Towards Context-Awareness in Ubiquitous Computing

Edwin J.Y. Wei and Alvin T.S. Chan

Department of Computing, The Hong Kong Polytechnic University
{csjwei,cstschan}@comp.polyu.edu.hk

Abstract. Future ubiquitous computing has accelerated the need of context-awareness that leverages information about surrounding situation so as to adapt applications. There is considerable interest in context-awareness, and many prototypes have been proposed, which have demonstrated the potential of context-aware applications. That notwithstanding, these kinds of systems are known to be difficult to design, develop and maintain. This paper considers these difficulties as it discusses the core issues of context-aware computing, including definition of context, techniques of acquiring, modeling and adapting to contextual information. It intends to provide the community with a comprehensive and detailed view of current state of the art.

Keywords: Context-Awareness, Ubiquitous Computing, Context Definition, Context Acquisition, Context Modeling, Context-aware Adaptation.

1 Introduction

The need for context-awareness, which leverages information about surrounding situation so as to adapt applications, has been accelerated by the vision of ubiquitous computing. Computing devices in ubiquitous computing environment now exhibit a high degree of mobility and their computational systems must adapt to heterogeneous and dynamic surrounding environment where they are within. At the same time, more and more everyday devices, such as digital cameras and watches, are now equipped with computing capabilities, so that applications in ubiquitous computing environment need to take into account the attributes of different devices, which otherwise will result in unsatisfactory user experience.

Due to the interest of context-awareness, many research works have been proposed in this area [1][2][3][4][5]. These prototypes have demonstrated the potential of context-aware applications, but have also shown that designing, developing and maintaining these kinds of systems are still extremely difficult, to say the least. Lots of technical challenges remain to be addressed before even simple context-aware systems can be widely and realistically developed. In particular, every context-aware application needs to consider four basic issues: what is context, how to acquire them, how to represent them, and how to adapt to them. This paper discusses these core issues of context-aware computing, and

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 706–717, 2007.

intends to provide the community with a comprehensive and detailed view of current state of the art.

The remainder of this paper is organized as follows: In section 2, we review definitions of context and present our own definition from an application point of view. Section 3 focuses on currently available context acquisition techniques for context-aware applications. Section 4 discusses several basic technical aspects of context models including data structure, integrity and manipulation. In section 5, we detail various design concerns of context-aware adaptation. Section 6 concludes and summarizes related challenges in context-aware research.

2 Definitions of Context

In order to effectively utilize contexts, we should first understand what contexts are. Researchers in the context-aware computing community have invariably offered their own definitions of context based on their research background. Schilit and Heimer [6] first introduced context-aware computing in 1994 and set three parameters for contexts: the software's location of use, the collection of nearby people and objects, and changes to those objects over time. For a long time, contexts are defined by enumerating examples or choosing synonyms [7][8][9][4]. In 2000, Dey and Abowd [10] proposed a more generic definition:

> "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

More recently, inspired by social sciences, some researchers argue that contexts are socially and psychologically constructed outcomes of human activities rather than stable, and objective sets of features that externally characterize activities [11][12]. Oulasvirta *et al.* [13] summarize these definitions of context, and divide them into two camps which are realism and constructivism. Realism posits contexts as existing ontologically, and that they can be correctly recognized and adapted to if properly instrumented and programmed. Whereas, constructivism recognizes that contexts are human creations, social and psychological, and that people should be provided resources to create and maintain these contexts.

No matter realism or constructivism, in their answers to "what is context", user interaction with applications is overly emphasized. Actually, many non-interactive programs can also make use of surrounding situation. We argue that considering the issue of context definition from an application point of view will be a more reasonable way to help developers to construct context-aware applications. With this in mind, we present here our own definition of context:

Context is application specific, and context of an application is any external information which can be utilized to adapt the data, behavior or structure of this application.

Our definition of context indicates three important features of context. First of all, context is application specific. Contexts of one specific application may make

no sense to others. Secondly, context is external to applications. Only information outside one application can be regarded as the context of that application. Finally, context can be used to adapt not only behavior of applications, but also their data, or even structures.

3 Acquiring Contexts

In practice, the information from three categories of context may be used to adapt applications, which are physical contexts, computing contexts and user contexts. Physical contexts are physical circumstances of an application like noise level and temperature. Computing contexts refer to an application's own execution conditions such as host computing resources, available peripheral devices, network capacity and connectivity, and so on. Finally, user contexts are anything regarding users such as their location, presence, identities, abilities, activities, etc. The vision of future ubiquitous computing paradigm requires these contexts to be captured without user interaction. In this section, we review various techniques that have been used to implicitly acquire these contextual information.

3.1 Acquiring Physical Contexts

Physical contexts are primarily obtained through specially designed physical sensors, which convert physical properties or phenomena, such as light and noise, into a corresponding measurable data. For example, the TEA project [4] uses photodiode, accelerometers, and other sensors to measure various contextual information such as light level, tilt and vibration, proximity of humans or other heat-generating objects, and so on. Further sensor progress in the development and manufacture of sensors will allow more and greater varieties of physical contextual information to be captured and used. Table 1 provides a summary of some common types of sensors for context-aware applications.

3.2 Acquiring Computing Contexts

Computing contexts are normally collected by software routines. Most mainstream operating systems provide primitives for developers to get related runtime information about the device hardware, and reduce the work involved in development. For example, in most Unix/Linux operating systems, one can employ commands like iostat, vmstat, and netstat to report statistics for I/O devices, virtual memory, and network condition. Symbian OS also provides a comprehensive software development toolkit for developers to monitor CPU, memory and storage usage, etc. In addition to using OS primitives, it is also possible to leverage user-level modules to sense computing contexts. For instance, applications can obtain device management information of most network devices like routers and firewalls by sending SNMP requests, and applications base on [14] or [15] can be notified of change of network bandwidth via upcalls.

Table 1. Common Types of Sensors

Type of Sensor	Context Sensed	Sensor Examples
Temperature Sensor	Temperature	Thermometer, Thermocouple, Thermistor
Heat Sensor	Heat	Bolometer, Calorimeter
Magnetism Sensor	Orientation	Magnetic Compass, Fluxgate, Compass
Pressure Sensor	Altitude, Atmospheric Pressure, Speed	Altimeter, Barometer
Gas/Liquid Flow Sensor	Velocity of the Wind, Rate of Fluid Flow	Anemometer, Mass Flow Sensor
Mechanical Sensor	Acceleration, Position, Angle, Deformation	Acceleration Sensor, Position Sensor, Selsyn
Chemical Sensor	Proportion of Gas	Carbon Monoxide Detector, Ion-Selective Electrode
Light Sensor	Light	Phototubes, Photodiode
Sound Sensor	Audio	Microphone, Hydrophone, Seismometer
Motion Sensor	Speed, Acceleration	Radar Gun, Speedometer, Odometer
Orientation Sensor	Orientation	Gyroscope, Artificial Horizon

3.3 Acquiring User Contexts

It is difficult to directly sense user context through dedicated hardware sensors or software routines. Rather, they have to be derived from original measurements by software programs where original measurements are processed collectively. In the followings, we discuss in some detail on the techniques used to acquire user contexts especially location information.

Location. The most commonly acquired attribute of user context is location. In this section, we describe four most frequently used techniques for the acquisition of location information, which are trigonometry, signature matching, cellular proximity and computer vision.

Trigonometry, including trilateration and triangulation, leverages the geometry of triangles to determine the positions of objects. Trilateration utilizes the measured distance between the subject and three or more reference points, as well as the known locations of these reference points, to compute the subject's location. Differently, triangulation uses angle measurements and at least one known distance to complete the computation. In order to measure the required distances and angles for trigonometry, time of arrival/time difference of arrival (TOA/TDOA) [16][17], received signal strength (RSS) [18] and angle of arrival (AOA) [19] of various communicational signals are most frequently used. Trigonometric approaches are fine-grained localization techniques, frequently used outdoors. However, due to rough wall surfaces and obstacles

between emitters and receivers, communication signal propagation in indoor environment suffers from multipath, non-line-of-sight (NLOS), and local shadowing, which result in unreliable measurements of location metrics such as TOA, TDOA and AOA. Therefore trigonometric approaches fail to provide adequate location accuracy indoors.

The positioning process based on signature matching consists of two phases: an off-line phase of collecting data, and a real-time phase of inferring the users' location [20]. In the off-line phase, necessary information of the entire zone of interest is collected to produce signatures, and the latter are then stored as a function of user's location. In the real-time phase, incoming data is analyzed and compiled into a unique signature which is compared with the recorded signatures to identify the closest record and then the location is inferred. In order to produce a unique signature for each location, several types of communicational signal information such as received signal strength (RSS), signal noise ratio (SNR), angular power profile (APP) and power delay profile (PDP) can be utilized. Another interesting information frequently used to construct distinct signatures is the ground reaction force (GRF) gathered by pressure sensors. GRF refers to the reaction force supplied by the ground in response to the weight and inertia of a body exerted on the ground. For example, the Smart Floor [21] sets load cells under floor tiles to gather GRF profile, and choose ten profile features, including the mean value, standard deviation, length of the profile and so on, to use as signatures for each GRF profile. Signature matching approaches can effectively counteract the problems of signal propagation in indoor environment. The major drawback of these approaches is that developers have to collect a great quantity of data to generate the signature database. Furthermore, changes to the environment may require reconstruction of the predefined dataset or retrieval of an entirely new dataset. Consequently, it is not suited for ad hoc deployment scenarios [22].

Cellular proximity location sensing techniques determine the location of a subject when it is near a known access point. In a cellular network, each fixed access point, with known location, owns its sensing cell. Whenever the subject enters the cell, it is sensed by the access point, and its location is therefore pinned down to the resolution of a cell. Cellular proximity approaches can be used both indoors and outdoors, and also requires no collection of off-line data. However, they are coarse-grained localization techniques. Since the subject's location information is sensed by judging whether the subject is in the range of an identified area, the sensing accuracy is determined by the radius of the identified area. Moreover, they also incur significant installation and maintenance costs. Using cellular proximity techniques, the cellular network must provide thorough coverage through adequate placement and density of access points.

Location information can be also derived from analysis of data from visual images. Visual processing techniques like depth and color segmentation, blink detection, and color histogram matching can be used to analyze visual streams from cameras, and recognize one or several objects, together with their 2D positions in the image or 3D positions in the scene. By combining these positions

with knowledge of camera's relative location, fields of view, and heuristics on the movements of objects, the final location of objects can be computed. For instance, Microsoft's Easyliving [23] uses two color stereo cameras each connected to a PC to track multiple people in a living room. Vision based location sensing techniques are the most flexible approaches. They can be used either indoors or outdoors, and do not require any sort of devices to be worn by users. However, as scene complexity increases and more occlusive motions occur, more works have to be done to maintain analysis accuracy [24].

Other User Contexts. Other user contexts can also be acquired via a number of novel ways. MIT's Office Assistant [25] uses pressure sensors to detect visitors. Schmidt *et al.* [26] make use of load sensing technique to explore more pervasive augmentation of surfaces in everyday environment including floors, tables and other high interactive spaces, and from these load-sensitive surfaces extract three context primitives: weight, position and type of interaction. Moore *et al.* [27] measure image-, object-, and action-based information from videos to recognize human activities such as reading, coffee break and washing dishes, and objects like winding road and parking car.

4 Modeling Context

Context modeling is concerned with representing, structuring and organizing contextual data and relationships between them, in order to facilitate the storage and operations of them. A well-designed context model needs to consider three basic technical aspects: data structure, integrity and manipulation.

4.1 Data Structure

The underlying data structure used to exchange context information inside and between applications is the first basic issue for context models. An appropriate data structure for contextual information will facilitate not only the representation of contextual data, but also their storage, validation, modification, retrieval, and even reasoning. Currently, tuples, objects and markup schemes are three most popular data structures used to represent contextual information.

The simplest structure to represent contextual information is key-value pair (2-tuple). Every pair describes one aspect of the surrounding situation of an application. A set of these pairs describes the whole environment where applications are actively deployed. Early works in context-awareness, [28] for instance, frequently used tuples as the underlying context structures. Tuples are easy to implement and manage. Most programming languages provide direct support to construct tuples. For example, as a fundamental data type, Lisp provides list, which is a finite ordered sequence of elements. Eiffel also has a built-in type of tuple. However, tuples lack structure and formality. As a result, they can not effectively express sophisticated contextual information and relationships.

Contexts can also be modeled as a set of related objects. Contextual information is embedded as the states of these objects, accessed and modified through

accessor and mutator methods. Object-oriented context models' encapsulation and reusability cover parts of the problems arising from the dynamics of contexts [29]. The major drawback of object-oriented models is that context objects are programming language dependent, which will affect their portability among different applications and platforms. Although there do exist some techniques advocating language- and platform- independent implementation, in this area much work remains to be done.

Various markup languages can be also used to model context data, which use a hierarchical data structure consisting of markup tags with attributes and content. Among them, XML schema languages, like DTD, XML Schema and RELAX NG, are the simplest ones. We can also model contexts using other more expressive and formal data representation markup languages. Resource Description Framework (RDF) is such an alternative. Using RDF, contexts are modeled as a set of statements about resources and can be exchanged between applications without loss of meaning. RDF also provides a vocabulary description language, RDF Schema (RDF-S), to help modeling not only structures of related resources, but also relationships between them. The capability of modeling context relationships enables context-aware applications to reason with contextual information. Another powerful data modeling markup language is the Web Ontology Language (OWL), which provides more vocabularies for expressing meaning and semantics than XML, RDF, and RDF-S. Using OWL, contextual information is modeled as a set of ontologies for specific application domains. Markup scheme context models can be used to represent complex contexts and relationships like object-oriented models. Moreover, they enable a high degree of context sharing. Contextual information can be share among different applications across different platforms. However, markup based syntax is redundant or large compared to binary representations of similar data.

4.2 Integrity

A well-designed context model needs to support the validation of structure and data integrity of contextual information. In an extra dynamic environment, contextual information may be partially lost and become incomplete, so they must be validated before used. Integrity validation can be performed at two levels: structure and data. The purpose of structure validation is to check whether the information acquired is complete in structure. For example, whether they have a predefined ending or the necessary components exist. Data validation is a more meticulous examination and attends to the data type, range, and even semantics of the information to be validated. What kinds of integrity validation a context model can support, to some extent, depends on the data structure it employs. When using tuples as the underlying data structure, limited structure integrity validation can be applied due to their lack of structure and formality; With the help of compilers and programming interfaces, object-oriented models can be validated in terms of context structure and data integrity; For markup scheme context models, there exist many tools and specifications available for validating structure and data integrity.

4.3 Manipulation

A comprehensive context model also needs to define operators which can be applied to the data structure. Except for the basic C.R.U.D. operations, another especially important operator for contextual data is context reasoning. Effective context reasoning can introduce more new contextual information derived from other types of contexts to improve user experience. For example, given contextual information like Jack's location is in the presentation room, his posture is sitting and the projector is on, the application may infer that Jack is now involved in a presentation, and automatically turn Jack's cell phone to vibration mode. Furthermore, context reasoning helps to resolve context inconsistency and conflict, and provide more exact context query results. Similar to integrity validation, context manipulation also depends on the underlying data structures of context models. For example, tuple-based contexts are easy to modify, however tuples support only simple context queries and reasoning, and are not easy to use with other efficient context retrieval and reasoning algorithms. By contrary, there are many techniques available for querying and reasoning with markup context information, but markup schemes suffer from its operational difficulties.

5 Adapting to Contexts

After acquiring raw contextual data, representing fine-grained contexts, context-aware applications have to struggle with the issue of adaptation. In particular, developers need to consider three adaptation-related questions: what to adapt, how to adapt, and when to adapt.

5.1 What to Adapt

There are three kinds of adaptations which applications may use in order to adapt to contexts: data adaptation, behavioral adaptation and structural adaptation. By data adaptation we mean that applications may change their operating data in some ways such as changing the quality of data to be accessed, transforming data form to a more suitable one, or accessing a different set of data. For example, in Odysssey, the server can select the most appropriate data at runtime from several pre-generated versions with different fidelity level according to the available resources or even energy [30]. By behavioral adaptation we refer to the fact that applications may behave differently in different contexts. For instance, a context-aware video player may pause when the audience leave and resume when they return. By structural adaptation we mean that applications may modify their internal structures or processing sequences to counteract contexts yet retain the same functions. For example, to adapt to poor computing resource, a compiler may unload the code optimization module.

5.2 How to Adapt

Two general approaches have been used to realize context-aware adaptation: transformational adaptation and compositional adaptation [31]. In transformational

adaptation, applications directly modify related specifications and/or implementations to suit changing contexts. Compositional adaptation, in contrast, does not directly modify. Rather, it responds to contexts through adding, removing, replacing, or even changing the interconnections of application algorithmic or structural parts. For example, to adapt web contents to resource-constrained devices, applications may directly transcode the original data by transformational adaptation, or replace the original data with a new version by compositional adaptation.

To use transformational adaptation, some crucial variables to describe context-aware aspects of applications are usually defined and tuned to adapt to contextual information. Although transformational adaptation is easier to implement than compositional adaptation, it suffers from two major drawbacks: first, it is hard to realize structural adaptation, which usually requires adding or removing structural parts of applications. Moreover, unimplemented versions of data and behavior cannot be introduced into applications during runtime. In transformational adaptation, everything should be designed and implemented in advance.

Compositional adaptation is more flexible. It is applicable for all types of adaptation, including data adaptation, behavioral adaptation and structural adaptation, and it allows new data, behavioral or structural parts to be adopted to address unforeseen concerns after the original construction of applications. On the downside, compositional adaptation is more complex than transformational adaptation. Developers need to pay attention to synchronization and state migration between different components [32]. For example, in the case of a streaming player which can replace various encoders and decoders using various algorithms, if the encoder at the sender side is replaced before the decoder is replaced at the receiver side, the video content may be decoded incorrectly. Thus, synchronization among components is required while adding, removing or replacing components. At the same time, in order to keep state consistence, when replacing components, state migration is necessary to correctly initialize the new version of a component.

5.3 When to Adapt

Context adaptation may occur throughout almost the entire lifetime of an application, from compile time, to load time, to run time. Generally speaking, later adaptation time supports more powerful adaptation methods, but also complicates the problem of ensuring consistency in adapted programs [33]. In compile time, context-aware applications can be adapted to contexts, especially computing contexts, with the help of compilers. For example, programs are expected to use various compiler flags for various target machine models when using GNU Compiler Collection (GCC). Another example is aspect-oriented programming (AOP). In AOP, during compilation, an aspect weaver can be used to weave different crosscutting concerns of the program together to form a program with new behavior. Context-aware applications can also defer the adaptation decision until the application load time, enabling developers to configure applications in respond to current environment after compile time. The simplest load time

adaptation implementation is to use command-line parameters. Another more flexible way is to use external configuration files. Apart from these two general methods, there are particular languages that provide primitives to support load time adaptation. For example, using Java, developers can design their own class loaders, and decide which classes are to be loaded according to current contexts, hence adapting applications. The most powerful adaptation takes place in application run time so that context-aware applications can be dynamically adapted while it is being used. Examples of techniques to implement runtime adaptation include computational reflection and dynamic weaving of aspects.

6 Challenges for Context-Awareness in Ubiquitous Computing

We have discussed the core issues of context-awareness in ubiquitous computing. During this process, we can observe that research into context-aware computing is still at its early stage and there exist several central research challenges in context-aware computing.

First of all, there is still a fundamental lack of understanding of context. Many definitions are available but none is satisfactory to most people. As a result, almost all current context-aware applications are built on their own understandings to contexts. This case makes it impossible to exchange context information between different applications. Secondly, more contexts wait to be sensed and used. Currently, most of the focus in context-aware research is on location and lots of location-sensing approaches have been explored but there have been few attempts to enhance applications by applying other contextual information, such as human emotions and activities. Finally, more comprehensive context-aware middlewares need to be provided. Early context-aware research efforts directly interact with the underlying network and operating systems to extract contextual information, process it, and adapt to it entirely at the application layer. These approaches were hard to be reused, and application developers have to "re-invent the wheel" each time a new context-aware application being developed. Moreover, the complexity of communicating with various sensors, modeling and reasoning raw contextual data, and adapting application behavior tends to distract developers from implementing real application logic, and this will slow down productivity and compromise product quality. Mechanism and primitives must be provided to conceal the complexity of these issues from context-aware application developers, and to facilitate the development process. However, although most of the existing middlewares provide software abstraction and management facilities for sensor devices, they do not provide off-the-shelf context acquisition components or underlying supports for communication with sensors. Additionally, the task of context-aware adaptation is seldom addressed at the middleware layer and just left as an application concern, and few of the works in context-aware middlewares support all needs of context acquisition, modeling, and adaptation.

References

1. Want, R., Hopper, A., Falcao, V., Gibbons, J.: The Active Badge Location System. ACM Transactions on Information Systems 10(1), 91–102 (1992)
2. Abowd, G.D., Atkerson, C.G., Hong, J., Long, S., Kooper, R., Pinkerton, M.: Cyberguide: A Mobile Context-Aware Tour Guide. Wireless Networks 3(5), 421–433 (1997)
3. Cheverst, K., Davies, N., Mitchell, K., Friday, A., Efstratiou, C.: Developing A Context-Aware Electronic Tourist Guide: Some Issues and Experiences. In: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 17–24 (April 2000)
4. Schmidt, A., Aidoo, K.A., Takaluoma, A., Tuomela, U., Laerhoven, K.V., de Velde, W.V.: Advanced Interaction in Context. In: Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing, pp. 89–101 (1999)
5. Siewiorek, D., Smailagic, A., Furukawa, J., Krause, A., Moraveji, N., Reiger, K., Shaffer, J., Wong, F.L.: SenSay: A Context-Aware Mobile Phone. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, Springer, Heidelberg (2003)
6. Schilit, B.N., Heimer, M.M.: Disseminating Active Map Information to Mobile Hosts. IEEE Network 8(5), 22–32 (1994)
7. Schilit, B.N., Adams, N., Want, R.: Context-Aware Computing Applications. Mobile Computing Systems and Applications, 85–90 (1994)
8. Long, S., Kooper, R., Abowd, G.D., Atkeson, C.G.: Rapid Prototyping of Mobile Context-Aware Applications: the Cyberguide Case Study. In: International Conference on Mobile Computing and Networking, pp. 97–107 (1996)
9. Brown, P.J., Bovey, J.D., Chen, X.: Context-Aware Applications: From the Laboratory to the Marketplace. IEEE Personal Communications 4(5), 58–64 (1997)
10. Dey, A.K., Abowd, G.D.: Towards a Better Understanding of Context and Context-Awareness. In: CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness (2000)
11. Dourish, P.: What We Talk About When We Talk About Context. Personal and Ubiquitous Computing 8(1), 19–30 (2004)
12. Tamminen, S., Oulasvirta, A., Toiskallio, K., Kankainen, A.: Understanding Mobile Contexts. Personal and Ubiquitous Computing 8(3), 135–143 (2004)
13. Oulasvirta, A., Tamminen, S., Höök Comparing, K.: Two Approaches to Context: Realism and Constructivism. In: Proceedings of the 4th Decennial Conference on Critical Computing: Between Sense And Sensibility, pp. 195–198 (2005)
14. Noble, B.D., Satyanarayanan, M., Narayanan, D., Tilton, J.E., Flinn, J., Walker, K.R.: Agile Application-Aware Adaptation for Mobility. In: Proceedings of the 6th ACM Symposium on Operating Systems Principles, pp. 276–287 (1997)
15. Andersen, D., Bansal, D., Curtis, D., Seshan, S., Balakrishnan, H.: System support for bandwidth management and content adaptation in Internet applications. In: Proceedings of 4th Symposium on Operating Systems Design and Implementation, pp. 213–226 (October 2000)
16. Ward, A., Jones, A., Hopper, A.: A New Location Technique for the Active Office. IEEE Personal Communication 4(5), 42–47 (1997)
17. Harter, A., Hopper, A., Steggles, P., Ward, A., Webster, P.: The Anatomy of a Context-Aware Application. Wireless Networks 8(2-3), 187–197 (2002)

18. Small, J., Smailagic, A., Siewiorek, D.P.: Determining User Location for Context Aware Computing Through the Use of a Wireless LAN Infrastructure, Project Aura Report, Carnegie Mellon University (2000), http://www.cs.cmu.edu/~aura/publications.html
19. Aitenbichler, E., Muhlhauser, M.: An IR Local Positioning System for Smart Items and Devices. In: Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops, pp. 334–339 (May 2003)
20. Nerguizian, C., Despins, C., Affes, S.: Geolocation in Mines With an Impulse Response Fingerprinting Technique and Neural Networks. IEEE Transactions on Wireless Communications 5(3), 603–611 (2006)
21. Orr, R.J., Abowd, G.D.: The Smart Floor: A Mechanism for Natural User Identification and Tracking. In: Proceedings of International Conference on Human Factors in Computing Systems, pp. 275–276 (2000)
22. Bulusu, N., Heidemann, J., Estrin, D.: GPS-less Low-Cost Outdoor Localization for Very Small Devices. IEEE Personal Communication 7(5), 28–34 (2000)
23. Krumm, J., Harris, S., Meyers, B., Brumitt, B., Hale, M., Shafer, S.: Multi-Camera Multi-Person Tracking for EasyLiving. In: Proceedings of the 3rd IEEE International Workshop on Visual Surveillance, pp. 3–10 (July 2000)
24. Hightower, J., Borriello, G.: Location Systems for Ubiquitous Computing. IEEE Computer 34(8), 57–66 (2001)
25. Yan, H., Selker, T.: Context-Aware Office Assistant. In: Proceedings of the 5th International Conference on Intelligent user Interfaces, pp. 276–279 (2000)
26. Schmidt, A., Strohbach, M., van Laerhoven, K., Friday, A., Gellersen, H.W.: Context Acquisition Based on Load Sensing. In: Proceedings of the 4th International Conference on Ubiquitous Computing, pp. 333–350 (2002)
27. Moore, D.J., Essa, I.A., Hayes, M.H.: Exploiting Human Actions and Object Context for Recognition Tasks. In: Proceedings of IEEE International Conference on Computer Vision 1999 (ICCV 1999) (March 1999)
28. Schilit, B.N., Theimer, M.M., Welch, B.B.: Customizing Mobile Applications. In: Proceedings of USENIX Symposium on Mobile and Location-Independent Computing (USENIX Association), pp. 129–138 (August 1993)
29. Strang, T., Linnhoff-Popien, C.: A Context Modeling Survey. In: 1st International Workshop on Advanced Context Modeling, Reasoning and Management, pp. 34–41 (2004)
30. Flinn, J., Satyanarayanan, M.: Energy-aware Adaptation for Mobile Applications. In: Proceedings of the 17th ACM Symposium on Operating System Principles, pp. 48–63 (December 1999)
31. Tekinerdogan, B., Aksit, M.: Adaptability in object-oriented software development workshop report. In: Cointe, P. (ed.) ECOOP 1996. LNCS, vol. 1098, Springer, Heidelberg (1996)
32. Biyani, K.N., Kulkarni, S.S.: Building Component Families to Support Adaptation. In: Proceedings of the 2005 workshop on Design and evolution of autonomic application software (DEAS 2005), St. Louis, Missouri, USA (May 2005)
33. McKinley, P.K., Sadjadi, S.M., Kasten, E.P., Cheng, B.H.C.: Composing adaptive software. IEEE Computer 37(7), 56–64 (2004)