Adaptive context management using a component-based approach

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 has become a necessity for adaptable intelligent applications and services. It is crucial for ubiquitous and pervasive computing because the context of a user or device serves as the source of information to adapt services. In this paper, we propose a modular context management system that is able to collect, transform, reason on and use context information to adapt services. By employing a component-based approach, we enable our infrastructure not only to support context-aware adaptation of services, but also to support adaptation of the context management system itself at deployment time and at runtime. This self-adaptation is based upon the service requirements and the current context of the device, such as the current resource usage or other devices in the neighborhood, resulting in an adaptive context management system for improved quality of service.

1 Introduction

Context-awareness [1] is considered to be the key challenge for making mobile devices aware of the situation of their users and their environment. This research area focuses on the management of context information in pervasive computing environments [2] where people are surrounded by and interacting with many unobtrusive networked devices. These devices offer personalized assistance by adapting their applications' intended functionalities to the current context of the user and the device. This context information includes current location and time, users' activities and preferences, devices' capabilities, and any information that may characterize user-service interactions. Context representation has evolved from simple key-value pairs [3] to more complex ontology models [4–7] to provide semantical uniformity and universal interchangeability.

The context management system is the heart of a context-aware architecture and processes instantiations of this context model. It is responsible for information retrieval and dissemination, structured storage of context, transformation of and reasoning on information, and the decision process to initiate certain actions. The current trend towards context-aware architectures is explained by the growing need for applications and services that are more sensitive to user requirements but less dependent on user attention. Hence, a critical success factor of a context-aware architecture in a mobile and ubiquitous computing environment

is the support available to adapt services to a broad range of hardware, such as PDAs, mobile phones and smartphones. The underlying context management system must support flexibility as well. Otherwise, the context management system can consume all resources and reduce the quality of service to the user. For this reason, we have used *components* as modular building blocks for the design and deployment of the adaptable services and for the underlying context management system.

A component-based development approach [8] is an ideal software engineering methodology for having flexible adaptation capabilities within applications to optimize resource usage and to adapt to changing working conditions. Advantages of using components include the possibility of live updating with better suited components [9], negotiating and enforcing resource contracts [10], distributing the execution and relocating component-based applications [11].

In section 2 we give a general overview of adaptation in our context-aware architecture. In section 3 we describe how our context management system fits within a component-based service-oriented platform. In section 4 we discuss our component-based implementation and support for self-adaptation. The modular composition manages the retrieval and dissemination of, the storage of, the reasoning on and the transformation of context information. In section 5 we evaluate our system and discuss future work. Section 6 provides an overview of related work. We end with conclusions in section 7.

2 Adaptation in a context-aware architecture

Both the services and the context management system are subject to adaptation triggered by a changing context, as discussed in the following subsections.

2.1 Service adaptation

First of all, service adaptation before deployment of a service ensures that the service is tailored to the capabilities of the device [12]. Secondly, service adaptation can also be activated during the execution of a service. By way of example, consider a video conferencing service that adapts to a reduced network bandwidth by lowering the video frame rate or by disabling video altogether. Our context management system initiates both deployment time and runtime adaptations by providing all the necessary information to activate the adaptations. Each service specifies constraints that define working conditions that guarantee proper execution of (a subset of) the provided functions of the service. These constraints are encapsulated by triggers. A context change causing certain constraints to be violated will then trigger the runtime adaptation of the service. A brief overview of the component-based service model is given in section 3.

2.2 Context management self-adaptation

Our context management system itself is also subject to adaptation. Specific components of the context management system can be eliminated if they are of

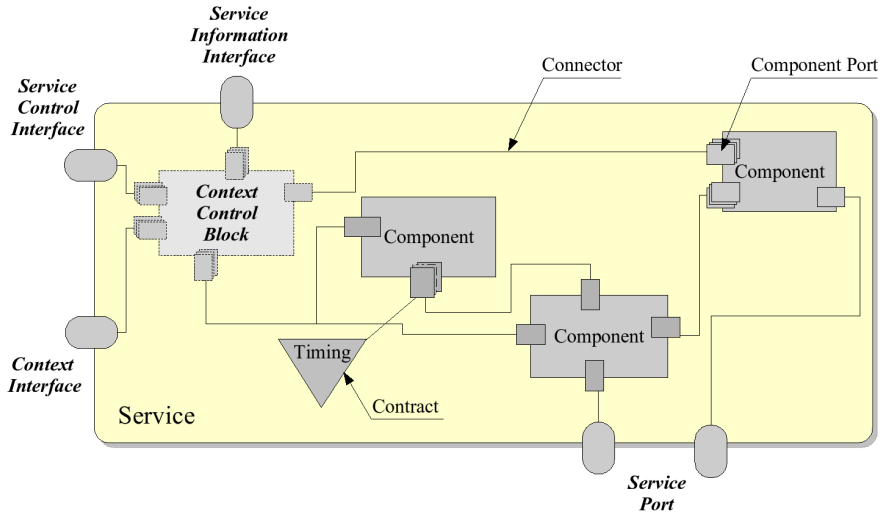

Fig. 1. Building blocks of a component-based service

no use to any service. For example, a due shortage of storage capacity triggers the context management to delete irrelevant context information to increase allocation space. If context-aware service adaptation only depends on the current value of local sensors, then the context storage can be eliminated altogether.

To not overcomplicate the self-adaptation of our context management system, the adaptation is only triggered by changes in resources and service requirements. This information is usually readily available and requires no intensive processing.

3 Context-awareness within a component-based service platform

A context-aware service platform requires the interaction between a context managing infrastructure and the services which offer personalized assistance through context-based adaptation. In this section we briefly introduce our component-based services and their interaction with the context management system.

In several computer science domains a service refers to a computational entity that offers a particular functionality to a possibly networked environment. Context-aware services in mobile computing also require support for user personalization, deployment on embedded systems, user mobility and service relocation. To accomplish this, we apply a component-based development methodology. A general overview of our component-based service is shown in Figure 1.

Components [8] provide the functional building blocks of a service and use *Component Ports* as communication gateways to other components. *Connectors* serve as the message channel between these ports. *Contracts* [10] 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 in order to guarantee a certain quality of service.

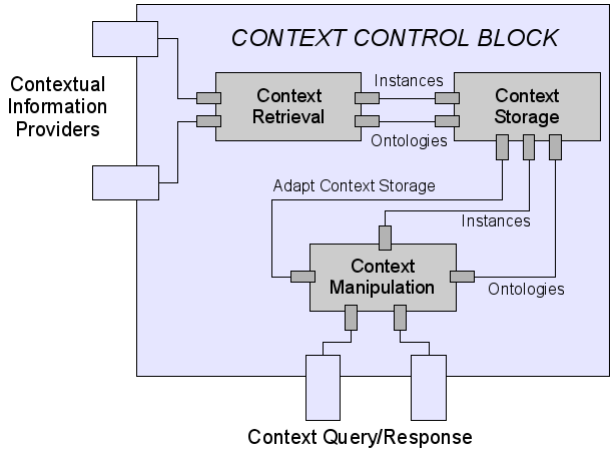

Fig. 2. Overview of the *Context Control Block* component

The *Context Control Block* is responsible for managing the context information. This *Context Control Block* is largely shared by all services on the same device to eliminate the need for duplication for all services. Only components with a service-specific function cannot be shared, such as those processing service-specific required accuracy of information. The *Context Control Block* which is the focus of the rest of this paper is discussed in section 4.

A service is a wrapper around these entities with *Service Ports* as message gateway proxies to internal *Component Ports* and three management interfaces: the *Service Information Interface* to provide static semantic and syntactic information about a service during service discovery, the *Service Control Interface* to manage the launching, the relocating, the stopping and the uninstalling of a service, and the *Context Interface* for the service-specific context interchange and interaction with the *Context Control Block*, i.e. the context management system. As shown in section 4, the *Context Control Block* in itself is also composed of several subcomponents, each with a specific function.

4 Context management

Retrieving and using context information require a uniform and interchangeable context representation. In the Context Toolkit [3], context is modeled as a set of key-value pairs. The more structured approaches for modeling context that have been proposed in the past use RDF [13], UAProf and CC/PP [14], and CSCP [15]. Ontologies, which allow the definition of more complex context models, have been used in several context modeling approaches [4–6]. For use in our context management system, we have designed a context ontology [7] based on the concepts of *User*, *Platform*, *Service* and *Environment*. This ontology is specifically targeted at context-driven adaptation of mobile services [16].

In the following subsections we discuss how an adaptable component-based context infrastructure, i.e. the *Context Control Block* as previously mentioned in

section 3, is able to manage context information. Apart from managing context, the strength of our component-based approach relies on the fact that components with similar function but different runtime requirements can adapt the behavior of the context management system. The following subsections treat respectively a general overview of the context management system, context retrieval, context storage and context manipulation. Where applicable, they discuss how alternative and optional components can adapt the context management system to better suite the needs of the context-aware services.

4.1 General overview of the context management system

The job of context management is performed by the *Context Control Block*. It consists of three components, each with a specific duty: *Context Retrieval*, *Context Storage* and *Context Manipulation*. See Figure 2 for a general overview.

4.2 Context retrieval

This component gathers information from sensors or other providers on the system itself or in the neighborhood. Several issues with respect to the source of information and accuracy are discussed in the following subsections.

Sources of information We distinguish the following information providers:

Sensors: Information can be acquired by sensors attached to the device [17]. This low-level information is prone to measurement errors and can require transformation into conceptually richer information before being usable.

User profiling: Another source of information is acquired through user profiling. Based upon a history of previous user actions and input, a general profile with user preferences can 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 in the neighborhood. This information can be raw sensor data or be derived by combining all kinds of information.

Properties of information The value of information is determined not only 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 guaranteed for user profiled information. By combining information, small errors can propagate through the derivation chain and result in unusable 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 have not had such an opportunity. Trust in other devices is managed by comparing all answers which influences trust in a positive or negative way. Note that this is no guarantee against hostile or malicious information providers.

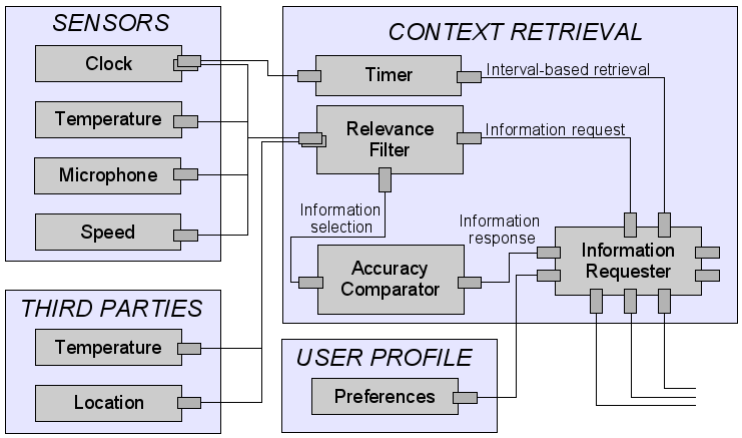

Fig. 3. Overview of the *Context Retrieval* component

AGE: Information, which proved to be valuable before, can now be too old to be useful. Therefore, information is labeled with a time stamp defining its age. If the measuring or deriving of information takes too long, we can fall back on a previous value, but only if that information is not too old.

Component-based information retrieval modeling An overview of all components involved in context retrieval is given in Figure 3. We have several components acting as information providers: *Sensors*, a *User Profile* and *Third Parties*. The *Information Requester* is the initiator of all information requests. In general, it monitors information that triggers service adaptations, such as changes in current network bandwidth. It sends these requests to the *Relevance Filter*, which forwards them to the information providers. Another function of the *Relevance Filter* component is to filter out unwanted information which has been pushed into the context management system by third parties. When several sources provide similar information in response to a request, the *Accuracy Comparator* selects the 'most reliable and accurate information' and forwards it back to the *Information Requester*. In Figure 3, a *Clock* also periodically sends a time signal and pushes this information to a *Timer* component. The *Timer* uses this information to enable configurable periodic signals. The *Information Requester* can then send a request to the *Timer* to be periodically notified to allow interval-based information monitoring.

Support for adaptation Depending on the processing capabilities of the device, components can be reduced in complexity or even eliminated. For example, instead of comparing and selecting on accuracy and reliability, we can replace the *Accuracy Comparator* by another component that only retains the first answer in a set of responses, with a possible reduction in accuracy of context information as a result. In the event a service only relies on the sensors of the device as information providers, then the *Relevance Filter* and *Accuracy Comparator* in Figure 3 can be completely removed, which means that the sensors are connected directly to the *Information Requester*.

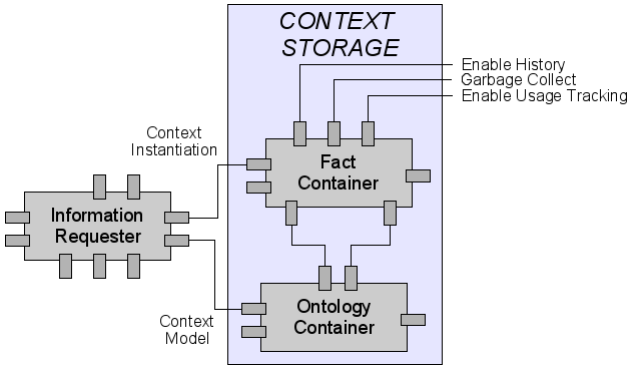

Fig. 4. Overview of the *Context Storage* component

4.3 Context storing

A context repository ensures persistency of context information. It should save only up-to-date and relevant information in a way that queries and information updates can be handled efficiently without losing the semantics of the data.

Context representation Context representation involves two aspects: the information itself, for example '`Age=23`', and how it relates to other concepts, for example a person. The first aspect can be represented by a set of key-value attributes. The second determines the semantic meaning of this information.

Our context management system stores context as an expandable knowledge base of ontologies for semantic modeling of concepts and as a fact container with key-value pairs providing instantiations of these concepts. Using a separate attribute container simplifies querying the instantiations.

History of context information For monitoring services it is useful to not only save the most recent value of a certain attribute, but to retain previously received values as well. In this way, the history of information can be exploited, for example by calculating the distance traveled by tracking the current location. This is implemented by including a time stamp with each key-value attribute.

Managing outdated and redundant information As storage capacity is limited, not all information can be retained. The oldest information is purged first, but the duration of relevance is not the same for all concepts. Therefore, we provide for each concept a lifetime during which it is still of value. If the information is older than the given lifetime, it is garbage collected.

Redundancy is a more complicated problem of information overhead. Should information be removed after it has been used to derive new information or should it be retained for later use? Our solution stores for each fact the latest occasion of when and how often it has been used. Rarely used and old information are the first candidates for removal. However, storing extra properties about facts requires storage space as well, and thus the advantages have to be thoroughly considered before implementing the removal of old or redundant data.

Table 1. Example of an instantiation of the *Fact Container*

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	-	-	-

Component-based context repository modeling The component-based repository is implemented as two different container components. See Figure 4 for an overview. The *Information Requester* sends new facts to the *Fact Container*, which holds instances of concepts from context ontologies. Two switches are used to enable and disable history preservation and usage tracking. When low on storage capacity, another signal is used to trigger the garbage collection of old facts. If one of the supplemental ontologies (i.e. not our base ontology [7], which serves as a common ground), is no longer referred to by a key-value pair, then it can be removed from the *Ontology Container* as well. An instantiation of the *Fact Container* is given in Table 1. Properties with respect to accuracy and reliability of information have been omitted for legibility reasons. 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 removed from the fact table.

Support for adaptation Storage can be unnecessary or outsourced to a device with more storage capacity. In the latter case, information requests have to be sent to a third party by using the *Context Interface* of a service. There is no requirement that all components have to execute on the same device.

4.4 Context manipulation

This part transforms and reasons on context information to provide suitable information for initiating the context-aware service adaptation.

Context transformation Context transformation changes the way certain information is represented. For example, a temperature expressed in $^\circ C$ can be transformed into $^\circ F$ using simple mathematical transformation rules. 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 can be replaced by the nearest major city, in this case Brussels, resulting in a better human understanding of the location. Classification is more complex compared to the mathematical equivalences as it requires extra information defining the categories and a general distance function to select the category that fits best.

Context reasoning Context reasoning derives new information based on existing facts and derivation rules. Whereas context transformation changes the way a concept is expressed, context reasoning combines derivation rules and facts into other facts which were only available implicitly. For example, a calendar

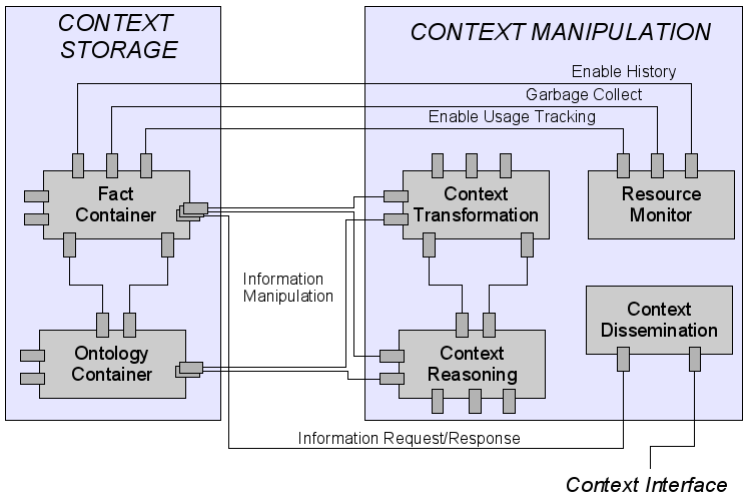

Fig. 5. Overview of the *Context Manipulation* component

service with information on events combined with the current time allows to predict the current location of a person. Initially, we investigated Racer and Jess as reasoning tools, but due to technical integration issues we chose for the Jena 2 Ontology and Reasoning subsystem [18] for context manipulation.

Context-based decision making and adaptation The decision making on service adaptation may require manipulation of context information. The adaptation is initiated by several triggers that fire due to a change in context. The necessary information is delivered by the context repository, or it is deduced after several transformation and reasoning steps.

Component-based context manipulation A general overview of the context manipulation is given in Figure 5. The *Context Transformation* component and the *Context Reasoning* component are able to derive new facts by combining information from the *Fact Container* and the *Ontology Container*. These new facts are stored again in the *Fact Container*. The *Resource Monitor* is responsible for enabling garbage collection 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 certain necessary context information is not present or cannot be derived, then the *Context Dissemination* component is also able to send a request to third parties using the *Context Interface*. For example, by providing name and address information as possible inputs, the *Context Dissemination* component can send an information request to a *Yellow Pages* service that is able to provide mobile phone contact information.

Support for adaptation The elimination of unnecessary transformation and reasoning components results in increased storage space as the transformation rules are no longer required. If, however, context transformation or reasoning is

Table 2. Memory requirements for deploying a specific component

Component	Type	Memory
`Weather`	Sensor	6264 bytes
`Clock`	Sensor	7024 bytes
`Relevance Filter`	Retrieval	11232 bytes
`Fact Container`	Storage	23484 bytes
`Context Transformation`	Manipulation	123766 bytes
`Context Reasoning`	Manipulation	1080944 bytes

required but is too resource intensive on the current device, then the necessary facts, ontologies and reasoning can be delegated to a more powerful device.

5 Evaluation, current status and future work

The component-based management system has been implemented to a large extent (user profiling is not yet supported) on top of Draco [19], an extensible runtime system for components in Java designed to be run on advanced embedded devices. The base system supports extensions for component distribution [11], live updates [9], contract monitoring and resource management [10] and provides a unique test platform for validating the proposed concepts in a pervasive and ubiquitous computing context.

Several information providers, the storage components, and the transformation components are operational. A test case showed that the advantage of regaining storage space by eliminating old information is minimal, as the resource requirements for the libraries responsible for reasoning on OWL ontologies are much higher than the storage capacity needed for small scale services (> factor 100). A small overview of the memory usage for the deployment of several components just before their activation is given in Table 2. It demonstrates how much memory can be saved when certain components are eliminated.

Future work will focus on the modeling of resource requirements for context transformations and context derivations. 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. This work will result in a better validation of the component-based context management system, because in the current system it is difficult to define an optimal deployment without having even a rough estimate of the processing time for all context management activities.

6 Related work

Research on context-awareness has already been focusing on service adaptations in the past. The CoBrA architecture [20] is a context broker that uses ontologies and maintains a shared model of context on behalf of a community of agents, services, and devices in a certain environment. It also provides privacy protection for the users in the space by enforcing the policy rules that these users define. The difference between this and our system is that CoBrA manages the context for

all computing entities in a certain environment, instead of each device managing its own context. The advantage of our approach is that it provides better support for mobility in ubiquitous and pervasive computing environments.

Efstratiou et al. [21] also propose a service platform with support for context-aware service adaptation. The authors describe the architectural requirements for adaptation control and coordination for mobile applications. In our paper, 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 be adapted to different working conditions, including support for distributed execution.

The M3 architecture [22] is an open component-based architecture for pervasive systems that supports context management and adaptation, and includes a coordination language to coordinate system events. The context manager uses RDF to describe devices and user context. The context manager of M3 does not support context-based self-adaptation.

7 Conclusions

This paper presents a component-based approach for managing context information. The novel contribution is that the management system itself can be adapted to a device's capabilities or service requirements by enabling or disabling certain components or specific properties of certain components.

A further advantage is that these components do not necessarily have to execute on the same device. In the event of excessive storage or processing power demands, the context management system can be distributed, if necessary by relocating components to other devices.

Future work will focus on the modeling of resource requirements for the transformation and reasoning components, so that processing time can be estimated and an optimal deployment of context operations can be achieved to further increase the quality of service of our context management system.

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. 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
4. Strang. T., et al.: 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
5. Gu, T., et al.: 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)

6. 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)
7. 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)
8. 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
9. Vandewoude, Y., Berbers, Y.: Run-time evolution for embedded component-oriented systems. In Werner, B., ed.: Proceedings of the International Conference on Software Maintenance, Canada, IEEE Computer Society (2002) 242–245
10. 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)
11. Rigole, P., Berbers, Y., Holvoet, T.: Mobile Adaptive Tasks Guided by Resource Contracts. In: the 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing, Toronto, Ontario, Canada (2004) 117–120
12. Wagelaar, D.: Context-Driven Model Refinement. In: Proceedings of the MDAFA 2004 workshop, Linköping, Sweden (2004)
13. Korpipää, P., et al.: Managing Context Information in Mobile Devices. IEEE Pervasive Computing, Mobile and Ubiquitous Systems **2** (2003) 42–51
14. Indulska, J., et al.: 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
15. 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)
16. DistriNet (K.U.Leuven), EDM (LUC), ELIS-PARIS (UGent), PROG (VUB) and SSEL (VUB): CoDAMoS: Context Driven Adaptation of Mobile Services. (http://www.cs.kuleuven.ac.be/cwis/research/distrinet/projects/CoDAMoS/)
17. Senart, A., Bouroche, M., Biegel, G., Cahill, V.: Component-based Middleware Architecture for Sentient Computing. In: Workshop on Component-oriented approaches to Context-aware computing, ECOOP '04, Oslo, Norway (2004)
18. HP Labs: Jena 2 - A Semantic Web Framework. http://www.hpl.hp.com/semweb/jena2.htm (2004)
19. 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)
20. Chen, H.: An intelligent broker architecture for context-aware systems. http://cobra.umbc.edu/ (2003)
21. 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
22. 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