An Architecture for the Effective Support of Adaptive Context-Aware Applications

Christos Efstratiou, Keith Cheverst, Nigel Davies and Adrian Friday

Distributed Multimedia Research Group, Department of Computing, Lancaster University, Bailrigg, Lancaster, LA1 4YR U.K.
e-mail: most@comp.lancs.ac.uk

Abstract. Mobile applications are required to operate in environments characterised by change. More specifically, the availability of resources and services may change significantly during a typical period of system operation. As a consequence, adaptive mobile applications need to be capable of adapting to these changes to ensure they offer the best possible level of service to the user. Our experiences of developing and evaluating adaptive context-aware applications in mobile environments has led us to believe that existing architectures fail to provide the necessary support for such applications. In this paper, we discuss the shortcomings of existing approaches and present work on our own architecture that has been designed to meet the key requirements of context-aware adaptive applications.

1 Introduction

Mobile applications are required to operate in environments that change. Specifically, the availability of resources and services may change significantly and frequently during typical system operation [8,11]. As a consequence, mobile applications need to be capable of adapting to these changes to ensure they offer the best possible level of service to the user [11]. While early research focused on applications which adapted to changes in network characteristics, there is now increasing interest in applications that adapt to general environmental and contextual triggers such as changes in a system's physical location, e.g. the GUIDE system [2,3] which supplies users with information tailored to their current location.

Current adaptive mobile applications are built using one of two approaches: either the adaptation is performed by the system which underpins the application (in an attempt to make transparent the effects of mobility) or, the application itself monitors and adapts to change. In some cases, these approaches are combined as, for example, in the MOST system [8] where the middleware platform adapts the operation of the network protocol in the face of changes in QoS and, additionally, reports these changes to the application to enable application level adaptation. However, in the general case, it has been demonstrated that maintaining transparency in the face of mobility is not practical and that it is difficult for a system to adapt without support from the application.

Careful examination of current approaches to supporting adaptation reveals two important facts. Firstly, support for adaptation is often fragmented with a range of mechanisms being used to notify applications of changes in different environmental and contextual attributes [4]. Secondly, there is a lack of mechanisms that support coordination of adaptive behaviour across the whole system, according to user requirements. In this paper, we explore the requirements for, and our research into creating, a unified architecture. It can support multiple contextual attributes coupled with a user driven adaptation control mechanism. The benefits of such an approach are clearly illustrated using a set of real-world examples.

2 Drawbacks of Current Approaches

Mobile systems need to be capable of adapting to a wide range of attributes such as network bandwidth, location, power etc. In general, current mobile systems provide support for adaptive applications by notifying applications when certain 'interesting' changes in attributes occur, e.g. bandwidth falls below some specified minimum threshold. It is then the responsibility of the application to adapt in an appropriate way, e.g. by reducing its bandwidth requirements. However, this approach can be shown to lead to inefficient solutions because of the lack of support for enabling coordination between the adaptation policies of multiple applications that may co-exist on the same system. In the following scenarios, we illustrate the kind of problems that could occur as a result of relying upon a simplistic notification based approach and isolated, uncoordinated, application adaptation.

2.1 Scenarios

The Need for Coordinated Application Adaptation for Power Management. This scenario illustrates the need for coordination in order to achieve efficient power management on a mobile system. One existing approach for handling power management, i.e. the ACPI [1] model, is to enable the operating system to switch hardware resources into low power mode when not in use, e.g. spinning down the hard-disk. This approach requires that applications leave hardware resources in an idle state for sufficient periods of time to make the transition between idle and active states worthwhile. Although this approach is suitable when only one application is running on a mobile device, the approach can prove ineffective when multiple applications or system services are sharing hardware resources. In more detail, the lack of coordinated access to hardware resources can result in poor utilisation of the shared resource and therefore sub-optimum power management. For example, consider the case of multiple applications that implement an auto-save feature. In the absence of any coordination between applications each application may choose to checkpoint its state to the disk at an arbitrary time, without considering the state of the disk (i.e. spinning or sleeping). In contrast, if applications are able to coordinate their access to the hard-disk then access to the disk can be clustered, allowing longer periods of inactivity. This latter approach is clearly more power efficient than the situation in which usage of the hard-disk is completely arbitrary and uncoordinated.

The Problem of Conflicting Adaptation. In this scenario, we illustrate the potential problems that can occur in a system that utilises separate adaptation mechanisms for different attributes. We consider a hypothetical mobile system which utilises two independent adaptation mechanisms, one for managing power and the other for managing network bandwidth. The two mechanisms can conflict with one another as the following example illustrates. If the system needs to reduce power consumption, the power management mechanism will request those applications that are utilising network bandwidth to postpone their usage of the network device in order to place the network device in to sleep mode. As a consequence of applications postponing their use of the network, the available network bandwidth increases. However, the network adaptation mechanism will detect this unused bandwidth and notify applications to utilise the spare bandwidth. In this way, the request to utilise available bandwidth is in direct conflict with the request to postpone network usage.

This example highlights the problem of relying on independent and uncoordinated adaptation mechanisms. Clearly, coordination or harmonisation is required in order to detect and avoid potentially conflicting adaptation mechanisms. In the example presented, the instruction given to applications to utilise more bandwidth should have been withheld if conserving power was the system's primary goal.

Utilising Alternative Location Sensing Mechanisms. This scenario considers the case of supporting multiple services for providing similar contextual information. In this case, we consider a location aware system that is capable of sensing its current location through two different mechanisms: a local GPS device and using beaconing in a cell-based wireless network. Using the latter mechanism, the system can identify the current cell in which it operates and thus specify its location. Both mechanisms deal with the same problem but they both have different characteristics. The GPS mechanism is typically the more accurate (accuracy in the region of 5m) but does require extra power to operate. Alternatively, the network-based solution is generally less accurate (depending on the size of a cell, e.g. approximately 200m for WaveLAN). However, the fact that the network device is already in use by the system for communications means that the addition power consumption required for identifying the location of the base-station would be minimal.

It follows that the mobile system would select to use the GPS based solution if accurate location information was required and concern over additional power usage was not an important issue. Alternatively, if the lifetime of the system's batteries (and therefore operation) was more important than achieving greater location accuracy then the network location mechanism would be the more appropriate mechanism. The adaptive strategy that would be most appropriate depends on both the user's requirements and the context of other attributes, such as power. In order for the system to make such decisions there is a basic requirement for system-wide adaptation policies. Without such policies, coordinated adaptation on the use of alternative context retrieval mechanisms is difficult because each application relies on a single mechanism without being able to identify the implications of its operation on other system resources and, consequently, other applications.

3 Analysis

Multiple Attributes. The previous scenarios illustrated a number of potential problems with current approaches to developing adaptive context-aware mobile systems. In this section, we generalise on these findings to present a critique of existing mobile systems and their suitability for supporting adaptive applications.

Based on the ideas of ubiquitous computing [20] future mobile systems should be able to discover changes in both the user and system environment and adapt to these changes. Current context-aware applications handle context in an improvised fashion. Application developers usually bundle the application with specific mechanisms for accessing context. However, this approach does not allow coordinated adaptation on context changes leading to the problems presented. Dey [5] has addressed this problem and suggested a general platform to support context-aware application. Our belief is that though a general platform for supporting context-aware application is necessary, this platform should also be capable of addressing the problems of coordinated adaptation.

The situation is complicated still further by the fact that the adaptive behaviour triggered by one attribute can cause side-effects on other attributes. These side-effects could, in-turn, trigger adaptation requests to other applications that result in conflicting actions (as illustrated in the conflicting adaptation scenario in section 2.1.2). Moreover, current research [4,6,7,12,13] has identified the need to provide adaptation solutions based on the combination of different attributes.

Existing architectures do not provide the necessary support to enable programmers to construct applications, which can adapt to multiple attributes and identify and cope with conflicts in adaptation strategies.

Adaptation Mechanism. Current mobile systems supporting adaptive applications tend to rely heavily on integrating QoS feedback and adaptation with network bindings. Examining the architecture of such systems allows us to identify a framework for analysing the architectural model of existing adaptive systems. The framework comprises two layers, the upper application layer and the lower representing the adaptation support platform. Between these two layers we can identify four distinct flows of control and information (see figure 1).

Flow A. Represents the requirements set by the application concerning the resources or attributes supported by the underlying infrastructure. For example, in the case of network adaptation this flow could represent the application's network QoS requirements.

Flow B. Represents the ability of the application to control the functionality of the underlying infrastructure. In the case of accessing a GPS device this could represent, for example, the control of the device by the application.

Flow C. Represents an information flow from the platform to the application. This could be used, for example, as a notification mechanism to inform the application when certain requirements cannot be met. Such notification could then trigger the application to adapt.

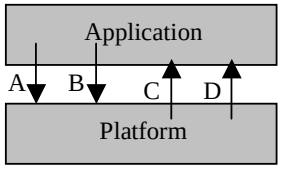

Fig. 1. Directed flows between applications and platform

Flow D. Represents the ability of the underlying platform to actually control the operation of the application. More specifically, this flow represents an explicit request from the system for the application to perform a specific adaptive behaviour. For example, the application might be requested to reduce its demand for network bandwidth or disk usage.

Consideration of this framework enables a classification of current systems according to the types of flows supported. For example, network based adaptive systems such as BAYOU [18], Odyssey [16], MOST [8] and Rover [10] support flows A and C.

Context aware applications like GUIDE [2,3], Stick-e Notes [17] and Cyberguide [14] are based on flows B and C. In more detail, for these applications, flow B represents the access to the various context-sensors while flow C represents the information flowing from the sensors to the application.

According to our knowledge no platform supporting context-aware adaptation provides a flow of control from the platform to the application. Indeed, although examples of systems providing this type of flow can be found in the distributed systems community, e.g. ISIS-META [15], it should be noted that these systems consider only network triggered adaptation.

4 Architectural Requirements

The previous sections have described the limitations of current approaches for supporting adaptive mobile applications. In particular, these approaches do not provide appropriate support to enable applications to adapt to multiple attributes in an efficient and coordinated way. This section considers a set of requirements that could be used to develop an appropriate architecture for supporting adaptive mobile applications.

4.1 Supporting a Common Space for an Extensible Set of Attributes

The first key requirement of the architecture is to provide a common space for handling the adaptation attributes used by the system. It is important that new attributes can be introduced into the system as and when they become important, e.g. the cost of specific services for mobile users or information about human physiology for wearable computers. The fact that new contextual attributes for triggering adaptation can arise implies that:

- the set of attributes that can trigger adaptation needs to be extensible,
- the characteristics of all these attributes may vary.

4.2 Application Control and Coordination

A second architectural requirement is the need to support the control of adaptative behaviour across all components involved in the interaction. As described earlier, one of the main limitations of current approaches is that applications themselves are responsible for triggering an adaptive mechanism when the underling infrastructure notifies them about any changes. In order to support flexible and coordinated adaptation there is a requirement for the triggering of adaptation on a system-wide level. Given this approach, the decision about when and how an application should adapt is pushed into an external entity, with cross-application knowledge, while the adaptive behaviour is still a part of the application's characteristics.

4.3 Support for System Wide Adaptation Policies

A further requirement is to support the notion of system-wide adaptation policies. More specifically, such policies should enable a mobile system to operate differently given the current context and the requirements of the user.

The specification of adaptation policies should be goal-oriented. Two kinds of goals can be identified:

1. effects on resources. The policy specifies a specific aim for the use a specific resource. Example policies include reducing the required network bandwidth and maximising the duration of operation of the system.
2. effects on applications. The policy specifies the mode of operation for specific applications. Example policies include defining priorities on applications which determine the order in which they are allocated resources and maximising the duration of operation of the system while having a specific application operating with full functionality.

5. Architecture

5.1 Structure of the Platform

We propose that future mobile adaptive applications should adopt an architecture in which mechanisms and policies are decoupled and, furthermore, mechanisms can be exposed and externalised in order to enable control by independent entities. Our architecture has been designed to address these requirements.

Figure 2 shows the relationship between the main components of the architecture of our proposed platform. The basic functionality of the architecture is two-fold, namely: the discovery and control of services offering contextual information and the coordination of adaptive behaviour of the system based on changes in context. More specifically, the platform discovers available context information in the system's environment and manipulates the contextual information in the *context database*.

Context aware applications expose their adaptive mechanism to the platform by registering with the *application database*. The *adaptation control* driven by *adaptation policies* (as specified by the user) coordinates the coexisting applications according to changes in context.

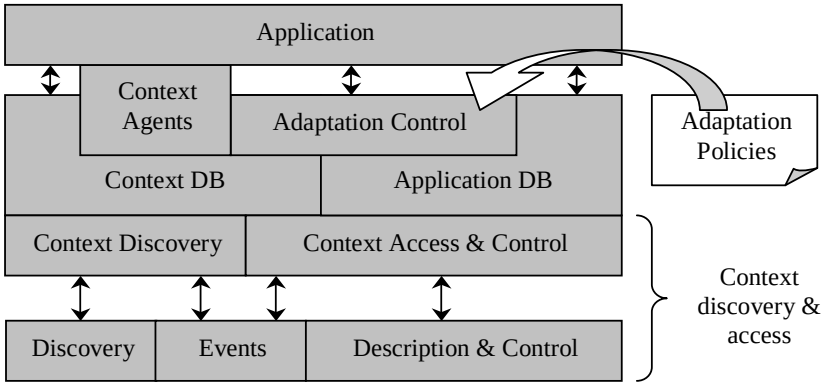

Fig. 2. The overall architecture of the system

All communications between the main components of the architecture are performed using HTTP as the transport protocol and XML to represent the format of the messages. The reason for this decision was based on the fact that these technologies support the lightweight integration of distributed components. Other alternatives, such as CORBA and RMI, are too heavyweight, and would require additional infrastructure, i.e. an ORB or RMI daemon respectively.

5.2 Context Discovery and Access

This part of our architecture is responsible for locating services that provide contextual information. These services can either be sensors embedded within the system (e.g. a body temperature sensor) or services in the surrounding environment (e.g. active devices in the user's environment). In a mobile environment the availability of context information can change rapidly. Thus, it is important to be able to discover context services when they are available and receive notification when they are not. For this reason, a considerable part of the architecture is based on the UPnP architecture [19]. In more detail, the mechanism for discovering available context services is based on services advertising themselves (using multicasting) to any interested context-aware applications.

Once a service is discovered the platform retrieves the XML description of the device, an abbreviated example of which is shown figure 3. The XML description provides the access points for sending control messages to the object and subscribing for event notifications. In addition, it defines the types of messages that can be sent to the service and the types of information that the service can offer. Given this information, the platform is capable of using the service and receiving notification events when the state of the service changes.

```
<service>
   <category> Location </category>
   <type> GPS </type>
   <action>
      <name> getXCoord </name>
   </action>
</service>
```

Fig. 3. Sample of the XML definition of a service

5.3 Context Database

This component serves as a registry for all those context services currently available. An important part of the context database is a classification of the context services into context types. More specifically, each context service has to specify the type of context that it supports, e.g both the GPS and the network based locator provide location information. The specification of context type is achieved using an XML template that defines the kind of information this type offers. Using this approach, an application can retrieve the specific contextual data in a way that is decoupled from the service used for acquiring the data.

This method of hiding the actual mechanism for retrieving contextual information, allows the platform to coordinate the access to context for different application. Moreover it offers our platform with the potential to switch between different services of the same type depending upon the predefined adaptation policies that have been specified.

5.4 Application Database

The application database serves as a repository for the adaptation mechanisms of all applications running on the system. The application developer is responsible for actually implementing the adaptive behaviour of his/her application. In addition, the application developer is also responsible for exposing this behaviour to the platform by ensuring that the application registers all of its adaptation mechanisms with the application database.

The description of the adaptation mechanisms should specify the type of context that can trigger this mechanism. This information is used by the adaptation control in order to coordinate the triggering of the applications based on changes in context. In figure 4, we illustrate an example of the information that may be provided by an application in XML, concerning its adaptive functionality.

5.5 Context Agents

A context agent is a piece of code that can be plugged into the platform in order to perform the application specific manipulation of contextual information.

```
<application>
   <name> WebBrowser </name>
   <adaptationMode>
      <name> lowBand </name>
      <trigger>
         <context> availableBand </context>
         <condition> lessThan-9600 </condition>
      </trigger>
   </adaptationMode>
</application>
```

Fig. 4. Sample of the XML definition of application's operation modes. The XML based description provides the different operational modes of the application, coupled with the contextual trigger that would make the application switch into that mode.

A common case of context manipulation is the combination of primitive context information for constructing a complex type of contextual data. For example, a context agent plugged into the platform can combine location data and current time in order to provide location-and-time tracking information similar to the data used by the Stick-e Note system [17].

To present even more clearly the operation of a context agent we will present an example based on the GUIDE system [2, 3]. The GUIDE system is a mobile electronic context-aware tourist guide. As part of its functionality it provides information about tourist attractions in HTML format, triggered by the location of the user. In order to introduce the GUIDE system into our platform we need to split the application into two parts: an ordinary web-browser and a location-triggered HTTP proxy. The HTTP proxy operates as a context agent which is plugged into the platform and has direct access to the location information provided by the platform. Triggered by changes in location, the agent can request the appropriate HTML data from the content server.

The key motivation behind introducing the notion of context agents is to enable the developer to distinguish the functionality of the application from the acquisition and manipulation of context data. Importantly, this allows context agents to be used for integrating non-context-aware applications (like an ordinary web browser) into a context-aware system.

5.6 Adaptation Control

This module is responsible for monitoring the status of contextual triggers and making decisions about the behaviour of the platform and the applications. The decision taking procedure is based on a set of adaptation policies specified by the user. These adaptation policies are specified by defining priorities both among the applications running on the system and among the resources of the system. This prioritisation represents the importance of these entities according to the user needs.

The adaptation control is further divided into two sub-modules: *internal adaptation* and *external adaptation as* explained below.

Internal adaptation. This module coordinates the context monitors that are required for all applications running on the system and coordinates potential adaptation within the platform itself. The adaptation actions that can be performed on context acquisition are tightly coupled with the context classification that has been described earlier (in section 5.3). Context services are clustered into context types according to the type of information they provide. For example, a GPS device and a network based location mechanism would be members of the same type of context, i.e. location. Both these mechanisms can provide similar types of information but have different specifications and different requirements. When the system gets into a state whereby one of the two mechanisms is favoured (according to the adaptation policies specified by the user) then the platform will switch to the mechanism that is preferred (as illustrated in figure 5).

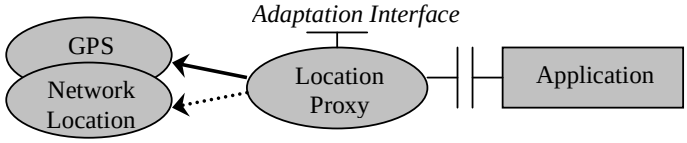

Fig. 5. Internal adaptation: switching between different location mechanisms.

In the example discussed, all applications that require location information do so via the location proxy object. Thus, the actual mechanism for retrieving contextual information is hidden from the application. This transparent access to context enables the platform to switch between mechanisms without affecting the applications involved. To illustrate this, consider the example described in section 2.1.4 and a scenario in which it is crucial to reduce the aggregate level of power consumption. In such a scenario, the platform could reduce power consumption by switching off the GPS device, and using the network locator in order to retrieve location information.

External adaptation. This module is responsible for coordinating the adaptive behaviour of the applications running on the system. Its operation is driven by both a set of adaptation policies defined by the user and the XML description of each application specifying its various operation modes (as described in section 5.4). Recall that the XML description of each operation mode is marked according to its effect on resource utilisation (e.g. power, network) and the use of context services. The adaptation control module can use this information in order to decide which operation mode has to be triggered under each potential set of circumstances. In more detail, when resources (power, network, etc.) become unavailable the adaptation control picks the adaptation mechanism with the lowest resource requirements.

In order to clarify this approach, consider the following example (which is illustrated in figure 6). Two adaptive applications run on a mobile device: a web browser and a video player. The adaptation policies specified by the user define priorities between the applications and the resources (such that the lower the number the greater the priority). The adaptation control module is aware of the adaptive modes that these applications can support by accessing the application database. It also knows the status of all the contextual data that is available on the system. When

any of the contextual triggers reach a value that triggers a reaction by the system, the adaptation control has to decide which adaptation mechanism should be invoked. In the example presented, both power and available bandwidth are low. However, the adaptation control would choose to overcome the power problem, because the user has specified that power is the most important resource. Note that enabling the user to specify priorities has in this case enabled the platform to overcome an area of potential conflict (as highlighted in section 2.1.2).

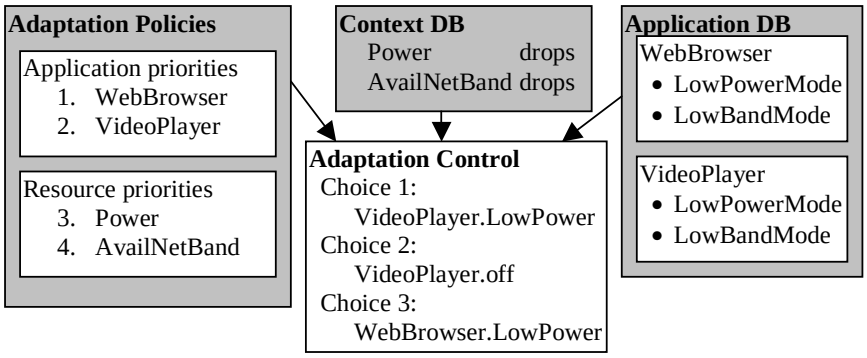

Fig. 6. External adaptation: making adaptation choices according to adaptation policies

In order to decide which application to invoke, the adaptation control component checks the prioritisation of the applications and picks the application that is less important to the user (i.e. the video player). The appropriate adaptation mode would then be triggered causing the video player to switch to low power mode. If the reduction in power resulting from this action is insufficient then the adaptation control component would proceed with the next course of action, i.e. turning off the video player and triggering the web browser to enter low power mode.

6. Conclusions

In this paper, we have argued that existing architectural approaches for supporting adaptive mobile applications have a number of shortcomings. Furthermore, analysis of these shortcomings has led to the identification of a set of architectural requirements. Namely: support for a common contextual space, mechanisms to support co-ordinated adaptations between multiple adaptive applications and support for user defined adaptation policies. We have also described the architecture of our platform, which has been designed to meet these requirements and which enables mobile systems to extend their awareness of all relevant contexts that might affect overall system adaptation policies. Fundamental to our approach is the idea of having system-wide decision making policies that consider the most efficient adaptation outcome from a number of possible adaptations. This is achieved by requiring the applications to provide information about themselves, their adaptation mechanisms, and the contextual triggers that can affect their behaviour.

References

1. Advanced Configuration and Power Interface Specification, Revision 1.0, Intel/Microsoft/Toshiba (1999).
2. Cheverst, K., N. Davies, K. Mitchell, A. Friday.: Experiences of Developing and Deploying a Context-Aware Tourist Guide: The GUIDE Project. In: Proc. of MOBICOM'2000, Boston, ACM Press (2000)
3. Davies N., K. Cheverst, K. Mitchell, A. Friday.: Caches in the Air: Disseminating Information in the Guide System. In: Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99) (1999)
4. Davies N., A. Friday, S. Wade, G. Blair.: L^{2}imbo: A Distributed Systems Platform for Mobile Computing. In: ACM Mobile Networks and Applications (MONET), Special Issue on Protocols and Software Paradigms of Mobile Networks, 3(2) (1998) 143-156
5. Dey A., Abowd G., Salber D.: A Context-Based Infrastructure for Smart Environments. In: Proc. of the 2000 Conference on Human Factors in Computing Systems (2000)
6. Elis C.: The Case for Higher-Level Power Management. In: Proc. of HotOS (1999)
7. Flinn J., M. Satyanarayanan.: PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. In: Proc. of the Second IEEE Workshop on Mobile Computing Systems and Applications (1999)
8. Friday A., N. Davies, G. Blair, K. Cheverst.: Developing Adaptive Applications: The MOST Experience. In: Journal of Integrated Computer-Aided Engineering, 6(2) 143- 157
9. Goland, Y., Cai T., Leach P., Gu Y., Albright S.: Simple Service Discovery Protocol, Version 1.03. IETF Internet-Draft. http://www.ietf.org/internet-drafts/draft-cai-ssdp-v1-03.txt
10. Joseph A., J. Tauber, F. Kaashoek.: Mobile Computing with the Rover Toolkit. In: IEEE Transactions on Computers: Special issue on Mobile Computing, 43(3), (1997)
11. Katz R.: Adaptation and Mobility in Wirless Information Systems. In: IEEE Personal Communications, 1(1) (1994) 6-17
12. Kravets R., P. Krishnan.: Application-Driven Power Management for Mobile Communication. In: Fourth ACM International Conference on Mobile Computing and Networking (MOBICOM '98) (1998)
13. Kunz T., J. Black.: An Architecture for Adaptive Mobile Applications. In: Proc. of the 11th International Conference on Wireless Communications (Wireless '99) (1999)
14. Long, S., R. Kooper, G.D. Abowd, C.G. Atkeson.: Rapid Prototyping of Mobile Context-Aware Applications: The Cyberguide Case Study. In: Proc. of the 2nd ACM International Conference on Mobile Computing (MOBICOM) (1996)
15. Marzullo K., R. Cooper, M. Wood, K. Birman.: Tools for Distributed Application Management. In: IEEE Computer, 24(8) (1991) 42-51
16. Noble B., M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, K. Walker.: Agile Application-Aware Adaptation for Mobility. In: Proc of the 16th ACM Symposium on Operating System Principles (1997)
17. Pascoe J.: The Stick-e Note Architecture: Extending the Interface Beyond the User. In: Proc. of the International Conference on Intelligent User Interfaces (1997)
18. Terry D. B., M. Theimer, K. Petersen A. J. Demers.: Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In: Proc of the 15th ACM Symposium on Operating System Principles (1995)
19. Universal Plug and Play Device Architecture, Version 0.91, Microsoft Corporation, March 2000. http://www.upnp.org/download/UPnP_Device_Architecture.mht
20. Weiser M.: Some Computer Science Issues in Ubiquitous Computing. In: Communications of the ACM, 6(7) (1993) 75-84