Journal of Mobile Multimedia, Vol. 4, No.3&4 (2008) 200-209
© Rinton Press

TECHNIQUES FOR THE EFFICIENT RESOURCE MANAGEMENT OF CONTEXT-SENSTIVIE MOBILE APPLICATIONS AND THEIR UTILIZATION IN INDUSTRIAL FIELD SERVICE

MARKUS ALEKSY, RALF GITZEL, GERHARD VOLLMAR,
NICOLAIE FANTANA, CHRISTIAN STICH
ABB Corporate Research, Industrial Software Systems, Lab Germany
Wallstadter Str. 59, 68526 Ladenburg, Germany
markus.aleksy@de.abb.com

MAKOTO TAKIZAWA
Department of Computers and Information Science, Faculty of Science and Technology, Seikei University
3-3-1 Kichijoji-kitamachi, Musashino-shi, Tokyo 180-8633, Japan

Received April 20, 2008
Revised July 7, 2008

Context-sensitive mobile applications require a certain amount of flexibility due to the fact that they have to provide services for many different situations. The limited resources available make resource management a major challenge in such applications. In this paper, we present different techniques for the efficient use of resources of mobile devices. These techniques support the development of adaptable and flexible context-sensitive applications. Afterwards, we present some application scenarios from industrial field service in which some of these techniques can be utilized to improve service processes by providing tailored information and knowledge support.

Key words: Context-sensitive applications, mobile computing, adaptable applications, software engineering

1 Overview

A central requirement for mobile context-sensitive applications is dynamic adaptation to different situations and locations. However, due to the limited resources of mobile devices, such applications can only be realized with an efficient resource management approach. A framework for such applications must have flexible functionality regarding context-recognition and –interpretation. It must also be able to discover, administer, and integrate location-based services effectively.

The restricted resources of the mobile devices presently offered on the market represent a huge problem, which must be handled when realizing the necessary flexibility of a mobile application. When compared to conventional computer systems, many of today's mobile devices, e.g., PDAs or mobile telephones show restrictions such as low computer power and storage capacity, limited input/output functionality, low bandwidth, high latent periods when using wireless communication, or restricted power supply. Despite the ongoing further development and improvement of mobile systems, there will remain a discrepancy between them and their static counterparts [26].

In this paper, we concentrate on the description of techniques that provide more efficient ways to manage the memory of a mobile device.

2 Techniques for Resource Management for Context-Sensitive Mobile Applications

Resource management techniques can be divided into four categories:

- Resource allocation only,
- Resource management, i.e. integrated allocation, management, and deallocation of resources,
- Resource deallocation only, and
- Server-side support (for resource management).

These techniques will be discussed in the following sections.

2.1. Techniques for Resource Allocation

Resource allocation techniques deal with the granting of resources to services, user etc. In the context of memory as a resource, it refers to the allocation of main memory to processes.

As it has been mentioned before, context information indicates which services are suitable at a specific point in time. Thus, a framework for the creation of context-sensitive mobile applications should be able to dynamically load classes or components at runtime. Such components can be anything from context sensors or service-specific GUIs to additional business logic.

Context sensors for instance should be loaded dynamically because many are not needed under all circumstances and take up memory. For example, localization of the user is normally done by the Global Positioning System (GPS) [14]. However, when there is no line-of-sight to the GPS satellites (e.g. indoors) other techniques have to be utilized. For example, a mobile device entering a building can dynamically load a WLAN-based positioning sensor to replace GPS [15].

Various solutions have been proposed that are suitable for this problem. Fowler's Lazy Load Design Pattern [5] is the first example for memory allocation management. The pattern is based on the idea that an object does not load all its data immediately. Instead waiting for the first time it is needed. However, the location of the additional data is known to the object and thus can be loaded when required.

Another approach dealing with memory allocation is the Virtual Proxy design pattern [6]. Here, access to objects providing a specific service is handled by proxy objects. The proxy has the same interface as the real object and passes on any requests via a network, keeping most of the functionality and data out of local memory. On the other hand, this pattern can be used to implement certain pre-fetching or caching techniques. Pre-fetching uses prognoses to anticipate which data or modules will be needed locally. It thus allows a reduction of network latency. Examples for location-aware pre-fetching mechanisms can be found in [3] and [20]. Caching [17] is a related technique where data that has been loaded once is kept in local memory to avoid reloading it later.

The use of caching might appear counter-productive at the first glance, as is uses up the already-sparse resource memory. The virtual proxy is interesting as is allows the management of both memory and bandwidth at the cost of each other. By employing bandwidth, data can be stored on the server, reducing local memory use. On the other hand, caching and pre-fetching reduce the latency of the network resource as well as energy consumption at the cost of memory.

An additional pattern suitable for this context is the Lazy Acquisition pattern [16] which aims at allocating a resource as late as possible. It is based on four roles (cf. Figure 1). The User wants to access a Resource which provides a specific functionality (e.g. a positioning service). A Virtual Proxy represents the Resource in the fashion described above. Finally, the Resource Environment manages the individual resources.

A similar goal is pursued by the Partial Acquisition pattern [11]. With the exception of the virtual proxy, it is based on the same roles as the Lazy Acquisition and supports a step-by-step allocation of resources. When a specific service is selected based on context information it can be loaded together with an appropriate GUI. In this case, the GUI-specific classes are loaded first to be displayed the business logic is retrieved.

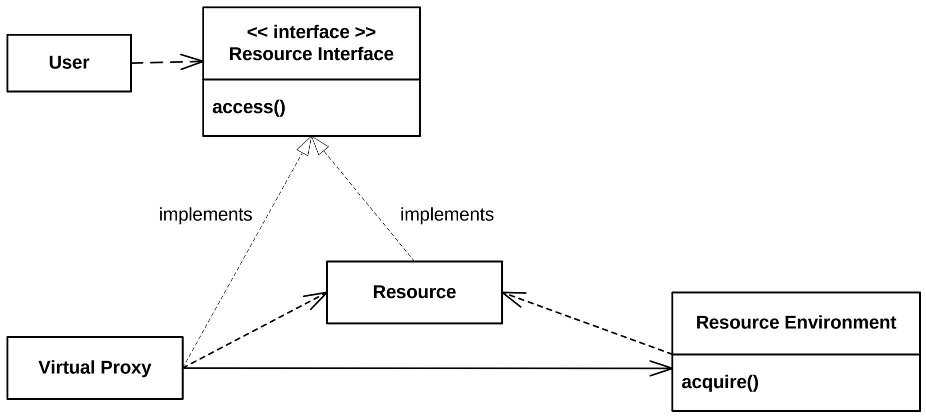

Figure 1: Lazy acquisition design pattern.

The techniques and patterns represented so far are limited to the aspect of resource allocation. An exception is the Virtual Proxy pattern, which not only provides functionality for the allocation of resources but also for deallocation. Unlike all the previous approaches, the following ones address both issues in equal measures.

2.2. Management Techniques for Resource Allocation and Deallocation

Intelligent allocation of resources is not the only way to reduce load. Another way is to end allocation as soon as it is no longer necessary. Thus, it is useful to look at techniques for resource deallocation as well.

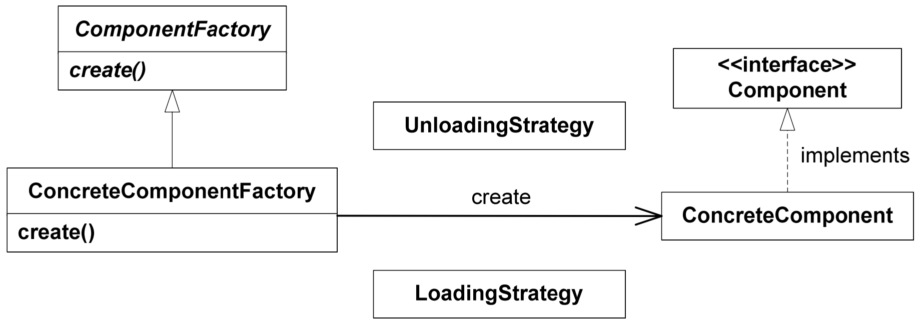

Figure 2: Virtual component design pattern [4].

The first pattern in this category is the Virtual Component pattern [4]. It is based on six elements (cf. Figure 2). The Component is the interface used to access a specific service. The functionality is realized by the Concrete Component, which implements the interface. The interface is provided by the Component Factory; Concrete Components in turn are provided by the Concrete Component Factory. The key to managing resources such as memory are the Loading Strategy and the Unloading Strategy, which determine when components are loaded or destroyed. The strategies are implemented by the factories.

Also, it is possible to reduce memory allocation by sharing objects. The Object Pool Design Pattern [7] allows objects to be used by multiple clients, thus reducing the number of required instantiations. It is based on the roles Reusable, Client, and Reusable-Pool.

Instances of the class ReusablePool are responsible for managing a pool of Reuseable object. Client objects can request an object to handle the required functionality from the ReusablePool. If an object is currently not in use, it does not become necessary to instantiate a new one, passing a reference to the existing one instead. This way, the number of new memory allocations is reduced.

Another solution is to use the Manager design pattern [28]. Its main roles are Manager, Client, and Subject. Instances of type Subject represent functionality that is needed by the Clients. The Manager-Object fully encapsulates the Subject and controls its entire life cycle. The Manager is the only source for obtaining references to the Subject. Since the Manager controls all the Subjects it can intelligently deallocate memory. It provides search, iteration, and deletion services for managed objects (cf. Figure 3).

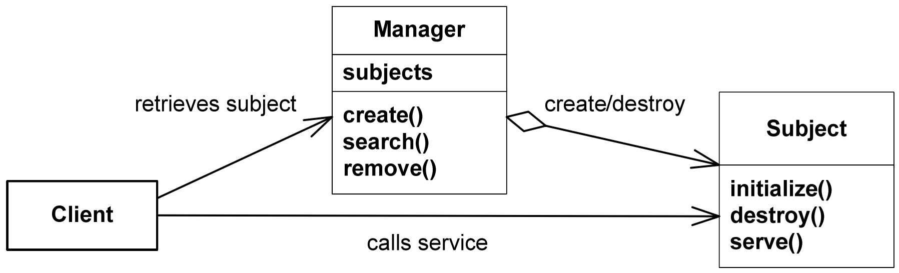

Figure 3: Manager design pattern [28].

A similar approach is taken by the Object Lifecycle Manager [21]. However, in this case, the focus is on the management of static or singleton objects [6]. Also, where the Manager pattern requires that the managed objects have a common base type, the Object Lifecycle Manager pattern allows objects of unrelated types to be managed. This is because the Object Lifecycle Manager relies on object composition and type parameterization to achieve greater decoupling of the manager from the managed objects. Drawbacks of the Object Lifecycle Manager pattern are its lack of support for search, iteration, and deletion.

Looking at the two patterns above, it can be said that the Manager pattern focuses on the repository aspects of object management, while the Object Lifecycle Manager pattern emphasizes the lifecycle aspects instead.

A final technique for intelligent deallocation of memory is the Evictor pattern [8], [9]. Since it is often used in the context of sever-side support, it is described in the next section.

2.3. Server-side Support for Resource Management

Even though one might assume that the problem of resource management is confined to the client, this is not the case. This is due to the fact that the flexible discovery, utilization, and management of services are not limited to the client application. It also requires a flexible infrastructure on the server side.

In order for a mobile client to use a service, the client must be able to locate services in a dynamic environment. For this reason it is necessary to decouple the client from the services.

2.3.1 Discovery of Services

To solve the problem of discovering services, the Lookup pattern can be used [18]. It consists of the roles Client, Service, and Lookup Service. The Lookup Service allows other Services to register themselves. Clients can use the Lookup Service to discover these other Services. This approach is used in many popular technologies such as the Common Object Request Broker Architecture (CORBA).

Additional patterns for service discovery exist. They are described as part of a pattern language by [24].

2.3.2 Management of Service Life Cycles

In the case of mobile clients, the usage of the Evictor design pattern [8], [9] appears to us to be the best solution for managing services. The evictor tries to ensure an efficient use of a service by monitoring it. Each time a service is accessed, a timestamp is set. Services which have been the least recently used (LRU) or are used the least frequently (LFU) are ideal candidates for resource deallocation. Deallocation can take place periodically or on demand. The pattern has four roles: The User of the service, the Resource (i.e. the service), the Evictor, which terminates services, and the Eviction Strategy, which defines the criteria for termination.

The main advantage of this design pattern is that it is transparent to the client. Service termination does not have to be handled by the client. Even if technical problems occur on the client or between client and server, the proper disposal of the service is guaranteed. The only drawback is that this transparency leads to an uncertainty – the client never knows when a service is terminated. However, this issue can be addressed by providing appropriate configuration mechanisms.

2.3.3 Limited-Time Availability of Services

Another problem is the fact that not all services will be available all the time. This can be due to technical reasons or be the result of a specific location. What a great opportunity to bring in the Leasing Pattern [10]! This pattern describes a method to limit the time a specific client has access to a service. Once this time ends, several possibilities exist. The leasing concept of Jini [30] for example, for which Sun has created a separate specification [29], has been neglected so far in the context of mobile location-based applications. One of the few exceptions is the use of context-based leases in Smart Spaces [13].

That the leasing concept is not limited to Jini is illustrated by the work of Jain and Kircher [10], who have used the idea to create the Leasing pattern. The authors describe several details of its applicability, structure, and variants. In technologies that do not inherently support leasing, it can thus be implemented based on the pattern. A good example is the CORBA Leasing Service described by [2].

The leasing concept is embodied by its four roles: Resources are generic services, which can be requested by clients from the Lessor/Grantor. The latter handles all the management tasks, matching resources to clients and ending the use of a resource when appropriate. A resource can only be used if the client has received a Lease, thus stopping to be a Resource Claimant and becoming a Resource Holder. The lease is a contract between the Lessor and the Holder and provides functionality such as the ability to prematurely release a resource.

Furthermore, several additional patterns exist, which support the development of dynamically configurable components. Examples are the Service Configurator [12] or the Component Configurator [27], as well as the pattern presented by [19] and [31]. Table 1 summarizes the techniques and design patterns discussed in this paper.

3 Related Work

Further design patterns for the implementation of memory-conscious software can be found in [22]. Besides the presentation of approaches to memory allocation the authors also discuss general concepts like small architectures and small data structures as well as techniques for the management of secondary storage and compression.

In [1] a pattern language for the development of adaptable location-specific mobile applications is presented. Additionally, patterns for mobile interaction have been discussed in [25].

ISSUES COVERED	**TECHNIQUES & DESIGN PATTERNS**
Resource Allocation	Lazy Load, Virtual Proxy, Lazy Acquisition, Partial Acquisition, Prefetching, Caching
Resource Allocation & Deallocation	Virtual Proxy, Virtual Component, Object Pool, Manager, Object Lifecycle Manager
Resource Deallocation	Evictor
Server-side Support	Evictor, Leasing, Service Configurator, Component Configurator

Table 1: Summary of the presented techniques and design patterns

4 Application Scenarios in Industrial Field Sevice

Changing business environments, technological progress and an aging workforce are some of the challenges that have to be faced in the industrial field service area. Especially in high wage countries, we observe some important developments that will influence the future's service organization and processes:

- There will be an increasing influx of non-experts into the workforce. As a consequence there will be an increased necessity to support service personnel with information and knowledge.
- Competition will increase cost-pressure on service units, with the consequence that mobile work must be performed more effectively and efficiently.
- Products and solutions will become more complex, with the consequence that field service engineers will require context-specific information and knowledge support.
- Partnerships for after-sales services will increase, i.e. personnel of external firms will cover some service tasks. Since they will not necessarily be familiar with the installed products, focused information to perform service specific tasks will have to be provided on an ad-hoc basis.

In the following sections, we will describe some application scenarios and discuss how far the aforementioned techniques can be utilized to improve field service processes.

4.1. Accessing Installed Based Information

Recent advances in information and communication technologies make it possible to provide tailored information about their current location to the mobile devices of field service engineers. Especially, access to installed base information is a key success factor to support the whole life cycle of systems and products as well as to provide lean and preventive service to ensure best conditions for the installed base. ABB's ServIS (Service Installed Base System) provides the facility to collect and consolidate information about installed base as well as a portal to access installed base knowledge. The system allows keeping track of all ABB products and systems information at a customer site, including technical and project details. It is integrated with other ABB information systems such as ABB Product, ABB People, and the global customer identification system.

In our scenario (cf. Figure 4), a combination of various technologies is used to support service engineers with tailored information and knowledge. The utilization of advanced identification and labeling technologies can act as a starting point in this approach. Advanced identification and labeling combines conventional product identification techniques, like nameplates and bar codes, with RFID-based tags. Radio Frequency Identification (RFID) is an identification, labeling, and data collection technology that uses radio-frequency waves to transfer data between a reader and an RFID tag placed on an item. Due to their small size, products can be retrofitted with RFID tags containing information such as serial number, ID, service history etc. A service engineer can use an RFID reader to retrieve data stored in RFID tags. Today, RFID readers that are equipped with Bluetooth allow connectivity with other mobile devices such as a service engineer's tablet PC. However, mobile device producers like Samsung has already developed a single-chip RFID reader for mobile devices [23]. The data stored on the RFID tag can be used to request further details from ServIS. The amount of knowledge received from ServIS might be large even if it is filtered using context-related attributes thus focusing on the current task of the service engineer. E.g., it can consist of technical and project data, customer and contact information, installed base value, service projects and events, spare parts, prior service reports, configuration data, etc. Here, design patterns, like Lazy Load and Partial Acquisition can be utilized to improve the loading of knowledge by waiting for the first time it is needed and incremental loading of knowledge, respectively.

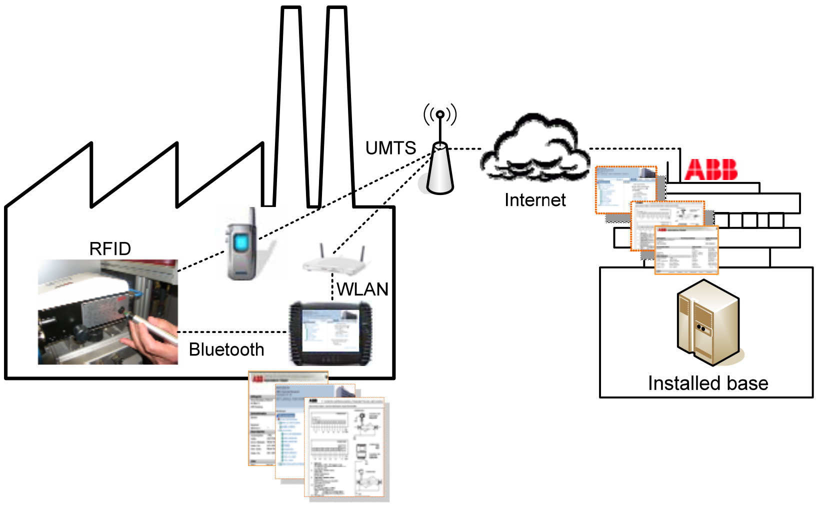

Figure 4: Accessing installed base information.

4.2. Utilizing Active Visual Labels for Product and Service Information

Bar codes act as a widely-used approach to represent information in a machine-readable manner. Besides one-dimensional (1D) bar codes also known as linear codes, such as the widely known EAN-13 code that is used worldwide for marking retail goods, two-dimensional (2D) bar codes (also known as matrix codes) recently gain more popularity. There is a myriad of different matrix code formats, e.g., the Datamatrix code used by Deutsche Post, Aztec code utilized by Deutsche Bahn as well as Schweizerische Bundesbahnen, and Quick Response (QR) codes which are applied in Japan. Matrix codes are able to store more data than linear bar codes. Furthermore, they can be utilized as foundation for three-dimensional bar codes by adding a third dimension, typically time or color. In the first case, a sequence of 2D bar codes is displayed successively.

In our scenario (cf. Figure 5), a product would use this technique to encode product and status information into 3D bar codes and to show it on its display. This information could consist of static data like serial numbers as well as dynamic data, e.g. error codes, messages etc. Service engineers can use a cell phone camera to capture these bar codes and to access product information by decoding them. However, due to the fact that there is a myriad of different matrix code formats, it is hard to store all of them on a mobile device. A solution to this problem is to equip it with one default decoding library and to load, install, and execute the remaining only in case they are needed by applying the Lazy Acquisition design pattern.

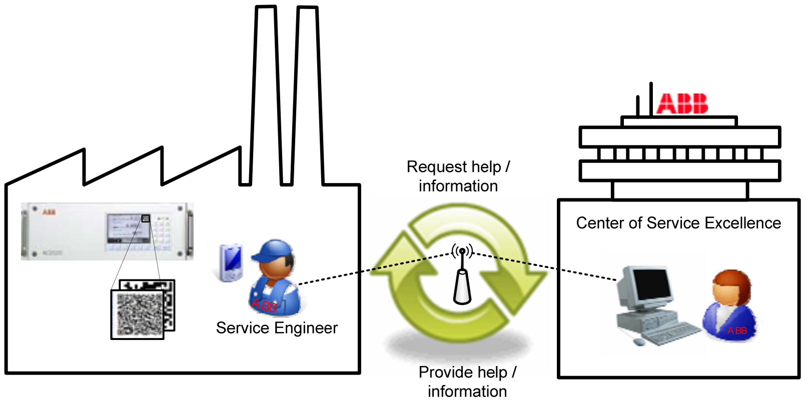

Figure 5: Applying 3D bar codes for product and service information storage.

5 Conclusions

As our survey shows, a myriad of techniques and design patterns exist that can be used for efficient resource management for context-sensitive mobile applications. Some of these techniques can be realized on all platforms. Others, like the dynamic loading of classes are limited to those platforms that support this capability. Dynamic loading is a powerful technique to save memory and allows a flexible design of applications. Besides the ability to dynamically load additional context sensors, it provides the opportunity to load different application GUIs or even complete applications for a specific context. Additionally, the use of suitable techniques on the server side allows a simpler realization of context-sensitive mobile applications by taking a lot of the resource intensive tasks away from the client.

Finally, we have presented some exemplary field service scenarios thus documenting the applicability of these techniques in real world applications.

References

1. Aleksy, M., Butter, T., A Pattern Language for the Development of Adaptable Location-Specific Mobile Applications. In Proceedings of the IADIS International Conference e-Commerce 2005, 15-17 December 2005, Porto, Portugal, IADIS, 2005.
2. Aleksy, M., Gitzel, R., Design and Implementation of a Leasing Service for CORBA-based Applications. In Proceedings of the 1st International Symposium on Cyber Worlds: Theories and Practices (CW 2002), 6-8 November 2002, Tokyo, Japan, IEEE Computer Society, 2002.
3. Cho, G., Using Predictive Prefetching to Improve Location Awareness of Mobile Information Service. In Proceedings of the 2002 International Conference on Computational Science (ICCS 2002), Lecture Notes in Computer Science Vol. 2331/2002, 21-24 April 2002, Amsterdam, The Netherlands, Springer Verlag, 2002, pp. 1128-1136.
4. Corsaro, A., Schmidt, D.C., Klefstad, R., O’Ryan, C., Virtual Component: A Design Pattern for Memory-Constrained Embedded Applications. In Proceedings of 9th Conference on Pattern Language of Programs (PLoP 2002), 8-12 September 2002, Monticello, USA, 2002.
5. Fowler, M., Patterns of Enterprise Application Architecture, Addison-Wesley Professional, 2002.
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Longman, 1995.
7. Grand, M., Patterns in Java Patterns in Java, Volume 1, A Catalog of Reusable Design Patterns Illustrated with UML, John Wiley & Sons, Chichester, 1998.
8. Henning, M., Vinoski, S., Advanced CORBA Programming with C++, Addison-Wesley, 1999.
9. Jain, P., Evictor. In Proceedings of 8th Patterns Languages of Programs Conference (PLoP 2001), 11-15 September 2001, Allerton Park, Monticello, Illinois, USA, 2001.
10. Jain, P., Kircher, M., Leasing. In Proceedings of 7th Patterns Languages of Programs Conference (PLoP 2000), 13-16 August 2000, Allerton Park, Monticello, Illinois, USA, 2000.
11. Jain, P., Kircher, M., Partial Acquisition. In Proceedings of 9th Conference on Pattern Language of Programs (PLoP 2002), 8-12 September 2002, Allerton Park, Monticello, Illinois, USA, 2002.
12. Jain, P., Schmidt, D. C., Service Configurator: A Pattern for Dynamic Configuration of Services. In Proceedings of the 3rd USENIX Conference on Object-Oriented Technologies and Systems (COOTS 1997), 16-20 June 1997, Portland, Oregon, USA, USENIX, 1997.
13. Jurmu, M., Perttunen, M., Riekki, J., Lease-Based Resource Management in Smart Spaces. In Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07), 19-23 March 2007, White Plains, New York, USA, IEEE Computer Society, pp. 622-626, 2008.
14. Kaplan, E., Understanding GPS. Artech House Publishers, 1996.
15. King, T., Kopf, S., Haenselmann, T., Lubberger, C., and Effelsberg, W., COMPASS: A Probabilistic Indoor Positioning System Based on 802.11 and Digital Compasses. In Proceedings of the First ACM International Workshop on Wireless Network Testbeds, Experimental evaluation and CHaracterization (WiNTECH 2006), Los Angeles, CA, USA, September 2006, ACM, 2006.
16. Kircher, M., Lazy Acquisition. In Proceedings of 6th European Conference on Pattern Languages of Programs (EuroPLoP 2001), 4-8 July 2001, Irsee, Germany, 2001.
17. Kircher, M., Jain, P., Caching. In Proceedings of the 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003), 25-29 June 2003, Irsee, Germany, 2003.
18. Kircher, M., Jain, P., Lookup. In Proceedings of 5th European Conference on Pattern Languages of Programs (EuroPLoP 2000), 5 - 9 July 2000, Irsee, Germany, 2000.
19. Kircher, M., Jain, P., Resource Lifecycle Manager. In Proceedings of the 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003), 25 - 29 June 2003, Irsee, Germany, 2003.
20. Kirchner1, H., Krummenacher1, R., Edwards-May, D., and Risse1 T., A Location-aware Prefetching Mechanism. In Proceedings of 4th International Network Conference (INC2004), 5-9 July 2004, Plymouth, UK, 2004.
21. Levine, D. L., Gill, C. D., Schmidt, D. C., Object Lifecycle Manager: A Complementary Pattern for Controlling Object Creation and Destruction. In Proceedings of 6th Conference on Pattern Language of Programs (PLoP 1999), 15-18 August 1999, Allerton Park, Monticello, Illinois, USA, 1999.
22. Noble, J., Weir, C., Bibby, D., Small Memory Software: Patterns for Systems with Limited Memory, Addison-Wesley Professional, 2000.
23. Nystedt, D., Samsung Develops RFID Chip for Cell Phones, IDG News Services, PC World, http://www.pcworld.com/article/id,139972/article.html, 2007.
24. Pärssinen, J., Koponen, T., Eronen, P., Pattern Language for Service Discovery. In Proceedings of 9th European Conference on Pattern Languages of Programs (EuroPLoP 2004), 7 - 11 July 2004, Irsee, Germany, 2004.
25. Röth, J., Patterns of Mobile Interaction, Personal and Ubiquitous Computing, Vol. 6, Issue 4, Springer, 2002, 2002, pp. 282-289.
26. Satyanarayanan, M., Mobile computing. IEEE Computer, Vol. 26, No. 9, September 1993, 1993, pp. 81-82.
27. Schmidt, D., Buschmann, F., Stal, M., Rohnert, H., Sommerlad, P., Pattern-Oriented Software Architecture–Patterns for Concurrent and Networked Objects”, John Wiley & Sons, Chichester, 2001.

28. Sommerlad. P., Buschmann, F., The Manager Design Pattern. In Proceedings of the 3rd USENIX Conference on Object-Oriented Technologies and Systems (COOTS 1997), 16-20 June 1997, Portland, Oregon, USA, USENIX, 1997.
29. Sun Microsystems Inc., Distributed Leasing Specification, January 1999, http://wwws.sun.com/software/jini/specs/jini10specs/lease-spec.html, 1999.
30. Sun Microsystems Inc., JiniTM Architecture Specification–Version 2.0", June 2003, http://wwws.sun.com/software/jini/specs/jini2_0.pdf, 2003.
31. Welch, L. R., Marinucci, T., Masters, M. W., Werme, P. V., Dynamic Resource Management Architecture Patterns. In Proceedings 9th Conference on Pattern Language of Programs (PLoP 2002), 8-12 September 2002, Monticello, USA, 2002.