et al., 1999; Dey and Abowd, 2000a; Dey, 2001), 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 toolkits object-oriented API provides a superclass called `BaseObject` which offers generic communication abilities to ease the creation of own components.

Another framework based on a layered architecture is built in the *Hydrogen* project (Hofer et al., 2002). Its context acquisition approach is specialized for mobile devices. While the availability of a centralized component is essential in the majority of existent distributed content-aware systems, the *Hydrogen* system tries to avoid this dependency. It distinguishes between a *remote* and a *local context*. The remote context is information another device knows about, the 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

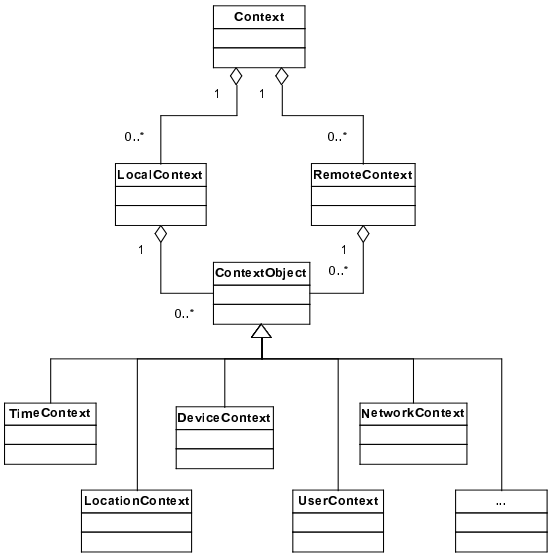

Figure 4: Hydrogen's Object-Oriented Approach

types, e.g., `LocationContext`, `DeviceContext`, etc. This approach allows the simple addition of new context types by specializing `ContextObject`. A context type has to implement the methods `toXML()` and `fromXML()` from the `ContextObject` class in order to convert the data from and to a XML stream.

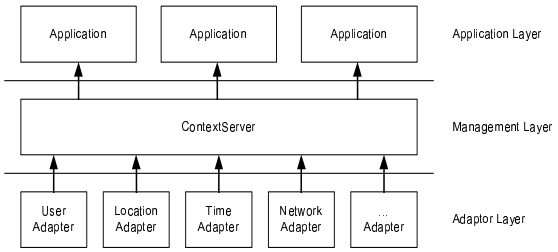

Figure 5: Architecture of the Hydrogen project

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 the 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.

The *CORTEX* system is an example for a context-aware middleware approach. The architecture is based on the *Sentient Object Model* (Biegel and Cahill, 2004) 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.

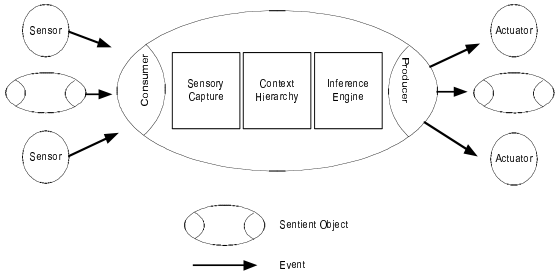

Figure 6: The Sentient Object Model

A sentient object is an encapsulated entity consisting of three main parts, as depicted in Figure 6, *Sensory capture*, *Context hierarchy* and *Inference engine*. Via interfaces a sentient object 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 (Roman et al., 2002; Gaia Project, 2005), 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 it 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 as depicted in Figure 7.

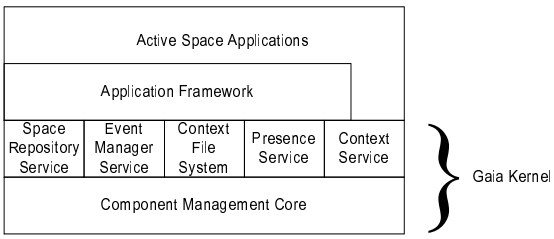

Figure 7: Architecture of the Gaia system

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. 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 components 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.

SOCAM 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 such as *Hydrogen* only uses local built-in sensors and does not connect to distributed sensors, therefore, no discovery mechanism is involved.

Sensing

The *Context Toolkit* 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. Furthermore, 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 such as 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 widgets 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* and *CoBrA* makes use of *context acquisition components*.

Context Model

An 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*. The *SOCAM* 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 (Chen et al., 2003, 2004). Listing 1 shows parts of an COBRA-Ont example. The ontology structure and vocabulary applied in the Context Managing Toolkit are described in RDF. Parts of its vocabulary are used as an example in Table 2 in Section 2.

```
<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>
```

Listing 1: COBRA-ONT Example

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 for forming inference rules.

Context Processing

As soon as the raw context data is 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 refers to the composition 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 of particular entities 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 Korpip et al. (Figure 2) offers various processing facilities as well. The resource servers' tasks are 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 by using Jena2 (Jena, 2005), a semantic Web toolkit.

In *CoBrA* 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* 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 neither recompiling nor restarting of components is necessary when rules change. Table 3 shows an example rule database entry containing criteria to display rather indoor than outdoor activities in a *CASS* based tour-guide application.

Rain	Brightness	Temperature	Goal
wet	dull	cold	indoor

Table 3: Example rule database entry

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 be evaluated are limited. Thus efficiency of the inference process is increased. 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)` $\Longrightarrow$ `Context(Social Activity, Room 2401, Is, Presentation)`. Almost all current context-aware frameworks permit the aggregation and interpretation of raw context data. Only in a few cases the higher-level abstractions are handled by the application layer, such as in *Hydrogen*.

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 maintenance 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 anyway to be stored at the server-side to offer it to customers, the majority of these systems has the facility to query historical context data. The *Context Toolkit*, *CoBrA*, *CASS*, *SOCAM* and *CORTEX* save sensed context data persistently in a database. A further advantage of using a database is the use of the Structured Query Language (SQL) which enables to read and to manipulate 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 opportunity to protect privacy. For these purposes the *Context Toolkit* introduces the concept of context ownership. Users are assigned to sensed context data as their respective owners. They are allowed to control the other users' 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. Now applications and components have to provide their identity along with the usual request for information. Finally the authenticator is responsible for proofing the identity by using a public-key infrastructure. *CoBrA* includes an own flexible policy language to control context access, called *Rei* (Kagal et al., 2003). 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.

4 DISCUSSION OF APPROACHES

In Table 4 we have summarized the main aspects of the discussed approaches. The architectural style of a context-aware system is mainly driven by the context acquisition method. The main criteria for a reasonable architectural approach is the separation of concerns between the context acquisition and the user components as proposed by Dey (2000). All the frameworks presented in this paper support this separation of concerns.

The sensing technology is implemented differently in every framework. It is important, that the concrete sens-

	Architecture	Sensing	Context model	Context processing	Resource discovery	Historical context data	Security and privacy
CASS	centralized middleware	sensor nodes	relational data model	inference engine and knowledge base	n.a.	available	n.a.
CoBra	agent based	context acquisition module	ontologies (OWL)	inference engine and knowledge base	n.a.	available	Rei policy language
Context Management Framework	blackboard based	resource servers	ontologies (RDF)	context recognition service	resource servers + subscription mechanism	n.a.	n.a.
Context Toolkit	widget based	context widgets	attribute-value tuples	context interpretation and aggregation	discoverer component	available	context ownership
CORTEX	sentient object model	context component framework	relational data model	service discovery framework	resource management component framework	available	n.a.
Gaia	MVC (extended)	context providers	4-ary predicates (DAML + OIL)	context-service module (first-order logic)	discovery service	available	supported (e.g., secure tracking, location privacy, access control)
Hydrogen	three layered architecture	adapters for various context types	object-oriented	interpretation and aggregation of raw data only	n.a.	n.a.	n.a.
SOCAM	distributed with centralized server	context providers	ontologies (OWL)	context reasoning engine	service locating service	available	n.a.

Table 4: Summary of discussed approaches

ing mechanism is encapsulated in separate components, to support the aforementioned separation of concerns. Furthermore, it encapsulates sensing and allows to access the contextual data via defined interfaces. Currently, there is no standard description language or ontology for sensing contextual information from various sources to enable reuse across various middleware systems and frameworks. Therefore, proprietary solutions as used by the different frameworks, have emerged. *SOCAM* use the most sophisticated approach for sensing context information. External virtual sensors are consumed via Web services (by using SOAP). Internal providers for querying sensors are consumed by using context events represented in OWL based on a predefined ontology.

The context model and the context processing logic supported by different frameworks is a major criteria for providing intelligent and adaptable context-aware services or applications. As mentioned before, ontologies provide a rich formalism for specifying contextual information. Based on such ontological models, highly sophisticated ontology reasoning engines can derive new concepts to adapt the service behavior accordingly. The major drawback of the *Context Toolkit* is, therefore, its context model, a set of attribute-value tuples. The development of intelligent

context processing and aggregation is limited due to the fact that such attributes do not have a meaning. Furthermore, using non-ontology based models requires a lot of programming effort and tightly couples the context model to the rest of the system. Moreover, the lack of declarative semantics of programs does not allow reasoning and knowledge sharing amount systems. For example, *SOCAM* uses a general upper ontology to specify basic common contextual properties and to refine this general ontology. Domain-specific ontologies can be defined to provide very fine grained possibilities for specifying and formalizing context.

Resource discovery mechanisms are currently rarely used in the presented frameworks. Such dynamic mechanisms are important, especially in a pervasive environment, where available sensors, the context sources, change rapidly (new ones are added or removed). *SOCAM*, for example, which is the only system that is based on a service-oriented architecture, offers a *service locating service*, which dynamically binds to available context providers. These providers can also be changed at runtime. The lack of resource discovery support is a major drawback of many frameworks, because it implies that the used context sources are stable and permanently available, which is not always the case in real-world applications. Erroneous behavior of one or more context sources may lead to a decreased availability of the context-aware service or application.

Managing historical context data provides the ability to implement intelligent learning algorithms which allow to provide highly adaptable context-aware services. Furthermore, based on learning algorithms, contextual information can be predicted to proactively provide a certain set of services to the user. Many of the systems store contextual information but non of them do not use learning techniques to provide context-aware service proactively.

Another important aspect is security and privacy. Contextual information mostly considers user profile information and other sensitive information. Therefore, concepts are needed to express policies and to define ownership of context information. *CoBra* uses the Rei policy language, to express (security) policies about contextual information. *Gaia* uses several mechanisms to define privacy restrictions and secure communication for tracking locations of people. The *Context Toolkit* implements the concept of context ownership, which is used to allow access to the context to the owner only. The other frameworks do not use security concepts which is one of their major drawback. When dealing with sensitive data, secure connections for context acquisition as well as privacy of different user specific contextual information is very important.

5 CONCLUSIONS AND FUTURE WORK

In this survey paper we described different design principles and context models 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 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 currently 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 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 service technologies seem to be an appropriate solution to achieve that aim as they provide standardized methods for service description and access.

Our future work in this area will investigate the use of service-oriented and autonomic computing concepts for building context-aware service frameworks. We believe that standardized technologies and protocols, such as WSDL and SOAP, could help to build more interoperable context aware services. Furthermore, the use of ontologies for building a context model is an important approach (as can be seen from existing approaches) to build more sophisticated algorithms which derive new contextual knowledge or patterns to proactively aggregate new context-aware services. Autonomously orchestrating atomic context-aware services into higher level services based on context information and available QoS parameters provides high potential for offering more accurate services to the user.

REFERENCES

Abowd, G. D., Atkeson, C. G., Hong, J., Long, S., Kooper, R., and Pinkerton, M. (1997). Cyberguide: A mobile context-aware tour guide. *Wirless Networks*, 3(5).

Ailisto, H., Alahuhta, P., Haataja, V., Kylloenen, V., and Lindholm, M. (2002). Structuring context aware applications: Five-layer model and example case. In *Proceedings of the Workshop on Concepts and Models for Ubiquitous Computing, Goteborg, Sweden.*

Biegel, G. and Cahill, V. (2004). A framework for develop-

ing mobile, context-aware applications. In *Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communication*.

Brown, P. J. (1996). The stick-e document: A framework for creating context-aware applications. In *Proceedings of the Electronic Publishing, Palo Alto*, pages 259–272.

Budzik, J. and Hammond, K. J. (2000). User interactions with everyday applications as context for just-in-time information access. In *Proceedings of Intelligent User Interfaces*. ACM Press.

Burrell, J. and Gay, G. (2002). E-graffiti: evaluating real-world use of a context-aware system. *Interacting with Computers – Special Issue on Universal Usability*, 14(4):301–312.

Chen, H. (2004). *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, University of Maryland, Baltimore County.

Chen, H., Finin, T., and Joshi, A. (2003). An ontology for context-aware pervasive computing environments. In *Proceedings of the Workshop on Ontologies in Agent Systems (AAMAS)*.

Chen, H., Finin, T., and Joshi, A. (2004). An ontology for context-aware pervasive computing environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 18(3):197–207.

Cheverst, K., Davies, N., Mitchell, K., Friday, A., and Efstratiou, C. (2000). Developing a context-aware electronic tourist guide: some issues and experiences. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 17–24, New York, NY, USA. ACM Press.

Dey, A. K. (1998). Context-aware computing: The CyberDesk project. In *Proceedings of the AAAI, Spring Symposium on Intelligent Environments, Menlo Park, CA*.

Dey, A. K. (2000). *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology.

Dey, A. K. (2001). The Context Toolkit – a toolkit for context-aware applications. `http://www.cs.berkeley.edu/~dey/context.html`.

Dey, A. K. and Abowd, G. D. (2000a). The Context Toolkit: Aiding the development of context-aware applications. In *Workshop on Software Engineering for Wearable and Pervasive Computing, Limerick, Ireland*.

Dey, A. K. and Abowd, G. D. (2000b). Towards a better understanding of context and context-awareness. In *Proceedings of the Workshop on the What, Who, Where, When and How of Context-Awareness, New York*. ACM Press.

Dey, A. K. and Abowd, G. D. (2001). A conceptual framework and a toolkit for supporting rapid prototyping of context-aware applications. *Human-Computer Interactions (HCI) Journal*, 16(2-4):7–166.

Espinoza, F., Persson, P., Sandin, A., Nyström, H., Cacciatore, E., and Bylund, M. (2001). GeoNotes: Social and navigational aspects of location-based information systems. In *Proceedings of the 3rd International Conference on Ubiquitous Computing, Atlanta, Georgia, USA*, pages 2–17.

Fahy, P. and Clarke, S. (2004). CASS – a middleware for mobile context-aware applications. In *Workshop on Context Awareness, MobiSys 2004*.

Finkelstein, L., Gabriolovic, E., Matias, Y., Rivilin, E., Solan, Z., Wolfman, G., and Ruppin, E. (2001). Placing search in context: The concept revisited. In *Proceedings of the 10th International World Wide Web Conference (WWW10), Hong Kong*.

Gaia Project (2005). Gaia project. http://gaia.cs.uiuc.edu/.

Gu, T., Pung, H. K., and Zhang, D. Q. (2004a). A middleware for building context-aware mobile services. In *Proceedings of IEEE Vehicular Technology Conference (VTC), Milan, Italy*.

Gu, T., Pung, H. K., and Zhang, D. Q. (2004b). A middleware for building context-aware mobile services. In *Proceedings of IEEE Vehicular Technology Conference (VTC 2004), Milan, Italy*.

Gustavsen, R. M. (2002). Condor – an application framework for mobility-based context-aware applications. In *Proceedings of the Workshop on Concepts and Models for Ubiquitous Computing, Goeteborg, Sweden*.

Harter, A., Hopper, A., Steggles, P., Ward, A., and Webster, P. (2002). The anatomy of a context-aware application. *Wirless Networks*, 8(2-3).

Hendricksen, K., Indulska, J., and Rakotonirainy, A. (2003). Generating context management infrastructure from high-level context models. In *Proceedings of the 4th International Conference on Mobile Data Management (MDM'03)*, pages 1–6.

Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., and Altmann, J. (2002). Context-awareness on mobile devices – the hydrogen approach. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, pages 292–302.

Hong, J. I. and Landay, J. A. (2001). An infrastructure for context-aware computing. *Human-Computer Interaction*, 16.

Hull, R., Neaves, P., and Bedford-Roberts, J. (1997). Towards situated computing. In *Proceedings of the International Symposium on Wearable Computers*.

Indulska, J. and Sutton, P. (2003). Location management in pervasive systems. In *CRPITS '03: Proceedings of the Australasian Information Security Workshop*, pages 143–151.

Jena (2005). A semantic web framework for java. http://jena.sourceforge.net/.

Kagal, L., Finin, T., and Joshi, A. (2003). A policy language for a pervasive computing environment. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY), 2003*, pages 63–74.

Kerer, C., Dustdar, S., Jazayeri, M., Gomes, D., Szego, A., and Caja, J. A. B. (2004). Presence-aware infrastructure using web services and RFID technologies. In *Proceedings of the 2nd European Workshop on Object Orientation and Web Services, Oslo, Norway*.

Korpipää, P. and Mäntyjärvi, J. (2003). An ontology for mobile device sensor-based context awareness. In *Proceedings of CONTEXT, 2003*, volume 2680 of *Lecture Notes in Computer Science*, pages 451–458.

Korpipää, P., Mäntyjärvi, J., Kela, J., Keränen, H., and Malm, E.-J. (2003). Managing context information in mobile devices. *IEEE Pervasive Computing*.

McCarthy, J. and Buvač (1997). Formalizing context (expanded notes). In Buvač, S. and Iwańska, Ł., editors, *Working Papers of the AAAI Fall Symposium on Context in Knowledge Representation and Natural Language*, pages 99–135, Menlo Park, California. American Association for Artificial Intelligence.

Munõz, M. A., Gonzalez, V. M., Rodriguez, M., and Favela, J. (2003). Supporting context-aware collaboration in a hospital: an ethnographic informed design. In *Proceedings of Workshop on Artificial Intelligence, Information Access, and Mobile Computing 9th International Workshop on Groupware, CRIWG 2003, Grenoble, France*, pages 330–334.

Prekop, P. and Burnett, M. (2003). Activities, context and ubiquitous computing. *Special Issue on Ubiquitous Compuitng Computer Communications*, 26(11).

Priyantha, N. B., Chakraborty, A., and Balakrishnan, H. (2000). The cricket location-support system. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 32–43. ACM Press.

Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H., and Nahrstedt, K. (2002). A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83.

Ryan, N., Pascoe, J., and Morse, D. (1997). Enhanced reality fieldwork: The context-aware archaeological assistent. *Computer Applications in Archaeology*.

Salber, D., Dey, A. K., and Abowd, G. D. (1999). The Context Toolkit: Aiding the development of context-aware applications. In *Proceedings of the ACM CHI, Pittsburgh, PA*.

Schilit, B. and Theimer, M. (1994). Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22–32.

Schmidt, A. and van Laerhoven, K. (2001). How to build smart applications? *IEEE Personal Communications*, 8(4).

Sheng, Q. Z. and Benatallah, B. (2005). ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services. In Society, I. C., editor, *The 4th International Conference on Mobile Business (ICMB'05)*.

Strang, T. and Linnhoff-Popien, C. (2004). A context modeling survey. In *First International Workshop on Advanced Context Modelling, Reasoning And Management, UbiComp 2004*.

Sumi, Y., Etani, T., Fels, S., Simonet, N., Kobayashi, K., and Mase, K. (1998). C-map: Building a context-aware mobile assistant for exhibition tours. In *Community Computing and Support Systems, Social Interaction in Networked Communities [the book is based on the Kyoto Meeting on Social Interaction and Communityware, held in Kyoto, Japan, in June 1998]*, pages 137–154, London, UK. Springer-Verlag.

W3C (2000). Resource Description Framework (RDF). `http://www.w3.org/RDF`.

W3C (2004a). Composite Capability/Preference Profiles (CC/PP). `http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/`.

W3C (2004b). OWL Web Ontology Language Overview. `http://www.w3.org/TR/owl-features/`. W3C Recommendation 10 February 2004.

Want, R., Hopper, A., Falcão, V., and Gibbons, J. (1992). The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91–102.

Wapforum (2001). User Agent Profile (UAProf) Specification. `http://www.wapforum.org`.

Weiser, M. (1991). The computer for the twenty-first century. *Scientific American*.

Winograd, T. (2001). Architectures for context. *Human-Computer Interaction (HCI) Journal*, 16(2):401–419.