A Software Architecture for Provision of Context-Aware Web-based m-Commerce Applications

Benou Poulcheria
Department of Computer Science and Technology
University of Peloponnese
Tripoli, Greece
pbenou@ethnodata.gr

Vassilakis Costas
Department of Computer Science and Technology
University of Peloponnese
Tripoli, Greece
costas@uop.gr

***Abstract*—Mobile commerce is gaining significant importance in the recent years as an alternative option of e-commerce for the moving user. The mobile applications through which m-commerce takes place operate in highly dynamic environments with diverse characteristics and under varying conditions. The characteristics and conditions of these environments –called *context*– should be exploited in order to provide adaptive services; services that offer a suitable user experience and deliver innovative and enhanced capabilities that will facilitate user interaction, attract new customers and maintain existing ones. The goal of adaptivity is realized through the adaptation of user interface, functionality and content of applications using the context information. Therefore, context-awareness constitutes an essential aspect – almost a requirement – of mobile services. In order to realize context-aware services, there is a necessity to capture the context information from its sources, process and distribute it to the software components that will use it. In this paper, we propose a software architecture for context information management suitable for m-commerce applications. We describe the functionality and characteristics of its components, as well as the interaction among these different components.**

Keywords-mobile commerce; context; context-awareness; context management; adaptivity

I. INTRODUCTION

With the wide use of mobile computing, e-commerce has broadened the spectrum of its application and users to a new form of commerce known as *mobile electronic commerce* or *m-commerce* [1]. M-commerce takes place in highly dynamic environments with greatly varying characteristics and conditions. These characteristics are related to (a) the properties of the individual devices (memory capacity, battery life, processing power, input/output and communication capabilities), (b) the properties of the networking infrastructure (latency, bandwidth, disconnections, cost), (c) the properties of the natural surroundings (noise level, brightness, temperature), and (d) the personal characteristics, preferences, computer literacy and skills, needs and desires of the target audience.

Moreover, user mobility leads to the need for extending the use of these applications both temporally and spatially, at the same time, allowing users to interact with mobile commerce applications while concurrently engaging in other activities (e.g. driving). Hence, the full attention of the user cannot be assumed and alternative communication modes may need to be explored (e.g. auditory instead of visual) [6] [12]. Lastly, the merchandise (tangible or intangible) traded within an m-commerce transaction is of focal interest, since the added value of an m-commerce transaction lies in the ability to promote and trade the merchandise within the "anytime/anywhere" framework.

All the aforementioned particularities are known under the general term *context* [4], [14], [18]. Benou and Vassilakis [2] give a more formal definition for the context information tailored to m-commerce applications, according to which:

"Context information of an m-commerce application is every piece of information which may be used to characterize a state of an entity, which may be considered to be relevant to the interaction of the user with the particular application. The entity state may be either static or dynamically changing, while the relevance of the entity to the user-application interaction can be derived from the potential to exploit the information describing the entity state to optimize this interaction so as to maximize the commercial value of the application."

Context information should be necessarily modeled [2] and consequently exploited in order to offer adaptive m-commerce applications in terms of *user interface*, *functionality* and *content.* In order to achieve the goal of adaptability [17] of m-commerce applications, we should be equipped with the proper software system that will collect, process and distribute the context information [15]. The design of the subsystem that will manage the context information can be standardized, since it constitutes a standard and repetitive process for each mobile commerce application. Therefore, in this paper, we propose a scheme for middleware-level support for building context-aware services and applications, describing an architecture for context management suitable for the special characteristics of mobile commerce applications. The encapsulation of the content management logic and procedures into a distinct subsystem, separate from the application logic, results in a number of advantages regarding its manageability, maintainability and speed of application development

The remainder of this paper is organized as follows: In section 2, we propose an architecture for context management suitable for m-commerce applications. In section 3 we present related work that has been performed in the field of pervasive and mobile computing. In section 4 we discuss the advantages of the proposed architecture and its suitability for m-commerce applications. Finally, section 5 concludes the paper.

II. THE ARCHITECTURE FOR CONTEXT MANAGEMENT OF M-COMMERCE APPLICATIONS

A. The context information manager

The process of designing the system that will manage context information is common to all context-aware mobile commerce applications (CAMCA). Despite the fact that the context that different CAMCAs manage can be quite diverse, a well-defined context management architecture with standardized interfaces between its components and towards its clients, may practically be used to support the context management requirements of any CAMCA. Such a standardized architecture will constitute a useful tool for speeding up the development of context-aware applications [10] and minimizing the probability for errors or omissions. Furthermore, it will increase the potential for reusability, since context components developed for some application will be able to be incorporated in other applications with few or no changes.

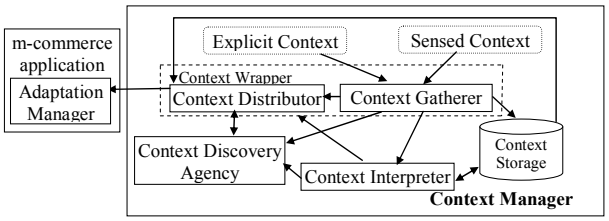

Figure 1. The Context Manager

In this paper, we will present below the design of the *Context Manager* module (Fig. 1), which is further decomposed into the following components: i) the *Context Gatherer, ii)* the *Context Interpreter, iii)* the *Context Storage, iv)* the *Context Distributor, and iv)* the *Context Discovery Agency*. The *Context Gatherer* is responsible for gathering the context information from the various sources of the application environment. The *Context Interpreter* is responsible for interpreting the context information to a higher level of abstraction. The *Context Storage* is responsible for storing the context information for subsequent use. The *Context Distributor* is responsible for distributing the context information to the applications that need it. The *Context Discovery Agency* is responsible for discovering the context information that can be made available to the interested parties. The interested parties are essentially the components responsible for performing adaptation within various information systems (frequently termed as *adaptation managers*). These components will use the information provided by the Context Manager to perform the adaptation of the application they provide.

B. Context Wrappers: Gathering and Distributing Context

The Context Gatherer is the subsystem which is responsible for collecting the context information from its sources. Context information may be gathered from *physical sensors* (e.g. location sensors such as GPS, identification sensors such as smartcard or fingerprint readers, motion sensors, etc) [8] or from *logical sensors* (e.g. APIs provided by the operating systems which allow the retrieval of information regarding the processing power, the available software and hardware components, the current time and so forth). An additional source of context information is the *user*, who is the source of explicitly provided context information (i.e. information directly entered by the user, such as gender, date of birth and so on). Some of this information may, of course, be stored into the main application database and from then on extracted from there. Depending on the source of the context information (physical sensors, logical sensors or users), the mechanisms that will capture it will be designed.

In order to decouple the applications from the details of the sensing process, we introduce a software module that undertakes the responsibility of reading context information from its source, encapsulating the peculiarities and idiosyncrasies of the particular context source and making the context information available for exploitation through a standardized interface, common for all kinds of context information. This module is named *context* wrapper. Fig. 2 illustrates the concept of the context wrapper through a UML diagram. Naturally, context wrappers will include source-dependent software, therefore a distinct context wrapper is required for each different context source. The presence of the context wrapper, however, enables us to handle introductions of new context sources or modifications of existing ones. This can be performed by correspondingly creating a new context wrapper or modifying the existing one, leaving the rest of the CAMCA and the context manager system intact.

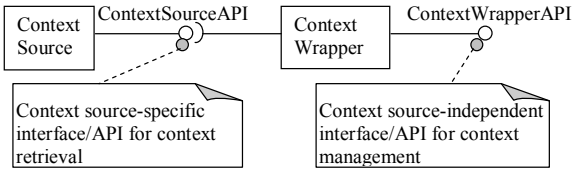

Figure 2. A Context Wrapper

Regarding their cooperation with other components, context wrappers provide the following functionalities:

1. they allow external entities, (e.g. adaptation managers of CAMCAs), to *retrieve* the values produced by the context source they manage.
2. they allow external entities to *subscribe* to notifications provided by the wrapper. These notifications allow interested applications to be informed about changes on the values of the context information sensed by some particular wrapper.
3. they *store* the values obtained in the context store for later usage.
4. they offer *reflection* capabilities, through which a context wrapper may be queried regarding the *context properties* it "measures" (e.g. user identity or user location),

5. they *register* themselves with the Context Information Discovery Agency. This registration allows the wrapper to be discovered by other software components. They also unregister themselves from the Context Information Discovery Agency when they cease their operation.
6. they enable their *detection* from the Discovery Agency, thus allowing the Context Information Discovery Agency to populate its context provider repository.

According to the above list of offered functionalities, the context wrapper interface depicted in Fig. 2 can be refined as shown in Fig. 3.

Essentially, context wrappers implement the *context gatherer* and the *context distributor* of the architecture depicted in Fig. 1, with the code liaising with the context source interface (cf. Fig. 2, Fig. 3) implementing the *context gatherer* and the code realizing the context source-independent interface/API being the *context distributor*. More specifically, the ContextQuery and ContextNotification interfaces of Fig. 3 implement the distribution of context information to interested parties, while interfaces ContextReflection, ContextDiscoverable and ContextDataStoreCom facilitate aspects of the context distributor's operation in the overall architecture.

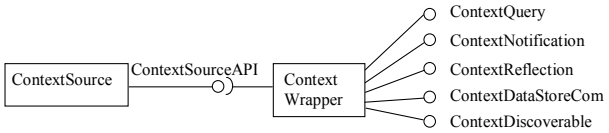

Figure 3. Refined Context Wrapper Interface

We must note here that the design presented above directly supports configurations where the context wrapper is not located on the same machine as the context source it manages. This is important for cases where some sensor is an embedded device with limited CPU power, communication capabilities or increased needs for energy preservation. In such cases, the sensor only needs to make available the data using a prominent mode (e.g. through an RS-232 connection or via Bluetooth), while the context wrapper will run on suitable hardware and undertake the tasks of context information gathering and distribution.

C. The context information distributor: Interface details and messages exchanged

The context distributor undertakes the task of making the context information available to the interested parties (notably the adaptation managers of CACMs) in a standardized and uniform manner. The context information distributor is implemented through the query and notification mechanisms built in the context information wrappers and realized by the ContextQuery and ContextNotification interfaces, respectively. These interfaces are complemented with interfaces ContextReflection, ContextDiscoverable and ContextDataStoreCom, where facilitate aspects of the context distributor's operation in the overall architecture. The query mechanism serves the need for *on-demand* provision of context information, with the initiative being on the side of the interested application. The notification mechanism (also referred to as publish/subscribe) [16] is suitable for repeating requests for context information where the interested application merely states the conditions under which it wishes to be notified of changes regarding the context information values.

1) The ContextQuery Interface

The interface to the query mechanism has the form:

queryContext(timeSpecificaion, attributeList)

attributeList designates which attributes provided by the sensor are requested. This is required since context wrappers may be attached to context sources (physical sensors, logical sensors or users), that provide numerous attributes but only few of which are needed. For instance, a meteorological data sensor may provide information about temperature, humidity, etc., and we need only to obtain information regarding temperature. Since *timeliness* is an important aspect of context information [2], the query mechanism allows the querying party to specify how "fresh" the context information is required to be through the *timeSpecification* designation.

The client defines to the wrapper the attributes it requires and the wrapper returns an appropriate reply, such as the one depicted in Fig. 4. This scheme decouples the querying mechanism from the context value obtainment implementation details, (e.g. interfacing to an RFID scanner, a floor sensor or a video image processor to detect the presence of an individual) and thus allows the application to be designed independently of the actual implementation of the sensing devices.

```
<ContextItem>
    <ContextAttributeName>Temperature</ContextAttributeName>
    <value>24.8</value>
    <metadata>
        <units>CelciusDegrees</units>
        <lastSensedTime>2010-04-08 12:32:11 EET</lastSensedTime>
    </metadata>
</ContextItem>
```

Figure 4. Reply to a queryContext request

2) The ContextNotification Interface

The notification mechanism of context information wrappers is activated when the software component, which is interested in receiving notifications regarding a particular piece of context information, places a *subscription* for a notification produced by a context wrapper (flow 1 in Fig. 5). Each such subscription is complemented with a *notification condition* which specifies the circumstances under which the particular subscriber wishes to receive notifications.

Every time the wrapper detects that a notification condition is satisfied, it will send a notification to the consumer that has placed the relevant subscription (flows *2* to *n-1* in Fig. 5). Finally, the context consumer may cancel

its subscription through an *unsubscribe* request (flow *n* in Fig. 5).

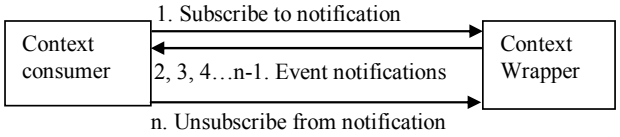

Figure 5. Publish/subscribe paradigm

A notification service is therefore fully defined through the following elements: (i) the notification service name, (ii) the attributes it monitors and their types and (iii) the elements that can be used to form notification conditions, as well as the types of these attributes. According to this description, a notification service which relates to the user location may be as shown in Fig. 6.

```
<Notification>
    <name>LocationUpdateNotification</name>
    <attributes>
        <attribute name=”location” type=”String”/>
        < attribute name=”identity” type=”integer”/>
    </attributes >
    <conditionElements>
        <conditionElement name=”location” type=”String”/>
        <conditionElement name=”previousNotificationLocation”
type=”String”/>
        <conditionElement name=”identity” type=”integer”/>
        <conditionElement name=”currentTimestamp” type=”datetime”/>
        <conditionElement name=”previousNotificationTimestamp”
”type=”datetime”/>
    </conditionElements>
</Notification>
```

Figure 6. Example of location update notification

When an interested party wants to register as a subscriber to a context information wrapper, it must specify (i) its identity, (ii) its location, i.e. the address at which notifications should be sent, (iii) the notification to which it subscribes, (iv) the attributes and the respective metadata which it wants to receive with each notification, and (v) the condition under which a notification should be sent to it.

3) The ContextReflection Interface

The *ContextReflection interface* allows context wrappers to be queried for the attributes and notifications they provide through the *queryContextAttributes* and *queryNotifications,* methods of the *ContextReflection interface*, respectively.

4) The ContextDataStoreCom Interface

The *ContextDataStoreCom* interface includes all provisions for communicating with the data store for storing values obtained by the context source for further perusal or for querying already stored values. These operations are accomplished through the *storeContextItemValue* and *retrieveContextItemValue* methods of the ContextDataStoreCom interface.

5) The ContextDiscoverable Interface

The *ContextDiscoverable* interface allows the context wrapper to be dynamically discovered by the respective modules within the context management architecture, and thus be subsequently used by interested context consumers. The *ContextDiscoverable* interface encompasses the methods *registerToDiscoveryAgency* and *unregisterFromDiscoveryAgency* to register and unregister the context wrapper to and from the discovery agency, as well as the *respondToContextConsumer* method in order to allow the context wrapper to be discovered from the discovery agent.

D. The context interpreter

The context interpreter is the module that produces context information of higher level of abstraction, as opposed to context wrappers which only produce low-level context data. More specifically, it collects “primitive” information elements from the context distributor and the data store and applies to them inference procedures according to rules that have been defined. For instance, it may retrieve the GPS coordinates corresponding to the user’s location to map it to a position on a specific road (e.g. “Motorway 5, 3rd kilometer”) or determine if the user’s location is “home,” “office” or “on the move.” The inference procedure may be performed using simple if/then rules or through more elaborate algorithms and techniques

The full definition of a context interpreter includes (i) the information that will be interpreted (e.g. specific attributes), (ii) the context attributes that will be produced as output of the interpretation procedure, (iii) the procedure that will perform the interpretation, and (iv) the notifications provided, if any.

Context interpreters adhere to the context wrapper specifications and implement the ContextQuery, ContextNotification, ContextReflection, ContextDataStoreCom and ContextDiscoverable interfaces, thus being ContextDistributor themselves and providing the services described in section C.

E. The context information discovery agency

The context information discovery agency implements facilities for storing information about the context providers (context information wrappers, context information interpreters), for locating them and for informing interested parties of how they can be contacted. Additionally, it offers information about itself in order to be detectable from context providers.

The *AddDiscoveredContextObject* and *RemoveDiscoveredContextObject* methods allow to add and remove entries of context providers to and from the discovery agency registry. The *QueryForDiscoveredContextObjects* offers information regarding registered context providers, while the *RespondToContextProvider method* allows context information providers to locate the context information discovery agency (and subsequently register to it).

F. The context information store

The context information store allows for long-term storage of context information. The context information may be produced by any context information provider and once stored in the context information store, may be later retrieved by context consumers. In this sense, the context information

store plays the role of a buffer between context producers and context consumers, decoupling the context production from the context consumption time, while it also offers the potential to store large amounts of context data, which would be infeasible to do in other components.

III. Related Work

Insofar, numerous researchers have proposed software systems that aim at managing context information. The most widespread architecture is the one involving one or more centralized components for context information management and some distributed components for context information collection. This approach has been proposed by Korpipää et al [13] and the related system comprises of three functional entities namely the *context manager*, the *resource servers* and the *context recognition services*. The *resource servers* and *context recognition services* are distributed components responsible for gathering context information, while *context manager* is a centralized server storing context information and delivering it to the client applications. The SOCAM architecture (Service-oriented Context-Aware Middleware) [9] also employs a centralized server termed *context interpreter,* which collects data from distributed context providers and offers it, in processed format, to client applications.

Another centralized middleware-based approach that has been designed for context aware mobile applications is the one proposed by Fahy and Clarke [7] in the CASS project (Context-Awareness Sub-Structure). The middleware contains an *Interpreter*, a *ContextRetriever*, a *Rule Engine* and a *SensorListener*. The *SensorListener* listens for updates from sensors which are located on distributed computers, called sensor nodes. Then the gathered information is stored in the database by the *SensorListener*. The *ContextRetriever* is responsible for retrieving stored context. Both of these classes may use the services of an interpreter. CoBrA (Context Broker Architecture) [3] is another centralized agent-based architecture that may support context-aware applications. The key component of the CoBrA architecture is the *intelligent context broker*, which maintains and manages a shared contextual model on behalf of a community of agents (applications hosted by mobile devices, services provided by a room, web services). The *context broker* consists of four main sub-components, namely the *Context Knowledge Base*, the *Context Inference Engine*, the *Context Acquisition Module* and the *Privacy Management Module.*

The Context Toolkit [5] is a context-aware framework that adopts a peer-to-peer architecture, introducing however a "super-peer" node which acts as a centralized discoverer. Distributed sensor units (called *widgets*), *interpreters* and *aggregators* register themselves to the centralized discoverer to ascertain that they are discoverable by the client applications.

The architecture proposed in the Hydrogen project [11] attempts to avoid the use of a centralized component, distinguishing initially the context information to local (context of the device itself) and remote (context from another device). The architecture has then three layers: The *Adaptor* layer is responsible for the gathering of context information by querying sensors; the *Management* layer is responsible for delivering context information; and the *Adaptation* layer which performs the adaptation of the application.

The context management systems overviewed in this section differ among themselves in the following respects: (a) the comprehensiveness of the context information elements they can manage efficiently, (b) the location of the different components that will perform the different context management operations within the network, (c) the spectrum of operations they offer for context management, and (d) the degree of suitability for web-based m-commerce applications. Moreover, taking into account that the notion of context is extensively used in the areas of pervasive and ubiquitous computing, most of these systems aim to include provisions for context management in smart spaces (e.g. smart vehicles, intelligent rooms, smart conferences places, etc). Within smart spaces, context information is transferred from its capture points (e.g. sensors) to the context information management server using WiFi, Bluetooth and Ethernet networks (which are high-bandwidth and with small or no usage costs), as opposed to GPRS/UMTS networks which are widely used in m-commerce settings.

IV. Evaluation

Our proposed architecture has similarities and differences with the aforementioned works. It covers the management of all kinds of context concerning the m-commerce applications (computational, environmental, user, application specific context) regardless of the way of acquisition (sensors, user, derived). It implements all the necessary functions that a context management system should offer (capture, store, interpret, discover, transit context), in contrast with some of the previous work. It also adopts the use of middleware for context management, using both centralized components (mainly for management, storage and dissemination of context information) and some distributed components for capturing the context information. This arrangement is suitable for mobile commerce applications for the following reasons:

i) A single software component (context manager) will manage issues stemming from concurrent access to sensors.
ii) Centralized management of context relieves the mobile devices from the burden of managing context themselves. This is particularly important since resources in mobile devices are scarce. Additionally, centralized management and storage allows us to store large amounts of context information and perform complex and advanced interpretation as needed.
iii) The user interface (data, presentation properties, functionality) of web-based m-commerce applications is performed on a centralized application

server. Taking into account that the context storage components is also centralized, the two components can communicate efficiently through high bandwidth networks, relieving mobile devices from the need to continuously transfer context information through slow and costly channels. The use of these channels is limited to the absolute minimum number of messages required to transfer context information from capture sources directly to the centralized server or other interpreters.

iv) The use of distributed context wrapper components allow for capturing of context from remote locations (mobile devices, weather and traffic sensors, etc).

v) The component-based architecture allows the implementation using web services technology, which promotes independence from programming language, underlying operating system or middleware, while it also guarantees interoperability, which is a requirement for web-based m-commerce applications.

The adopted approach for context management allows for hiding the low-level sensing details from all context consumers (interpreters, adaptation manager, applications). Additionally, the main code of the mobile commerce application doesn't need to receive notifications (these are forwarded to the context manager). This kind of implementation removes the need for using advanced programming techniques, such as a separate thread to receive notifications or signal handlers to be invoked upon arrival of an incoming notification, simplifying thus the mobile e-commerce application development and reducing the possibility of bugs.

V. CONCLUSION

The design of the subsystem that will manage the context information can be standardized, since it constitutes a standard and repetitive process for each mobile commerce application. Additionally, the encapsulation of the content management logic and procedures into a separate subsystem results in a number of advantages regarding its manageability, maintainability and speed of application development.

In this paper, we have presented a high-level software architecture for context information management, suitable for m-commerce applications. Additionally, we have described the functionality and characteristics of its components, as well as the interaction among these different components. The presented architecture is modular, hides the complexity associated with different sensing methods, diverse context sources and various access technologies. Additionally, it leads toward a user-transparent infrastructure that provides application developers with services that facilitate and quicken context aware mobile commerce applications development.

VI. REFERENCES

[1] Benou, P., Bitos, V. (2008) Developing Mobile Commerce Applications. Journal of Electronic Commerce in Organizations, Vol. 6, No.1, pp. 63-78.

[2] Benou P., Vassilakis C. (2010) The Conceptual Model of Context for Mobile Commerce Applications. Journal of Electronic Commerce Research, Vol. 10, Vo. 2, pp. 130-165, Springer-Verlag.

[3] Chen, H. (2004) An Intelligent Broker Architecture for Pervasive Context-Aware systems. PhD Thesis, University of Maryland, Baltimore County.

[4] Dey, A., Abowd, G. (1999) Towards a Better Understanding of Context and Context-Awareness. Technical Report 99-22, Georgia Institute of Technology.

[5] Dey, A., Abowd, G. (2001) A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human- Computer Interaction, 16 (2-4), pp. 97–166.

[6] Dunlop, M., Brewster, S. (2002) The Challenge of Mobile Devices for Human Computer Interaction. Personal and Ubiquitous Computing, Vol. 6, No. 4, pp. 235-236.

[7] Fahy, P., Clarke, S. (2004) CASS – a middleware for mobile context-aware applications. Proceedings of the Workshop on ContextAwareness, MobiSys.

[8] Gellersen, H., Schmidt, A., Beigl, M. (2002) Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts. ACM Journal of Mobile Networks and Applications, Vol. 7, No. 5, pp. 341 –351.

[9] Gu, T., Pung, H. K., Zhang, D. Q. (2005) A Service-Oriented Middleware for Building Context-Aware Services. Journal of Network and Computer Applications (JNCA), Elsevier, Vol. 28, Issue 1, pp. 1-18.

[10] Henricksen, K., Indulska, J., McFadden, T., Balasubramaniam, S. (2005) Middleware for Distributed Context-Aware Systems. On the Move to Meaningful Internet Systems, Springer, LNCS 3760, pp. 846-863.

[11] Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., Altmann, J. (2002) Context-awareness on mobile devices – the hydrogen approach. Proceedings of the 36th Annual Hawaii International Conference on System Sciences, pp.292–302.

[12] Kaikkonen, A., Kallio, T., Kekäläinen, A., Kankainen, A., Cankar, E. (2005) Usability Testing of Mobile Applications: A Comparison between Laboratory and Field Testing. Journal of Usability Studies, Issue 1, Vol. 1, pp. 4-16.

[13] Korpipää, P., Mäntyjärvi, J., Kela, J., Keränen, H., Malm, E. J. (2003) Managing Context Information in Mobile Devices. IEEE Pervasive Computing, Vol. 2, No. 3, pp. 42-51.

[14] Koukia, S., Rigou, M., Sirmakessis, S. (2006) The Role of Context in m-Commerce and the Personalization Dimension. Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology, pp. 267-276.

[15] Kranenburg, H., Bargh, M.S., Iacob, S., Peddemors, A. (2006) A context management framework for supporting context-aware distributed applications. Communications Magazine IEEE, Vol. 44, Issue 8, pp. 67-74.

[16] Mühl, G., Fiege, L., Pietzuch, P. (2006) Distributed Event-Based Systems. Springer, 1st edition.

[17] Noble, B., Satyanarayanan, M., Narayanan, D., Tilton, J., Flinn, J., Walker, K. (1997) Agile application-aware adaptation for mobility. In Proceedings of thw sixteen ACM Symposium on Operating Systems Principles, pp. 276-287.

[18] Rakotonirainy, A., Loke, S., Fitzpatrick, G. (2000) Context-Awareness for the Mobile Environment, Proceedings of the Conference on Human Factors in Computing Systems.