A component-based approach for managing context information

Davy Preuveneers
Yolande Berbers

Report CW 397, December 2004

A component-based approach for managing context information

Davy Preuveneers
Yolande Berbers

Report CW 397, December 2004

Department of Computer Science, K.U.Leuven

Abstract

Context-awareness involves any kind of information that may characterize user-service interactions within a ubiquitous and pervasive computing environment. In this report we propose a component-based approach for the collecting, updating, transforming, reasoning, querying and the use of context information. This approach has resulted in a modular component-based context management infrastructure, with support for distributed execution and runtime adaptations of the underlying information processing algorithms.

Keywords : context information management, component-based infrastructure, ubiquitous computing
AMS(MOS) Classification : Primary : H.3.4, H.3.5 Secondary : C.2.4, C.5.0, I.2.4.

A component-based approach for managing context information

Davy Preuveneers and Yolande Berbers

Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium,
{davy.preuveneers, yolande.berbers}@cs.kuleuven.ac.be,
http://www.cs.kuleuven.ac.be

Abstract. Context-awareness involves any kind of information that may characterize user-service interactions within a ubiquitous and pervasive computing environment. In this report we propose a component-based approach for the collecting, updating, transforming, reasoning, querying and the use of context information. This approach has resulted in a modular component-based context management infrastructure, with support for distributed execution and runtime adaptations of the underlying information processing algorithms.

1 Introduction

Context-awareness [1] is being considered as the key challenge for making mobile devices aware of the situation their users are working and living in. This research domain focusses on the management of context information in ubiquitous and pervasive computing environments [2], where people are surrounded by and interacting with many unobtrusive networked devices. These devices will then offer personalized assistence by adapting the applications' intended functionalities to the current context of the user and the device. This context information involves any kind of information that may characterize the user-service interaction and includes, among other things, information regarding current location and time, users' activities and devices' capabilities.

This trend towards context-aware architectures within a mobile environment is the driving force behind applications and services being more sensitive to user requirements and being less dependent on user attention. A critical success factor is the support for adaptation as one of the foundations for general intelligence. This adaptation is needed due to applications and services being hosted on a broad range of hardware with different capabilities, such as PDAs, mobile phones and smartphones. However, not only do the hosted services need to be adaptable, the underlying context infrastructure must show some support for flexibility as well. If not, we risk of having the context infrastructure consuming all available resources without the device being to able to offer services with a reasonable quality of service. A component-based development approach [3] is an ideal design methodology for having flexible adaptation capabilities within

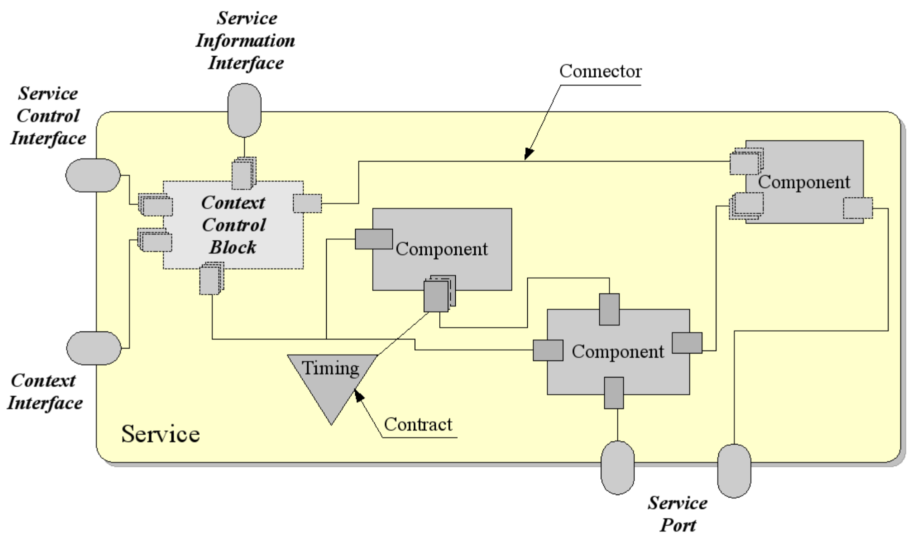

Fig. 1. General overview of a component-based service

applications as it incorporates interconnected components as functional building blocks. These components can be replaced by other components with similar behaviour but different runtime requirements, thus providing a way to optimize resource usage by pluggable components and to adapt to other working conditions. Other advantages of using components include the possibility of doing live updates using state transfers [4], negotiation and enforcement of resource contracts [5], distributed execution and relocation.

In section 2 we describe how the context management infrastructure fits within a component-based service platform. In section 3 we discuss how this context management infrastructure can be implemented using a component-based approach. This modular composition is responsible for the retrieval and dissemination, the storing, the reasoning and the transforming of context information. Section 4 provides an overview of related work. We end with conclusions and future work in section 5.

2 Context-awareness within a component-based service platform

A context-aware service platform requires the collaboration of, on the one hand, an infrastructure for managing context information and, on the other hand, the services which will offer personalized assistence through context-based adaptation. In this section we will briefly introduce the concept of component-based services and give a general overview of how these services interact with the context management system.

In several computer science domains the concept of a service refers to a computational entity that offers a particular functionality to a possibly networked environment. In contrast to the better known web services, context-aware services in mobile computing environments need to support user personalization, deployment on small embedded systems, user mobility and service relocation. To accomplish this, a component-based development methodology is being used. The general overview of a component-based service is shown in Figure 1.

Components [3] provide the functional building blocks of a service and use their *Component Ports* as communication gateways to other components. *Connectors* serve as the message channel between these ports. *Contracts* [5] define restrictions or requirements on two or more components or ports. They are used, for example, to limit or guarantee memory and network bandwidth availability or to define timing constraints, such as in videostreaming applications which require a minimal number of frames per second during operation to guarantee a certain quality of service. The *Context Control Block* is optionally shared by many services on the same device and responsible for managing the context information. This component is the focus of this paper and will be discussed in section 3.

A service then is a wrapper around these entities with some obligatory management interfaces and *Service Ports* as message gateway proxies to internal *Component Ports*. Hence, a service acts as a component with a hierarchical structure. As will be shown in section 3, the *Context Control Block* in itself is also composed out of several subcomponents. Regarding the obligatory management interfaces, the *Service Information Interface* provides static semantic and syntactic information about a service. This information is to be used by other devices during service discovery. The *Service Control Interface* is used to manage the runtime behaviour of a service, such as the launching, the relocating, the stopping and uninstalling of a service. The *Context Interface* is responsible for the service-specific context interchange and includes, among other things, information about the current resources in use. Together with the *Context Control Block*, it provides the coupling between the context management algorithms and the services.

3 Context management

In the following subsections we will discuss how an adaptable component-based context infrastructure, i.e. the *Context Control Block* as previously mentioned in section 2, is able to manage this information. The strength of this component-based approach relies on the fact that alternative components with similar function but different runtime requirements can be proposed, or that some components can be made optional if appropriate.

3.1 Context modeling

Before retrieving, storing and using context information for service adaptation, we require a uniform and interchangeable context representation. In the Context Toolkit [6], context is modeled as a set of key/value pairs. More structured approaches for modeling context have been proposed using RDF [7, 8], UAProf and CC/PP [9–11], and CSCP [12, 13]. Ontologies, on the other hand, allow the definition of more complex context models and have been used in several context modeling approaches [14–16].

The context model we will use here is an extensible context ontology tailored to context-driven adaptation of mobile services [17]. This context ontology is based on the concepts `User` {*profile, preferences, activities, role*}, `Platform` {*software, hardware, resources, I/O*}, `Service` {*syntactic and semantic service description*} and `Environment` {*time, location, temperature, humidity, pressure, noise, ...*}. See Figure 2 for a general overview of the ontology. This context model is known on each device so that each device shares the same concepts and confusion about their semantics is reduced to a minimum.

3.2 Context retrieval

This part of the context management infrastructure is responsible for the gathering of information from data providers on the system itself or in the environment. Several important aspects with respect to the source of information, the accuracy and mechanisms for information retrieval are discussed in the following subsections.

Sources of information

SENSORS: Information might be acquired by sensors attached to the device. This low-level information is subject to measurement errors and not always immediately usable without transformation to conceptually richer information. The MTS/MDA Sensor and Data Acquisition Boards from Crossbow Inc. [18] for wireless sensor networks, for example, provide support for gathering information about light, temperature, location, humidity, movement, accoustic ranging, magnetic fields, etc.

USER PROFILING: Another source of information is acquired through user profiling. Based upon a history of previous user actions and user input with respect to a certain service, a general profile with user preferences could be determined. It is clear that this kind of information is error prone and subject to change.

THIRD PARTIES: Information can also be exchanged with other parties, i.e. devices or information providers, in the neighborhood. This information might be raw sensor data, or be derived by combining various information. The accuracy and reliability of this information may or may not be available.

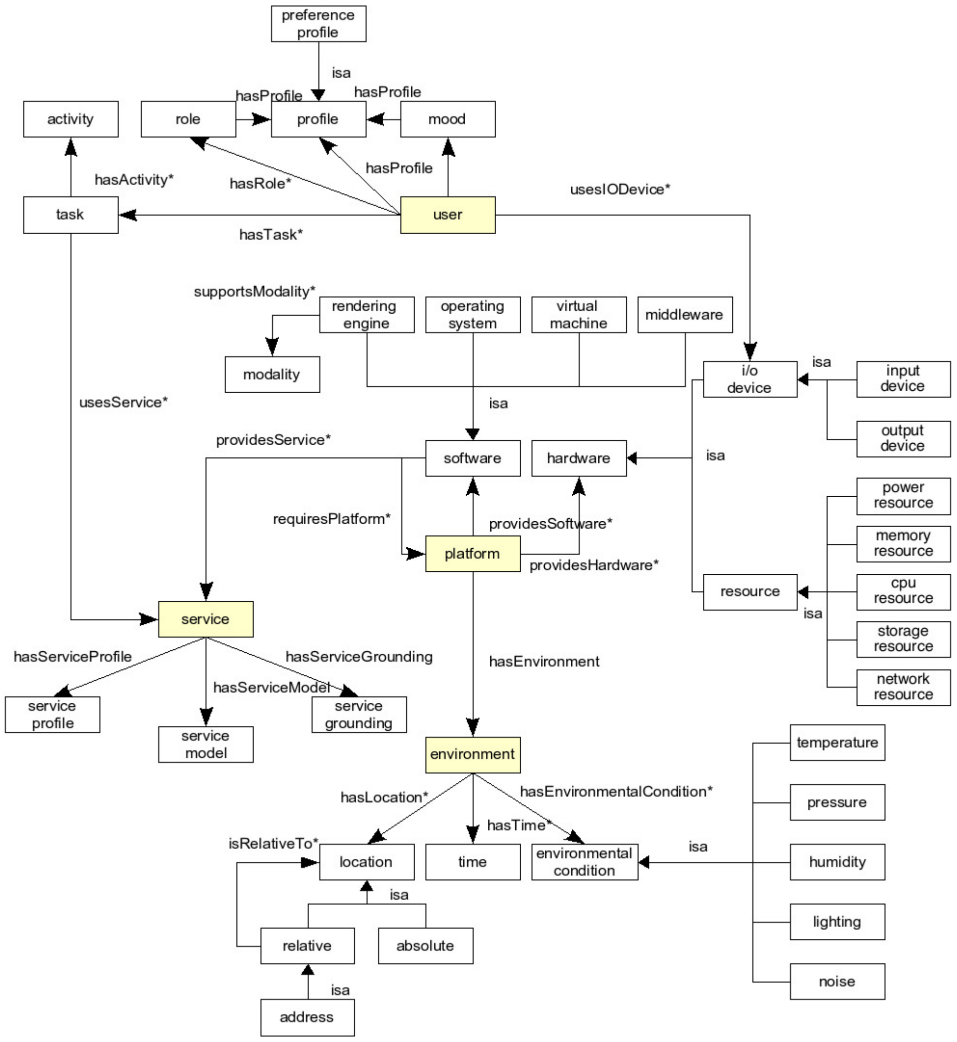

Fig. 2. Overview of the context ontology concepts

Properties of information

The value of information is not only determined by the information itself, but also by several information properties.

ACCURACY: With sensors as information providers, it is easy to determine the real value of sensed data, as the granularity of measurement and accuracy is usually provided by the manufacturer of the equipment. This is not always the case with profiled information or information provided by third parties. By combining information, small errors might propagate through the derivation chain and could in the end result into unuseful information.

RELIABILITY: Trust is important when a device is using information provided by third parties. Well-known devices have already had many occasions to prove their information to be accurate, whereas unknown devices never had such an opportunity. If many devices of both the well-known and unknown kind provide similar answers upon a request for information, trust and thus the reliability of these new unknown devices increases. In case of conflicts, a large majority of similar responses or feedback on the correct answer later on might influence the reliability of an information provider in a negative way.

AGE: Information might have been correct before, but might now be too old to be useful. Therefore, information should be decorated with a time stamp defining its age. If measuring or deriving information takes to long, we can fall back on a previous value if the information is not too old.

Mechanisms for information retrieval

There are several mechanisms to receive information, each with their own advantages and disadvantages:

PUSH-BASED INFORMATION RETRIEVAL: In this case, the context management infrastructure does not send any information requests. The information is offered all the same and the context management infrastructure may decide to store and use it, or may choose to ignore it. An example of such an information provider would be a clock, periodically broadcasting time signals to all the services on a device. This mechanism is well-suited for time-critical services requiring accurate time information, such as multi-media applications, as they do not need to poll every time for this information, but it also increases communication overhead when every component or service is ignoring the broadcasted information.

PULL-BASED INFORMATION RETRIEVAL: With a pull-based mechanism, the context management infrastructure must explicitely send an information request to a sensor or to third parties in its vicinity. The receiver of the request may perform a new measurement or send the latest registered value if still appropriate.

INTERVAL-BASED INFORMATION RETRIEVAL: This is more or less a combination of the previous mechanisms. The context management infrastructure first sends

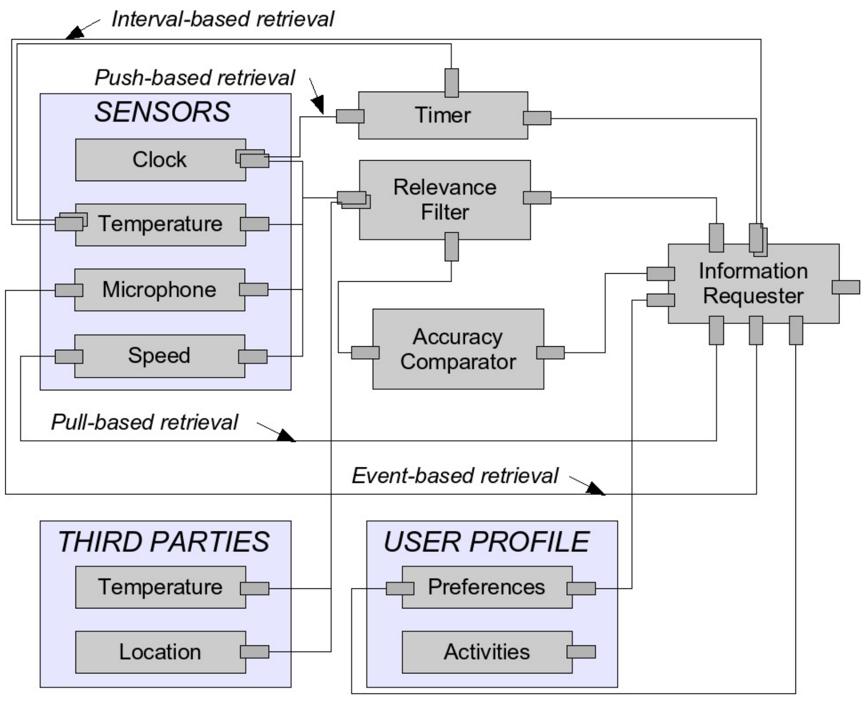

Fig. 3. Overview of the context retrieval

an information request and a time interval after which it will be periodically notified with information updates until the client requests to end the notification. This mechanism is the ideal candidate for monitoring purposes, for example to notify services of increased network bandwidth.

Event-based Information Retrieval: This mechanism uses the publish-subscribe design pattern to notify interested parties when new information is available. Polling mechanisms might prove to be very resource consuming when monitoring a certain information provider which rarely changes. Event-based notification provides a solution for this problem and decreases unnecessary communication.

Component-based information retrieval modeling

An overview of all components involved with context retrieval is given in Figure 3. On the one hand, we have several components acting as information providers. These are split up into *Sensors*, *User Profile* and *Third Parties* categories. In fact, the *User Profile* is actually part of our context ontology in Figure 2. On the other hand, we have components responsible for the filtering and selecting of the most relevant and accurate information. The *Information*

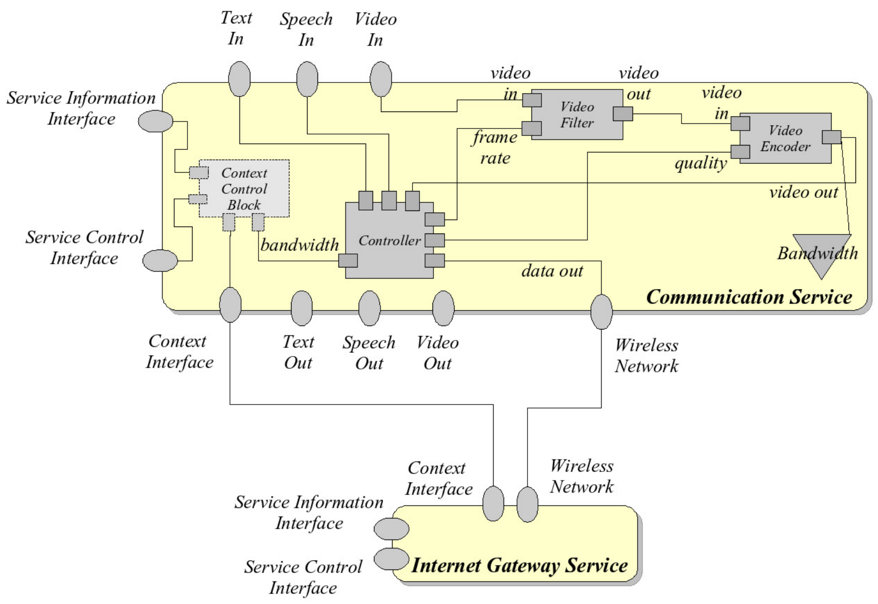

Fig. 4. Internet Gateway Service and Communication Service

Requester is the initiator of all information requests. In this example, a *Clock* periodically sends a time signal and pushes this information to a *Timer*-component which is used to enable interval-based *Temperature* information retrieval. The difference between the pull-based and event-based retrieval initiated by the *Information Requester* is not clear in the figure above. However, for event-based information retrieval the *Information Requester* submits a subscription message to the *Microphone*, so that a notification will be sent when the microphone detects noise.

Depending on the services that are being hosted and the capabilities of the device, some components could be reduced in complexity or even eliminated. For example, instead of comparing the accuracy and reliability, we can simplify the *Accuracy Comparator* by only retaining the first answer in a set of responses, with a possible reduction in accuracy of context information as a result. Suppose that a service relies only on its own sensors to provide information, then the *Relevance Filter* and *Accuracy Comparator* can be completely removed.

3.3 Context dissemination

Context dissemination is useful for sending context information about a certain service to other interested third parties. As discussed before, each service has a specific interface for sharing this information, i.e. the *Context Interface*. We will discuss how the context information is exchanged by demonstrating it with an example.

Context dissemination by example

For example, a *Communication Service* on a PDA with support for text messaging, audio- and videostreaming is interacting with a *Internet Gateway Service* being offered on an airplane. See Figure 4 for a simplified overview of both services. The type of communication depends on the available bandwidth as the Internet broadband connection down to earth needs to be shared by several people. Every user thus has a certain bandwidth quota. The *Communication Service* is subscribed to message events from the *Internet Gateway Service* that notifies all clients about a change in maximum bandwidth usage. This message flows from a *Network Monitor Sensor* to the *Information Requester* (not shown in figure), it then travels on to the *Context Interface* of the *Internet Gateway Service.* The message is then forwarded to the *Context Interface* of the *Communication Service*, after which it is handled by its *Information Requester* as a message from a third party. The rest of the processing was discussed in the previous section.

Component-based information dissemination modeling

No new components need to be introduced for the dissemination of context information, as the relevant components have already been described in the previous section.

3.4 Context storing

A context repository is responsible for persistency of context information, and thus can be considered as a small-scale database filled with facts. However, since mobile devices have a limited storage capacity, special care needs to be taken to only save up-to-date and relevant information. Another issue is the storing of data so that queries and information updates can be handled efficiently without losing the semantics of the data. These topics are discussed in the following subsections.

Context representation

Context representation involves two aspects. On the one hand, we have the information itself, for example '`Age=23`', and on the other hand, the fact that this information relates to other concepts, for example the name of the person. The first aspect can be respresented by a set of key/value attributes. The relationships between this fact and other concepts determine the semantic meaning of this information. To accomplish this, ontologies are used as knowledge representation.

Hence, context information is represented as a set of ontologies, modeling the concepts, and an attribute container with a flat list of key/value pairs which provides facts as an instantiation of concepts in these ontologies. Therefore, each fact in the attribute container also refers to the concept it instantiates. A special knowledge base stores all ontologies, including the base ontology in Figure 2, so that knowledge can be expanded with new concepts when necessary. The advantage of using a separate attribute container is the ease with which this container can be queried.

History of context information

When the available storage capacity allows it, it might prove useful not only to save the most recent value of a certain attributes, but to also retain previously received values. In this way, the history of information can be exploited. Typical examples would be to track the current location to calculate the travelled distance or to monitor resource usage in a pay-per-unit billing service. This can easily be implemented by including a time stamp with each key/value attribute.

Managing outdated and redundant information

As storage capacity is not unlimited, not all information can be retained. The oldest information is purged first, but the time during which information stays relevant is not the same for all concepts. Therefore, a possible solution is to provide for each concept a certain lifetime for which the information is still of some value. If the information is older than the given lifetime, it will be garbage collected.

A more complicated problem of information overload is the presence of redundant information. Should information be removed after it has been used to derive new information or should it be retained for later use? One solution for this complex problem is to store for each fact the latest occasion of when and how often it has been used. It is clear that rarely used and old information is the first candidate to be removed. However, storing extra properties about facts requires storage space as well, and thus the advantages and disadvantes should be thoroughly considered before taking any decision how to implement the removing of old data.

Component-based context repository modeling

The component-based repository implementation is largely based on two different container components with their available storage capacity as most important characteristic. See Figure 5 for an overview. The *Information Requester* sends new facts to the *Fact Container* which holds instantances of concepts from ontologies. These ontologies are also forwarded by the *Information Requester* and held in the *Ontology Container*. Two switches are used to enable and disable history preservation and usage tracking. When low on memory or storage capacity, another signal is used to trigger the garbage collection of old facts. If one of the supplemental ontologies, i.e. not the base ontology in Figure 2, is no longer refered to by a key/value pair, then the ontology can be removed from the *Ontology Container* as well. A small instantiation of the *Fact Container* is given

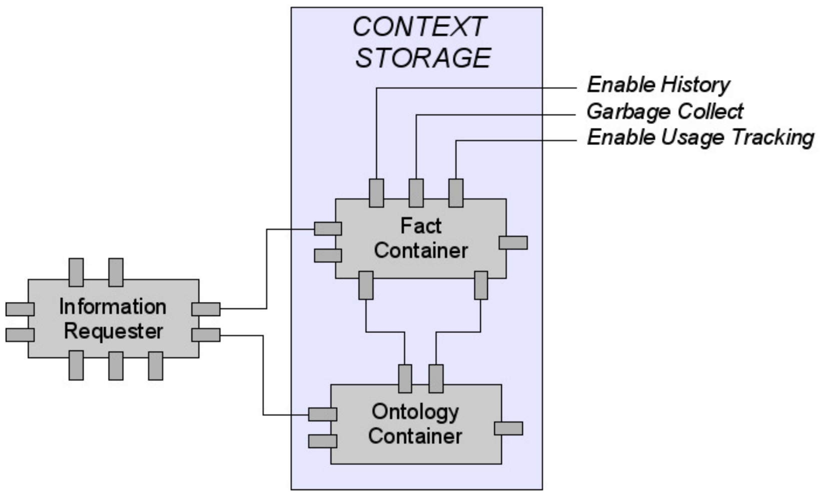

Fig. 5. Overview of context storing

ID	Attribute	Value	Concept ID	Time Stamp	Last Used	Usage Count
1	`Name`	`John`	`ID74358`	`07:53am`	`11:52am`	`7`
2	`Age`	`53`	`ID69230`	`07:54am`	`10:16am`	`2`
3	`Location`	`50°52' N 4°22' E`	`ID38031`	`02:37pm`	`02:37pm`	`119`
4	`Bandwidth`	`1112 kbps`	`ID16789`	`02:38pm`	`02:41pm`	`37`
5	`*LIFETIME*`	`30 sec`	`ID16789`	`-`	`-`	`-`

Table 1. Instantiation of the *Fact Container*

in Table 1. Properties with respect to accuracy and reliability of information have been omitted from this table for readability purposes. In this fact table, the measurement of the current bandwidth usage is specified to be valid for at most 30 seconds, after which it is invalidated and removed from the fact table.

3.5 Context manipulation

This part of the context management infrastructure is responsible for the transforming and reasoning on context information. It is also responsible for the decision making and adaptation of services, and thus its main objective is to make services context-aware.

Context transformation

Context transformation changes the way how certain information is represented.

For example, a temperature expressed in degrees Celsius could be transformed into degrees Fahrenheit or Kelvin using simple mathematical transformation rules. Similar transformations are possible for converting lengths specified in English or metric units of length. Classification is another kind of transformation, where accuracy of information is given up for the sake of more meaningful information. For example, the geographical location specification in Table 1 using longitude and latitude coordinates could be replaced by the nearest major city, in this case Brussels, resulting in a better human understanding of the location. Classification however is a more complex transformation as it requires extra information defining the categories and a general distance function to select the category which fits best.

Context reasoning

Context reasoning derives new information based on existing facts and derivation rules. Whereas context transformation changes the way how a concept is expressed, context reasoning combines derivation rules and facts resulting into other facts which were only available implicitely. Suppose a calendar service has information regarding events and activities planned during the day. By combining this information with the current time, it is possible to derive and predict the current location of a person. Several expert systems with rule engines (Jess [19], CLIPS [20]), general purpose inference systems (Prolog's resolution refutation) and ontology reasoners (Racer [21], Pellet [22], FaCT [23]) already exist to implement the reasoning part of context manipulation.

Context-based decision making and adaptation

Deriving and transforming context information without actually using it is pointless. The focus of this part of the context management infrastructure is to make decisions or take actions automatically without any user intervention. Most of the decisions and actions will have an effect on the adaptation of services. For example, consider the *Communication Service* and *Internet Gateway Service* in Figure 4. The *Communication Service* has the following components:

- *Audio Encoder and Decoder*: Adaptable components for (de)compressing the audio stream with high, medium or low quality encoding.
- *Video Filter*: Optional component for reducing the video frame rate.
- *Video Encoder and Decoder*: Adaptable components for (de)compressing the video stream with high, medium or low quality encoding.
- *Controller*: (de)multiplexes text, speech and video, and sends/receives the combined data stream.

The video-encoding in the communication service can be adapted to adhere to certain bandwidth constraints by enabling optional filters or by changing the compression scheme. The adaptation of this service is based on several triggers being activated. In this case, triggers define different bandwidth conditions to optimize the quality of service of the communication. The information that

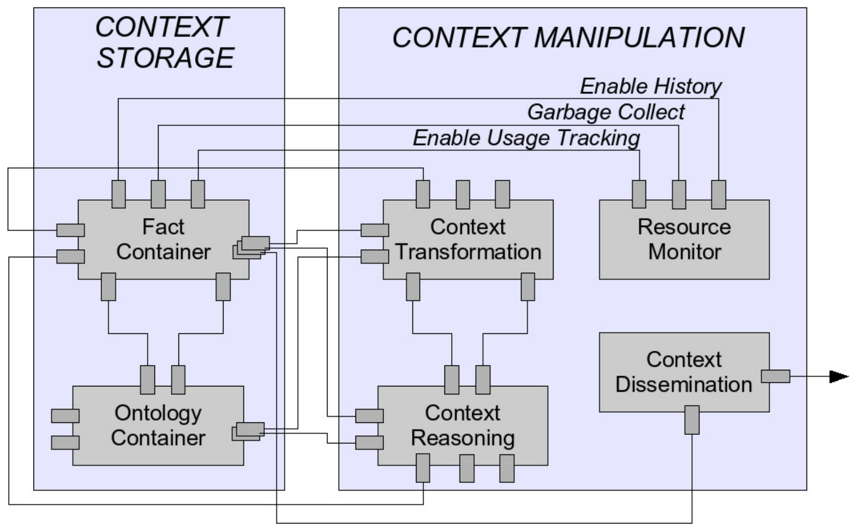

Fig. 6. Overview of context manipulation

activates these triggers might come from sensors or be deduced after several transformation and derivation steps.

Component-based context manipulation

The general overview of the component-based context manipulation is given in Figure 6. The *Context Transformation*-component and *Context Reasoning*-component are able to derive new facts which are stored in the *Fact Container*. The *Resource Monitor* is responsible, among other things, for enabling garbage collecting on the *Fact Container* when running low on storage capacity. The *Context Dissemination*-component is responsible for providing the necessary information to the triggers (not shown in the figure) that activate service adaptation. If, for some reason, certain necessary context information is not present or cannot be derived, then the *Context Dissemination*-component is also able to send a service request to third parties by specifying all possible inputs and required output. For example, we might be able to derive the home location of a person, but have no way to contact him. By providing name and address as possible inputs, the *Context Dissemination*-component could send a service request to a *Yellow Pages*-service that is able to provide the necessary information. Hence, this is a simple example of context-based service discovery and interaction.

3.6 Implementation

The component-based management infrastructure has been partially implemented on top of Draco [24], an extensible runtime system for components designed to be run on embedded devices. The base system is very small and lightweight with support for extensions such as component distribution, live updates, contract monitoring and resource management. This runtime environment with extensions provides a unique test platform for validating the proposed concepts in a pervasive and ubiquitous computing context.

4 Related work

Research on context-awareness has already resulted in several applications [25] which use a specific type of context information, for example location- and time-based information. Examples are tour guides such as Cyberguide [26], reminder applications such as MemoClip [27] and CybreMinder [28], or intelligent environments as in Classroom 2000 [29].

More advanced applications are tailored to a specific research approach. For example, the Context Toolkit [6] is a framework that aims to facilitate the capturing, interpreting and sharing of context information using *Context widgets*, similar to GUI widgets which are responsible for the presentation layer of an application. Service adaptation is not the main focus of this framework.

The CoBrA architecture [30, 15, 31] is a context broker using ontologies that maintains a shared model of context on the behalf of a community of agents, services, and devices in a certain environment and provides privacy protection for the users in the space by enforcing the policy rules that they define. The advantage of our component-based approach over this architecture, is the better support for an adaptable context management system. Another difference is that CoBrA manages the context for all computing entities in a certain environment instead of each device carrying and managing its own contextual knowledge. Thus, the main advantage of our approach is the better support of mobility in ubiquitous and pervasive environments by not being dependent on another device for managing the context information.

Another service platform with support for context-aware service adaptation is the platform proposed by Efstratiou et al. in [32]. The authors describe the architectural requirements for adaptation control and coordination for mobile applications. In our approach, we have not only shown how services can be adapted, but also how the driving force behind adaptation, i.e. the context management system, can also be adapted to different working conditions with support for distributed execution and relocation.

The M3 architecture [33] is a reactive framework supporting different levels of context-awareness and adaptation for a variety of mobile work situations. Services are also component-based. Their context management, however, is less flexible compared to the plugable and distributed executable component-based context management system proposed in this report.

5 Conclusions and future work

In this report, we have presented a component-based approach for managing context information. The main advantage is that the management infrastructure can be adapted to a device's capabilities or service requirements by enabling or disabling certain components or specific properties of certain components, such as the preservation of context history information in the *Fact Container*-component.

Another advantage is that these components do not necessarily have to be executed on the same device. When, for example, available processing power is very limited, then the necessary facts and ontologies can be sent to a more powerful device so that the transformation and reasoning on context information can be delegated.

Future work will focus on the modeling of resource requirements for context transformations and context derivations, so that a rough estimate of processing time can be made. This is useful if a user has a preference for receiving a rough but quick response, or for receiving a more accurate answer for which he is willing to wait a bit longer.

References

1. Dey, A.K., Abowd, G.D.: Towards a better understanding of context and context-awareness. In: Workshop on The What, Who, Where, When, and How of Context-Awareness, Conference on Human Factors in Computer Systems (CHI2000). (2001)
2. Satyanarayanan, M.: Pervasive Computing: Vision and Challenges. IEEE Personal Communications (2001) 10–17
3. Urting, D., Van Baelen, S., Holvoet, T., Berbers, Y.: Embedded Software Development: Components and Contracts. In: Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems. (2001) 685–690
4. Vandewoude, Y., Berbers, Y.: Run-time evolution for embedded component-oriented systems. In Werner, B., ed.: Proceedings of the International Conference on Software Maintenance, Montréal, Canada, IEEE Computer Society (2002) 242–245
5. Wils, A., Gorinsek, J., Van Baelen, S., Berbers, Y., De Vlaminck, K.: Flexible Component Contracts for Local Resource Awareness. In Bryce, C., Czajkowski, G., eds.: ECOOP 2003 Workshop on resource aware computing. (2003)
6. Dey, A.K., Salber, D., Abowd, G.D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction (HCI) Journal **16** (2001) 97–166
7. Beckett, D.: RDF/XML Syntax Specification (Revised). http://www.w3.org/TR/rdf-syntax-grammar/ (2003)
8. Korpipää, P., Mätyjärvi, J., Kela, J., Keränen, H., Malm, E.J.: Managing Context Information in Mobile Devices. IEEE Pervasive Computing, Mobile and Ubiquitous Systems **2** (2003) 42–51
9. FORUM, W.: UAProf User Agent Profiling Specification (1999, amended 2001)
10. Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M.H., Tran, L.: Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0. http://www.w3.org/TR/2003/PR-CCPP-struct-vocab-20031015/ (2003)

11. Indulska, J., Robinson, R., Rakotonirainy, A., Hendricksen, K.: Experiences in Using CC/PP in Context-Aware Systems. In Stefani, J.B., Dameure, I., Hagimont, D., eds.: LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003). Volume 2893 of Lecture Notes in Computer Science (LNCS)., Paris/France, Springer Verlag (2003) 224–235
12. Buchholz, S., Hamann, T., Hubsch, G.: Comprehensive Structured Context Profiles (CSCP): Design and Experiences. In: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops. (2004)
13. Held, A., Buchholz, S., Schill, A.: Modeling of Context Information for Pervasive Computing Applications. In: Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI2002). (2002)
14. Strang, T., Linnhoff-Popien, C., Frank, K.: CoOL: A Context Ontology Language to enable Contextual Interoperability. In Stefani, J.B., Dameure, I., Hagimont, D., eds.: LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003). Volume 2893 of Lecture Notes in Computer Science (LNCS)., Paris/France, Springer Verlag (2003) 236–247
15. Chen, H., Finin, T., Joshi, A.: An Ontology for Context-Aware Pervasive Computing Environments. Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review (2003)
16. Gu, T., Wang, X.H., Pung, H.K., Zhang, D.Q.: An Ontology-based Context Model in Intelligent Environments. In Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, San Diego, California, USA (2004)
17. Preuveneers, D., Van den Bergh, J., Wagelaar, D., Georges, A., Rigole, P., Clerckx, T., Berbers, Y., Coninx, K., Jonckers, V., De Bosschere, K.: Towards an extensible context ontology for Ambient Intelligence. In: Proceedings of the Second European Symposium on Ambient Intelligence, Springer (2004)
18. Crossbow Inc.: Smarter Sensors for Ubiquitous Computing. http://www.xbow.com/Industry_solutions/Ubiquitous_Computing.htm (2004)
19. Sandia National Laboratories: Jess, the Rule Engine for the Java Platform. http://herzberg.ca.sandia.gov/jess/ (2004)
20. Riley, G.: CLIPS: A Tool for Building Expert Systems. http://www.ghg.net/clips/CLIPS.html (2004)
21. Haarslev, V., Moller, R., Wessel, M.: Racer, Semantic Middleware for Industrial Projects Based on RDF/OWL, a W3C Standard. http://www.sts.tu-harburg.de/˜r.f.moeller/racer/ (2004)
22. Mindswap: Pellet OWL Reasoner. http://www.mindswap.org/2003/pellet/index.shtml (2003)
23. Horrocks, I.: The FaCT System. http://www.cs.man.ac.uk/˜horrocks/FaCT/ (2003)
24. Vandewoude, Y., Rigole, P., Urting, D., Berbers, Y.: Draco : An adaptive runtime environment for components. Technical Report CW372, Department of Computer Science, Katholieke Universiteit Leuven, Belgium (2003)
25. Chen, G., Kotz, D.: A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College (2000)
26. Abowd, G., Atkeson, C., Hong, J., Long, S., Kooper, R., Pinkerton, M.: Cyberguide: A mobile context-aware tour guide. ACM Wireless Networks **3** (1997) 421–433

27. Beigl, M.: MemoClip: A location-based remembrance appliance. Personal Technologies **4** (2000) 230–234
28. Dey, A.K., Abowd, G.D.: CybreMinder: A Context-Aware System for Supporting Reminders. In: HUC. (2000) 172–186
29. Abowd, G.D.: Classroom 2000: An experiment with the instrumentation of a living educational environment. IBM Systems Journal **38** (1999) 508–530
30. Chen, H.: An intelligent broker architecture for context-aware systems. http://cobra.umbc.edu/ (2003)
31. Chen, H., Finin, T., Joshi, A.: Using OWL in a Pervasive Computing Broker (2003)
32. Efstratiou, C., Cheverst, K., Davies, N., Friday, A.: An Architecture for the Effective Support of Adaptive Context-Aware Applications. In: Proceedings of 2nd International Conference in Mobile Data Management (MDM‘01). Volume Lecture Notes in Computer Science Volume 1987., Hong Kong, Springer (2001) 15–26
33. Indulska, J., Loke, S., Rakotonirainy, A., Witana, V., Zaslavsky, A.: An Open Architecture For Pervasive Systems. In: Proceedings of the Third IFIP TC6/WG6.1 International Working Conference on Distributed Applications and Interoperable Systems, Kluwer (2001) 175–187