

Context-aware Mobile Computing

D. Schall, C. Dorn, and S. Dustdar

Distributed Systems Group, Vienna University of Technology, Austria

Keywords: *Hierarchical Context Model, Mobile Teams, Team Context, Context Sharing, Interaction Patterns*

Definition: *Context-aware computing in mobile collaborative working scenarios fosters collaboration by establishing team awareness and thus supports users in their team interactions regardless of device type.*

In the past, research in context-aware mobile computing and applications mainly focused on location-based systems. While location is a crucial and vital part of context information, it's only a subset thereof. Mobile context systems need to utilize more than just spatial information. Any information on the environment relevant to the user's task can serve as context data. This information covers not only nearby people, people in the same project, their activities, and their availability in terms of communication capabilities but also includes technology related facts such as wireless link quality, remaining battery lifetime, installed software, and services within a distributed workflow. A wide-spread definition of context in the area of computer science is by Dey and Abowd [1]:

“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.”

While this definition is rather generic, any more detailed specification would be domain or application dependent.

Mobile Context-aware computing has to deal with the same challenges as traditional context support systems. However, in addition to context modeling, processing, storing, privacy, uncertainty, or information aging issues, in mobile environment device constraints such as battery lifetime, processing power, available memory, or bandwidth as well as disconnections between nodes need to be at the center of attention. These requirements are matched with existing distributed systems technology such as peer-to-peer networks, agent systems, sensor networks, and regular client-server models.

So far, most mobile context-aware applications have focused solely on a number of independent users interacting with the system [2]. Also mobile context frameworks such as by Costa et al. [3], Sørensen et al. [4], or Biegel and Cahill [5] lack this support. However, context information is of even greater benefit if we extend the scope of such applications to groups, especially collaborative working teams, as we will discuss in the following sections.

Context in Distributed Teams

We observe a trend towards new emerging team forms [6] where team members are increasingly engaged in ad-hoc collaboration and a growing number of people are equipped with mobile devices. New team configuration require accelerated responsiveness, fast software configuration, and a flexible communication infrastructure that changes according to their work activities. In our research we consider three team forms: a) nimble teams (N-teams), which collaborate in short timeframes, engage in work activities, and then dissolve again, as the circumstances require. Virtual teams (V-teams) require and enable people to collaborate across geographically disparate locations, organizational boundaries and have a somewhat stable team configuration. Finally, nomadic teams (M-teams) allow people to work from home, while on the move, or in flexible office environments. These new team forms and collaboration among them ultimately leads to challenging and new requirements with respect to the software infrastructure. This is especially true in a mobile team context, where issues such as presence awareness, location awareness, and knowledge sharing may

imply tight requirements for the underlying access network technologies and personal devices in use (e.g., PDAs, smart phones, tablet PCs, laptops, etc.).

To highlight characteristics of team forms we group team idiosyncrasies and specific challenges in four views: spatial, organizational, project, and human interaction (Figure 1). Each view establishes information regarding the overall team context. Some views have high or low importance depending on particular team configuration.

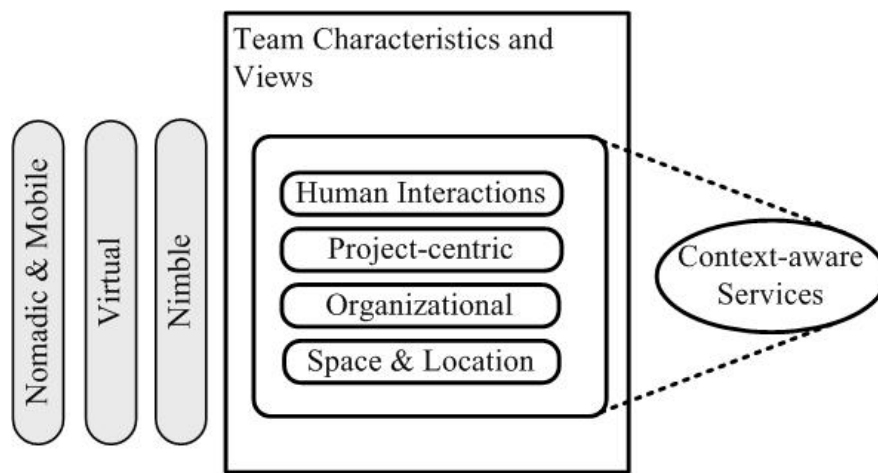


Figure 1. Team centric context views.

- **Spatial View.** The spatial view shows geographical distribution of team members and location dynamics that can be observed at various granularities. At *large scale* space geometric properties are mostly expressed by point or area based representations. The GPS system is an example for point based location around the globe. At *metropolitan* area scale we typically find geometric and symbolic models. Symbolic models describe location and space in terms of names and abstractions (i.e., hierarchies). Unlike geometric models, humans, computational entities and services can understand this model. In dense metropolitan areas location estimation techniques based on GSM traces can be used (i.e. point/area based location). *Indoor* location modeling (e.g. building and

offices) employs symbolic and semantic view models. Semantic models do not focus purely on location but also attempt to express relationships of entities in space, between spaces (e.g. floor plan), and proximity of entities within the space.

Various spatial models and techniques for location estimation can be used in combination; however, we argue that teams require location information at some spatio-temporal granularity depending on their current team configuration.

- **Organizational View.** In midsize or large organizations, human activities are commonly structured in form of organizational hierarchies. Individuals are organized into groups, sections and departments. An organizational layer defines one central point of authority (e.g., department head), with multiple groups beneath it. Hierarchies in large organizations provide an efficient way to propagate message from a small number of sources down to a large number of receivers (e.g., distributing messages such as memos to workers and employees), but are inefficient in providing a return path for information from a significant number of people. A hierarchical structure has naturally its problems with allowing “sideways communication” and may render inefficient communication within and across teams. Strictly hierarchical systems without feedback or cross-level communications may be fragile and inefficient in many settings, thus we aim at finding suitable methods for cross-level communication and information sharing by looking at different team forms and interaction patterns.

- **Project-centric View.** The project view, in a traditional sense, aims at organizing and managing resources and artifacts. The management of such a project includes the definition of scope of required work, planning and allocation of resources, assignment of responsibilities, and monitoring of constraints such as time, costs, and risks. The output of a project could be a service, artifacts, or intellectual property that reflects the intellectual endeavor carried out by the team. In contrast to permanent operations, which reproduce the same product or service repeatedly (e.g. structured

process), the project is of temporary nature and thus can be seen as a “one-time” activity (see link 1).

We note that time of existence of particular teams may greatly vary depending on team form in question, so organizing a set of activities in form of a project workspace requires software services to adapt according to specific team requirements (e.g., by simplifying setup phase or enabling tightly coupled collaboration).

- **Human Interaction View.** Human interactions in collaboration can be seen as patterns that represent reoccurring situations. An interaction pattern describes the sequence and type of interactions between individuals (actors in an interaction). An interaction can be spawned by an activity or task which may have been defined within the scope of a project. Ideally in teams, it should be possible to continuously form and destroy ad-hoc associations to other team mates or even across teams by means of communication channels in order to react upon changing requirements (e.g. due to emerging problems) or disseminate information regarding the progress of a particular task. However, most collaborative software tools in use today do not support such dynamic/peer-to-peer binding of information sources in form of links as they predominantly rely on client-server architectures.

Interaction Context

Patterns in human collaboration are reoccurring interactions between human actors. Such patterns may impact a team, in terms of being aware of dynamics, status updates, etc., as well as the measures that can be taken to support collaboration. A communication pattern in human collaboration shows how distributed teams exchange information by means of synchronous or asynchronous communication channels. Whether or not individual interactions have great or only limited impact on a team is delimited by the scope of the interaction and the different roles in a team. This scope is given by the interaction context.

By inspecting the scope of interactions we look at what can be said about the impact of individual interactions on the team as a whole. In other words, to what degree should the whole team be aware of interactions between two team members or of interactions within sub-groups. Figure 2 shows an abstract collaboration pattern – the Master/Slave pattern [7], which can be interpreted as a metaphor for a boss/assistant or supervisor/employ relation in a working environment.

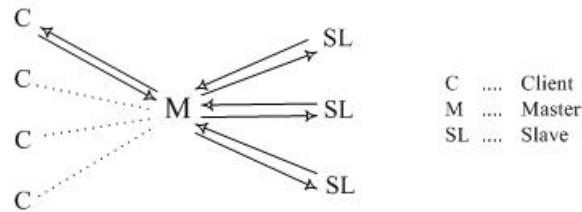


Figure 2. Master-Slave pattern in human collaboration [7].

A Client C interacts with the Master M who has the ability to split and delegate tasks to his Slaves SL. The Master may assign a single task to multiple Slaves (redundancy) or distribute task according to context information and status of Slaves. Slaves process these tasks and return the result to the Master. The Master may then need to aggregate received input and respond to the Client C by computing a final result. It becomes apparent that Master/Slaves need to be tightly coupled as workload needs to be distributed efficiently. Figure 3 shows how to apply such an abstract pattern to a concrete collaboration scenario.

Team-awareness trough Context and Collaboration Patterns

In Figure 3 we see multiple actors communicating with each other. A new collaboration is kicked off by an Initiator, for example by issuing the request 1 to the Costumer Support Service. A Broker gets the pending request 2 and needs to find a staff member who is able to process that request. We assume a scenario where Field Staff, e.g., technicians inspecting machines at costumer sites – Member I and II, need to be able to process such customer requests while being mobile. The Broker

uses the Contact Team Member Service (3), which interacts with the Context Sharing Platform (CSP, 4). The CSP allows to store and share hierarchical context information (a detailed description of the CSP will be given in following sections).

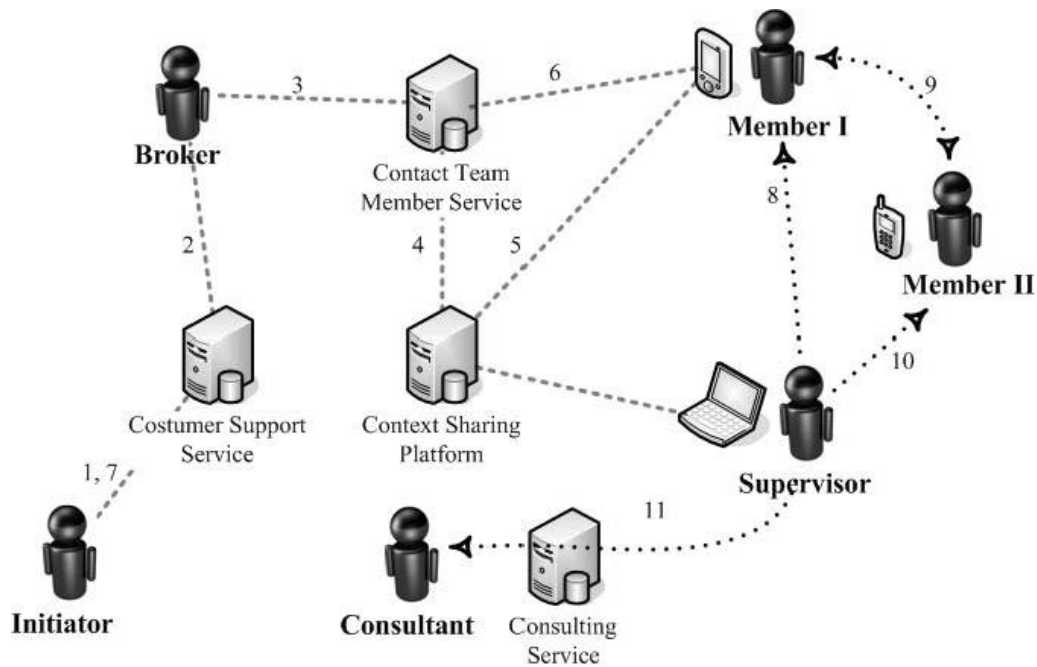


Figure 3. Collaboration and interaction in (mobile) teams.

Context information such as location (e.g., in form of GPS coordinates or GSM based location), availability, current work-load, and a time-schedule indicating pending work-items (e.g., ToDo lists) and deadlines is made available through the platform (flow 5). This context information is utilized to find and select the right team member according to criteria such as “nearest location” and ability to process the customer’s request (i.e., availability, work-load, no pending request that prevent the member for processing a new request).

Team members working in the field are typically monitored by a Supervisor (flow 8). A Supervisor may want to view Member I’s work context such as requests processed during the day, completed, deferred or cancelled requests or context information such as current/past location at a high level.

In this example, the Supervisor and Member I stand in a Master/Slave relationship. The Supervisor may delegate or assign tasks or advice on specific problems that Member I might face.

Another facet of the collaboration scenario in Figure 3 is the interaction between Member I and Member II (flow 9), which can be realized mobile Web service [14]. An interaction may happen opportunistically by triggered context events such as geographical proximity or context based queries. Member I may need to find a team member in proximity that is able to, for example, assist in work on a given task. However, this interaction should also be made visible to the Supervisor. Thus, the CSP allows handling such situations by subscribing to context information of third parties [13], i.e., Member II depicted by flow 10. The supervisor is now able to participate in the interaction, initiated by Member I and II, by receiving task related (or collaboration related information in general), if desired.

Flow 11 in Figure 3 depicts an interaction between the Supervisor and a Consultant. We define *Consulting* as a collaboration pattern with following properties:

- Consultants provide their expertise in a specific, usually narrow, field where they have deep levels of expertise (it can be assumed that their set of skills is well defined).
- Consultants are not “core” members of teams. They join a team temporarily, provide their services or expertise and leave the team once the job is done.
- Consultants communicate through well defined interfaces - the Contractor (e.g., the Supervisor in our example).
- Consultants offer and provide their services as long as their services are demanded or at least as long as commitments exist. Hence, services can be published in form of Consulting Services and revoked at a later point.

- Consultants may work on multiple tasks/projects simultaneously. However, as opposed to the Master/Slave pattern where a Master has full control over the Slave's assignments, a Contractor has only status information regarding the Consultant's tasks that are executed in the given contract.

Teams that characterize such properties are best defined as *Nomadic Teams* [6]. The CSP accounts for such a case by providing the ability to tunnel context information or collaboration specific information through intermediate nodes (e.g., Member I - Supervisor - Consultant). For example, artifacts that are provided by Member I and which need to be commented or reviewed are shared with the Supervisor directly. The Supervisor may want to share relevant parts of those artifacts with a Consultant, and return back comments and annotations made by the Consultant to Member I. A tight coupling (i.e., team awareness) in form of context information such as status or task progress that needs to be exchanged is only needed between Member I and the Supervisor, and the Supervisor and the Consultant.

Context Modeling

How to model context greatly depends on the application domain. In the area of Collaborative Working Environment (CWE) members in emerging team forms need awareness support of their distributed and mobile peer. At the same time, these members are part of several teams, switching between activities related to different projects in which they take on different roles. Consequently, each peer requires relevant information coming from multiple independent sources. As a dynamic environment generates continuous updates of context information that would flood team members with irrelevant information, those peers require a mechanism to define what the information is that is relevant in their current situation and at what level of detail. Moreover, peers must be able to base their decisions on reliable context information. As context is intrinsically uncertain to some degree, a fallback mechanism has to be provided, if confidence of data is too low. In the end, these requirements need to be reflected in the context model.

Having to rely purely on mobile devices from time to time, handling of the context model needs to be flexible and yet light-weight. Strang and Linnhoff-Popien [8] have evaluated a number of modeling techniques in terms of distributed composition, partial validation, richness and quality of information, incompleteness and ambiguity, level of formality, and applicability to existing environments. In their analysis, ontology based models and object-oriented models dominated in most categories over logic based models, key-value models, markup scheme models, and graphical models.

We designed a hierarchical model, which can be seen as a mixture of an ontology based and an object-oriented model. The idea behind a hierarchy is the structuring of context information according to levels of details [9]. An exemplary location hierarchy is visualized in Figure 4. The ASC model proposed by Strang [10] has similar properties, but is situated at a conceptually higher level.

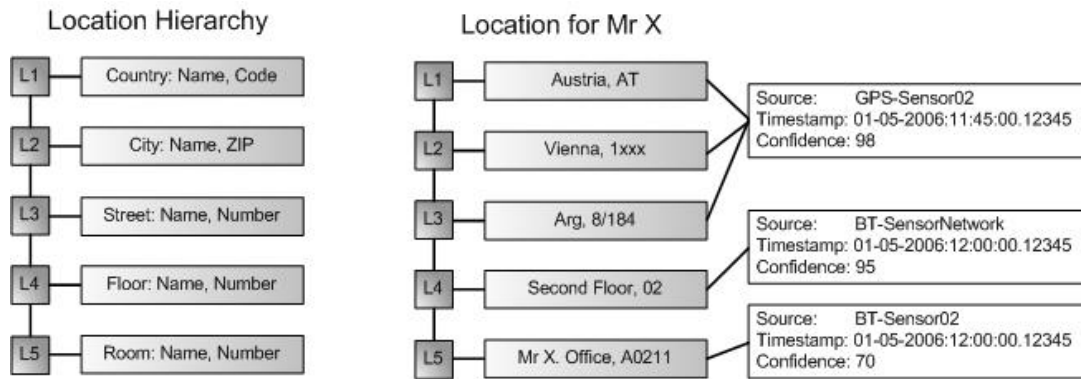


Figure 4. Context Location Hierarchy and Instance.

Each context value is tagged with source, timestamp, and confidence, where all confidence values within a hierarchy are monotonously growing [11]. As a consequence, context consumers need no longer worry about implicit sensor uncertainty as this information is provided at all levels. In doing so, our context model fulfills both CWE team and generic context modeling requirements:

- Specify preferred levels of detail: Peers can query and subscribe to context changes at their preferred level of detail.
- Fallback mechanism: If quality of context at one level is too low, more confident context information at a higher, coarse-grained level is available. An information theoretical approach to this problem by Castro and Muntz [12] focuses on the entropy of context information.
- Distributed composition: Hierarchies are described by XML schemas and can be compared to detect similarities and differences between hierarchies. Thus, there is no need for a centralized storage.
- Partial validation: For the purposes of comparing context values to existing hierarchies, no complete knowledge of the whole hierarchy is required.
- Richness and quality of information: The values within each hierarchy are derived from a number of sensors, each suitable for the respective level of detail.
- Incompleteness and ambiguity: Hierarchical structuring enables reasoning about low-level facts by means of high-level information. The extent of ambiguous values can be restricted to certain sub-trees of the respective hierarchy.
- Level of formality: By matching different versions of hierarchies, a common understanding can be created. Nevertheless, if no shared understanding at a detailed level can be created, it is still possible to match higher-level context values. As these are linked within a hierarchical ordering, matching context information is far less complex than having to compare all possible values for syntactic consistency and semantic correlation. Moreover, additional levels can be included to extend and unify hierarchies.

- **Applicability to existing environments:** As demonstrated below, sharing and querying of context information is easily achieved by means of Web services. Moreover, by having confidence values for every level of detail within a hierarchy, we decouple context consumers from context sensors even further.

Having discussed the benefits and adequacy of a hierarchical structuring for a context model, we present our architecture for processing context information in mobile environment in the next section.

A Distributed Architecture for Processing Context Information in Mobile

Environments

Our architecture design was guided by the requirement to support collaborative teams that have different needs. A goal was to abstract sensor characteristics from context consumers. Applications using context information need no longer know about detailed sensor characteristics and confidence in obtained context information. They can simply rely on the value for each level. Factors that have impact on confidence values are manifold.

- **Sensor Model.** Each sensor class has confidence values associated to it (e.g., statistically calculated) that describe the sensor's performance. Performance characteristics include a number of factors that limit the sensor's applicability for a particular application domain.
- **Heterogeneous Context Sensors.** In case of multiple sensors being used, sensors at different levels mutually improve the precision (e.g., GPS in combination with Bluetooth for localization).
- **Context Entities.** Accuracy of context data depends on a number of physical attributes and characteristics of the target entity we wish to observe (e.g., walking speed of a person, color, etc.).

•**Computing Context.** Accuracy and confidence of hierarchical context data depends on the inter-level mapping function (i.e., confidence increases if we step one level up).

Context at different levels of granularity, achieved through modeling in form of hierarchies, is not only used to establish team awareness at the organizational level but also reduces the required bandwidth for exchanging context at the link level. The amount of context data, that is being shared in tightly coupled or distributed teams, is likely to be larger than in collocated or loosely coupled teams. In addition to saving bandwidth, granular context helps to reduce computing time of context information (especially suitable for resource constraint devices) as only relevant levels of detail are transferred, thereby limiting processing and the amount of information that needs to be saved [13].

Container-based Context Processing Platform

A number of context-aware software services (i.e., bundles) can be deployed in a light-weight OSGi container. OSGi (the Open Services Gateway initiative) is a service-oriented, component-based environment that allows developers to manage the software lifecycle (see link 2 for more details) and is sometimes regarded as the “Universal Middleware”. OSGi is based on the Java platform and can be deployed on a wide range of computers and devices.

Services can be discovered and consumed by other bundles residing either in the same container or remote containers. A loose coupling of software services using OSGi containers and Web services for communication means among containers allows us to implement a scalable and flexible infrastructure. Figure 5 shows a block diagram of the set of services and components, which we implemented in Java and deployed in an OSGi container (we used the Knopflerfish open source OSGi implementation – link 3).

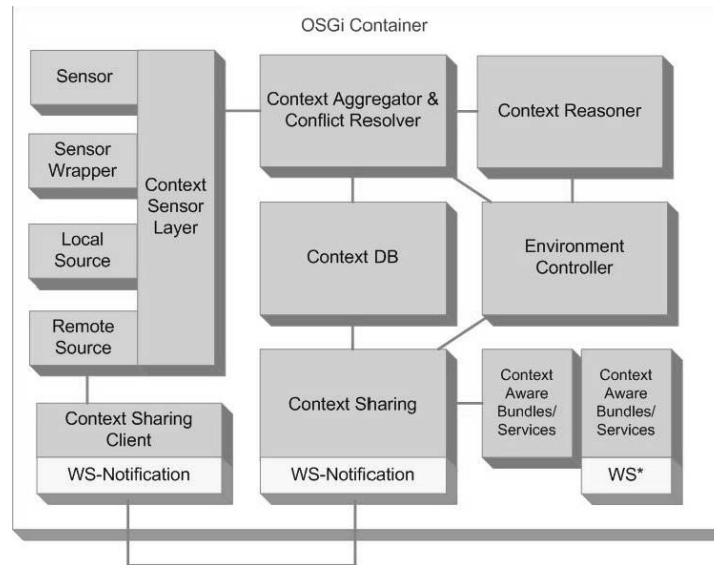


Figure 5. OSGi based platform for context-aware service provisioning.

- Context Information Acquisition.** A Context Sensor Layer abstracts various context sources which can be either software sensors or actual hardware elements. Bluetooth beacons and devices are an example for physical sensors that can be used to acquire indoor location information in an unobtrusive way. Bluetooth devices such as USB dongles are wrapped as logical entities, regardless of physical location, and controlled by the Sensor Layer. In our system Bluetooth measurements are fed into a Context Aggregation and Context Reasoning Component, which is implemented for each context source. A Bluetooth context source, for example, requires a Bluetooth Context Reasoner as location measurements may be imperfect or ambiguous. Note that location information can be acquired from a number of heterogeneous sources to increase confidence of obtained location estimates. In our implementation we make observations by scanning the environment with Bluetooth sensors periodically. Mobile entities such as Pocket-PCs and Smartphones, also equipped with Bluetooth, respond to these frequent scans. Thus, we are able to localize entities by resolving the mobile devices location through anchor points. These anchor points are associated to logical locations which are provided by an Environment Controller (e.g., associates a node to location XYZ

in a building). The Environment Controller essentially holds all persistent information needed for our context-aware infrastructure, such as anchor nodes, users, and devices belonging to a user.

- **Context Sharing.** Since our platform targets mobile devices, we need to have a flexible and efficient way to share context information. Web services subscriptions and notifications are used to access context information. The WS-Notification interface, data in hierarchies and the hierarchy structure itself need to have a common data format and schema (XML schema). We use JAXB2 (Java Architecture for XML Binding) to transfer Java objects into XML and vice versa. When a context change event is being fired, the Context Sharing bundle checks for context subscriptions that match the level or levels below. The subpart of the hierarchy that has changed is then encoded in XML and context change events are being sent to WS-Notification clients at remote locations.

Summary

Mobile context-aware applications are no longer only concerned with single or independent users. New Emerging, dynamic team forms such as Nimble, Virtual, and Nomadic teams and their specific interaction patterns demand a new form of representing and modeling of context information. Interaction context, respectively team context, is based on exchanging information that serves as the relevant situational or context-dependent data. The presented hierarchical context model and the context processing architecture are tailored to the needs of peers that are equipped with resource constraint devices in dynamic environments.

Acknowledgment

Part of this work was supported by the EU STREP Project inContext (FP6-034718) and Austrian Science Fund (FWF) grant P18368-N04 Project OMNIS.

Links

1. Project Management. http://en.wikipedia.org/wiki/Project_management.
2. OSGi - The Dynamic Module System for Java. <http://www.osgi.org/>.
3. Knopflerfish - Open Source OSGi. <http://www.knopflerfish.org/>.

References

1. A. Dey, G. Abowd, G. Towards a better understanding of context and context-awareness. In: Workshop on the What, Who, Where, When, and How of Context-Awareness at CHI 2000.
2. M. Baldauf, S. Dustdar, and F. Rosenberg. A Survey on Context Aware Systems. International Journal of Ad Hoc and Ubiquitous Computing, 2006. To appear.
3. P.D. Costa, L.F. Pires, M. van Sinderen, J.P. Filho. Towards a service platform for mobile context-aware applications. In: 1st International Workshop on Ubiquitous Computing - IWUC 2004. (2004) 48-61.
4. C.F. Sørensen, M. Wu, T. Sivaharan, G.S. Blair, P. Okanda, A. Friday. H. Duran-Limon. Context-aware middleware for applications in mobile ad hoc environments. In: ACM/IFIP/USENIX International Middleware conference 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing (online proceedings), Toronto, Canada (2004).
5. G. Biegel, V. Cahill. A framework for developing mobile, context-aware applications. In: Second IEEE Annual Conference on Pervasive Computing and Communications, 2004. PerCom 2004. (2004) 361-365.
6. T. van Do, I. Jørstad, S. Dustdar. Mobile Multimedia Collaborative Services. In Handbook of Research on Mobile Multimedia, Edited by Ismail Khalil Ibrahim, Idea Group Publishing, USA, 2006.
7. S. Dustdar, T. Hoffmann. Interaction pattern detection in process oriented Information systems, Data and Knowledge Engineering, Elsevier, 2006. To appear.
8. T. Strang, C. Linnhoff-Popien. A context modeling survey. In First International Workshop on Advanced Context Modelling, Reasoning, and Management, UbiComp, 2004.
9. C. Dorn, S. Dustdar, S. Sharing Hierarchical Context for Mobile Web services, Distributed and Parallel Databases, Springer, Special Issue on Context-Aware Web Services, To appear.
10. T. Strang. Service Interoperability in Ubiquitous Computing Environments. PhD thesis, L-M University Munich (2003).
11. C. Dorn, D. Schall, and S. Dustdar. Granular Context in Collaborative Mobile Environments. Proceedings: International Workshop on Context-Aware Mobile Systems CAMS'06, Oct 31 - Nov 1, 2006, Montpellier, France, Springer LNCS 4278, November 2006.
12. P. Castro, R. Muntz. Managing Context Data for Smart Spaces, IEEE Personal Communications, Vol.7, No.5, October 2000.
13. R. Gombotz, D. Schall, C. Dorn, and S. Dustdar. Relevance-Based Context Sharing through Interaction Patterns, The 2nd International Conference on Collaborative Computing, IEEE, Nov. 2006.
14. D. Schall, M. Aiello, and S. Dustdar. Web Services on Embedded Devices. International Journal of Web Information Systems (IJWIS), Troubador Publisher. February 2006.