Cent. Eur. J. Comp. Sci. • 2(2) • 2012 • 87-117
DOI: 10.2478/s13537-012-0010-z

Central European Journal of **Computer Science**

A context management architecture for m-commerce applications

Research Article

Benou Poulcheria[1]*, Vassilakis Costas[1]†

1 University of Peloponnese,
Terma Karaiskaki 22100 Tripoli, Greece

Received 24 July 2011; accepted 11 April 2012

Abstract: Mobile commerce applications operate in highly dynamic environments with diverse characteristics and interesting challenges. The characteristics and conditions of these environments –called *context*–, can be exploited to provide adaptive mobile services, in terms of *user interface*, *functionality* and *content,* in order to offer more effective m-commerce. Today, building adaptive mobile services is a complex and time-consuming task due to the lack of standardized methods, tools and architectures for the identification, representation and management of the context. Addressing some of these issues, recent works have provided formal extensions for various stages of the m-commerce application lifecycle, such as extended UML class diagrams for building design models and have used context parameters in order to offer adaptive applications. Using these works as the basis, in this paper we propose a context management architecture, which accommodates the requirements that have been identified for m-commerce applications. The proposed architecture is evaluated in terms of completeness, complexity, performance and utility, and compared against other approaches proposed in the literature regarding its suitability for supporting context-aware m-commerce applications.

Keywords: mobile commerce • context • context-awareness • context management • adaptivity • software architecture

1. Introduction

With the appearance and penetration of mobile computing devices and the advent of wireless communication technologies, e-commerce has broadened the spectrum of its application and users to a new form of commerce known as *mobile electronic commerce.* According to [5], "*Mobile commerce* or *m-commerce is defined as any activity related to a commercial transaction (or a potential one) – an exchange of services or goods for money - and is conducted via wireless and mobile communication networks and uses wireless and mobile devices as user interface.*"
The process of designing and developing mobile commerce applications is inherently more complex and demanding, as compared to traditional applications, due to the fact that they are executed in diverse environments, as opposed to

* *E-mail: pbenou@ethnodata.gr*
† *E-mail: costas@uop.gr (Corresponding author)*

"traditional applications" which are typically executed on the relatively stable desktop PC. Within these environments, there are greatly varying characteristics regarding (a) the properties of the individual devices (memory capacity, battery lifetime, processing power, input/output and communication capabilities), (b) the properties of the networking infrastructure (latency, bandwidth, disconnections, cost), and (c) the properties of the natural surroundings (noise level, brightness, temperature). Moreover, user mobility leads to the need for extending the use of these applications both temporally and spatially, while, at the same time, users may 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) [18, 41].

M-commerce applications are, furthermore, addressed to an audience with greatly varying qualities regarding their personal characteristics, preferences, computer literacy and skills, needs and desires. 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. These particularities, known under the general term *context* [16, 45, 59], demand from an m-commerce application adaptability, in terms of *user interface*, *functionality* and *content* [42], so as to maximize user satisfaction and increase the amount of sales. The utilization of context for the provision of adapted applications may offer a number of advantages for providers of m-commerce as well as for users. We will indicatively mention some of them:

i) The providers will be able to offer the functions of the same basic application to a wide range of devices and networks with the result of approaching more users to whom to offer their services anywhere and anytime. Also, they will be able to utilize environmental parameters (e.g. location) for the provision of innovative services (e.g. theatres or restaurants closest to the user, location-based advertising) with the goal of attracting and maintaining clients.

ii) The user, who on some occasions may have to give some personal data (e.g. age or preferences) and the consent to use them, will have the advantage of being able to receive timely interesting information at any time and regardless of his/her location. Examples of that would be product offers at a shopping mall, parking areas or theatres closest to him/her, product or service lists according to his/her interest, stock prices exceeding a limit, etc.

In order to achieve the goal of adaptability of m-commerce applications, we should be equipped with (a) a solid perception of concepts and structures related to context, (b) a methodology to capture the important context factors and include them in the m-commerce application design and (c) the proper software system that will collect, process and distribute the context. So far, a number of systems utilizing context (context-aware systems) have been developed, especially in the field of pervasive computing, but either the array of the context they manage is very limited (e.g. user's location and identity), or they are not widely available to everyday users [30]. This can be attributed to the fact that while the process of developing context-aware mobile services constitutes a complex and time-consuming task, the above requirements for achieving adaptability of m-commerce applications, especially (b) and (c), are still lagging. In this paper, we propose a scheme for middleware-level support for building context-aware services, describing an architecture for context management suitable for the special characteristics of mobile commerce applications; our proposal thus focuses on factor (c) listed above. To better support the presentation of the proposed architecture, we also include in this paper appropriate elements for context definition (factor a) and representation (factor b).

The remainder of the paper is organized according to the framework proposed by March et al. [50] for presenting design science research outputs, which suggests that design science research output or artifacts should include: *constructs* (the vocabulary of the domain, constituting a conceptualization used to describe problems within the domain and to specify their solutions); *models* (representing situations as problem and solution statements); *methods* (facilitating the construction of a representation of user needs, the transformation of user needs into system requirements and then into system specifications and finally into an implementation); and *instantiations* (realization of artifacts in their environment). Following this framework, the paper is structured as follows:

1. in Section 2, we describe the "*problem*" or otherwise the necessity and requirements that derived from a decision to exploit the context in order to offer adaptive m-commerce applications,

2. in Section 3, we present a set of *constructs* which conceptualize the concepts of context and convey the relevant knowledge in a form understandable from IT practitioners,

3. in Section 4, we present the extended UML class diagrams as a *model* for expressing constructs and the relations among them,

4. in Section 5, we propose a context management, presenting the overall design and the individual modules) and discuss how the presented architecture tackle the requirements for context management presented in Sections 2 and 3; a number of implementation issues for the proposed method are also discussed in subsection 5.7,

5. in Section 6, we present our experiences from the usage of the proposed context management architecture, discussing the implementation of an adaptive application as a case-study,

6. in Section 7, we review other approaches tackling context management and compare them to the method proposed in this paper, focusing on the suitability of each approach for m-commerce application development and the comprehensiveness of features provided, while we also point out our contribution in the field,

7. finally, in Section 8 conclusions are drawn and future work is outlined.

2. Problem statement and requirements

Context has been independently studied in the domains of: i) mobile computing [44], ii) pervasive and ubiquitous computing [17], and iii) e-commerce. Each of these domains approaches and handles context from its own viewpoint.
In the domain of mobile computing, emphasis is placed on *computational context*, i.e. the context referring to device characteristics (e.g. memory capacity, processing power) and network characteristics (e.g. latency, bandwidth); these characteristics are mainly collected by logical sensors (e.g. operating system APIs). In the domain of pervasive and ubiquitous computing, context is approached through individual environmental parameters (*environmental context*), which are mostly captured by physical sensors (e.g. location, presence, motion, temperature sensors). Finally, in e-commerce, the aspect of *personalization* has been investigated in order to tailor the behaviour of a system's interaction to match the skills, tasks and preferences of its users. Additionally, effort is made to present to the user the information considered most appropriate for his/her current information needs. In the area of personalization, the meaning of context revolves around the user and his/her preferences (*user context*), and context information is captured either explicitly by user input or by recording the user's habits or through reasoning processes (*derived* or *interpreted context*).
Each of the three aforementioned domains (mobile computing, pervasive and ubiquitous computing and e-commerce) is of interest for m-commerce applications, and manages only one portion of the context approached from its own viewpoint, whether it may be *computational context* or *environmental context* or *user context*, without being interested in the *application-specific context* [6] (context related to the data and functionality of the specific application in question). Also, each of these domains collects context from different sources, either from logical sensors or physical sensors or by the users, using different mechanisms each time, with little or no standardization regarding the mechanisms for collection, processing and provision of context. Additionally, context management will on several occasions take place in ad-hoc ways inside the main application, which will use it to adjust its function. This practice however complicates the code, decreases software cohesion and increases software module coupling, resulting to decreased understandability, manageability, maintainability and code reusability.
Thus, m-commerce, being on the cutting edge of mobile computing, pervasive and ubiquitous computing and e-commerce technologies, faces the challenge of distinguishing the context that concerns it and standardizing the mechanisms for its collection, management and provision, taking into consideration the limitations set by the mobile networks and the mobile devices in use. Benou and Vassilakis [6] defined the context that concerns m-commerce applications (computational, environmental, user and application-specific context) and recording the metadata that it should bear, extended the UML class diagrams for its representation (as we overview in the Sections 3 and 4) and introduced a methodology for its definition in each case [6]. The next step to be made so as to utilize context from m-commerce applications is the provision of standardized mechanisms for its collection, management and distribution, suitably adapted to the particularities of m-commerce applications, which are dictated by the user's devices, networks and mobility.
Therefore, and given that m-commerce is interested in all types of context information, a software architecture must be made available, which will be able to:

1. manage all types of context information (computational, environmental, user, application-specific),

2. standardize the partial functions of managing the context information (collection, processing, distribution) regardless of the type of context and the mechanism of its collection (sensed, explicitly provided, or derived),

3. facilitate the implementation of adaptive applications, through the provision of standardized and uniform interfaces,

4. take into consideration the technological limitations of devices and networks.

3. The concept of context

3.1. Definition of context information and context domains

After analyzing the concepts of context [45, 64, 70, 71] in the domain of m-commerce applications, we define context as the set of all possible conditions and states that surround an electronic commerce operation, whereas we define context information as the set of data elements comprising the operation context. Context is therefore an abstract model, which – through a series of design and implementation activities – will be mapped into concrete context information elements; the latter will be finally utilized to support the adaptive services.
In the stages of m-commerce application analysis, design and development, we will mainly address *context information*; for the requirements of these stages, context information may be extended as follows [6]:
"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".
With the term "entity", they refer both to the term "entity" and the term "relationship" of the Entity Relationship Model (ER-Model) [37]. Benou and Vassilakis [6] also organize context information in four domains: i) *the user domain*, ii) *the environment domain*, iii) *the computing domain* and, iv) *the application-specific domain* (Figure 1). Context domains are also called *entity groups*, because they group together entities pertaining to the same actor. The *user domain* includes information relevant to the user. The *computing domain* includes information regarding the computing (and communication) infrastructure. The *environment domain* encompasses information regarding the real-world aspects of the user and computing surroundings, such as location, time, weather, etc. The *application-specific domain* contains information that is conceptually related to the particular application. We can observe that the first three groups are common to all m-commerce applications' categories, whereas the fourth (*application-specific* domain) is specialized for different application classes or even at application level.

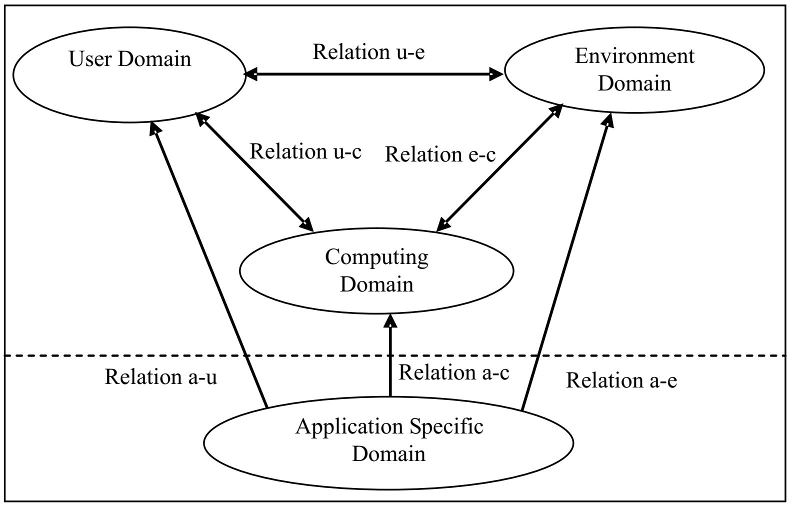


Figure 1. Types of Context Information.

The base entities of these four context domains may be interrelated; relationships may be established either among base entities within the same context domain, or among base entities belonging to different domains; these relationships, are called “associative entities”. Relationships between entities of different domains - namely the associative entities -, are depicted in Figure 1 as lines labeled “Relation u-e”, “Relation u-c”, “Relation e-c”, “Relation a-c”, “Relation a-u”, “Relation a-e”. Associative entities, derived by relationships among entities of the same domain, are naturally classified within the domain to which both associated entities belong to. For the classification of associative entities derived by interrelating entities of different domains, the analyst should consider the semantics of the relationship -if some of the interrelated entities are deemed more important, or even properties of the development philosophy (e.g. user-centric model vs. process-based models). For example, we may consider the associative entity “user access devices” (a relationship between the base entity “user” of the user domain and the base entity “device” of the computing domain), which is more naturally classified into the user domain, since the properties of this relationship are more user-oriented. The arrows’ directions in Figure 1 show the domain in which the produced associative entities could be classified.

3.2. The formal definition of context

Before dealing with the representation of context information, we will present a series of definitions that formalize this representation and further elaborate on the definition of context information that we have introduced in Section 3.1.
Definition 1:. An entity O_i is defined as a tangible or intangible real-world entity, such as a device, a place, a CD, an electronic product such as an mp3 file or a customer order.
Definition 2: The *context domain* is a high-level abstraction which partitions entities into the following categories: $\{$*user, computing, environment, application-specific*$\}$.
Definition 3: An entity O_i, may be modelled using a number of properties that describe aspects of the object O_i and a number of relationships, which describe how the entity relates to other entities. The set that includes all attributes (i.e. both properties and relationships) for entity O_i will be denoted as Ai = $\{a_{i,1}, a_{i,2}, ..., a_{i,m}\}$. Note that relationships may model the “part-of” semantics.
Definition 4: The *state* of an object Oi during a particular transaction *t* will be denoted as $S_i(t)$ and is derived by assigning a concrete value to each attribute of A_i. Each value may be atomic, record-typed, array-typed or any combination of the above. Orthogonally to their types, attribute values may be *sensed* (i.e. be gathered from physical or logical sensors), *explicitly provided* (i.e. the user enters the value) or *derived* (i.e. other values are processed to compute the value of the particular attribute). This is effectively the *context information* of object O_i during transaction *t*.
Definition 5: The context of a transaction *t* will be denoted as *C(t)* and is defined as the collection of all states of objects Oi which can be perceived as relevant to the user, the executed application or their interaction during the transaction *t*. Formally, C(t) = $\cup_k S_k(t)$ for all objects O_k that are considered relevant.
According to the definitions above, context information for a particular transaction can be denoted as a set of quadruples *(object context domain, object, attribute, value*). The element *object* denotes the object to which the particular piece of context information corresponds, e.g. a particular stock share (which includes information regarding “share description”, “share daily prices”, “shareholders registry,” etc.), location (which includes information regarding the longitude and latitude, but may include more “high-level” information such as “office”, “home”), etc. The element *object context domain* specifies the context domain to which the *object* belongs; the element *attribute* identifies the particular attribute that is measured; and the *value* element gives the exact value for the attribute within the specific transaction.
Note that the level of abstraction considered in the selection of an entity is dependent on the requirements of the application domain and the choices of the systems’ analysts. For instance, a PDA may be modelled as a single entity O_{pda}, having attributes representing its keyboard, screen, etc, or these parts may be modelled as separate entities and be connected to the O_{pda} entity through relationships of type “part-of” (cf. “FIPA Device Ontology Specification” [24]). These modelling choices do not affect the generality of the modelling method, since the goal, of capturing all the required context state information, can be accomplished independently of whether this information has been represented as a value of a property within a linked entity or as a component of a structure-valued attribute of a single entity.

4. Representing context: the extended UML class diagrams

According the definitions which have been presented in paragraph 3.2, context-related information, which encompasses a commercial transaction conducted via a mobile device by a moving user, may be organized into a sum of interrelated entities [29, 34, 62]. The attributes of those entities represent the context elements, which will be utilized by the services

implementing the adaptation of the m-commerce application.
These entities and their interrelationships can be illustrated as enhanced UML class diagrams [6], in which the enhancement refers to the inclusion of the special characteristics of context information and more specifically:

1. the *dynamicity of the value of each attribute*, given that context information is distinguished into *static* and *dynamic* [16], depending on how often it changes,
2. the *acquisition method* of each attribute value (sensed, explicitly provided, derived [16]),
3. the *metadata information* accompanying each attribute [39]. Such information is the *source* from which the context element is retrieved, the *timestamp* of the retrieval, the *confidence* for the correctness of its value, the *frequency* of its collection, the *validity period*, and the *metric* in which it is expressed,
4. the *need to record past values* for the information [6],
5. the possibility that *the type of a related entity may change*. This is important in adaptive services, since such a change may make new context items or sensing methods available or may terminate the availability of context items/sensing methods, which, in turn, may trigger changes to the user interface, processing or available data [6]. For instance, a change in the user's location from an instance of "office" to an instance of "shopping mall" may lead to the withdrawal of the user interface item "Read corporate memos" and establish the item "Get offers."

An example of an enhanced class diagram is illustrated in Figure 2. The notations used in this diagram are described in the following paragraphs.

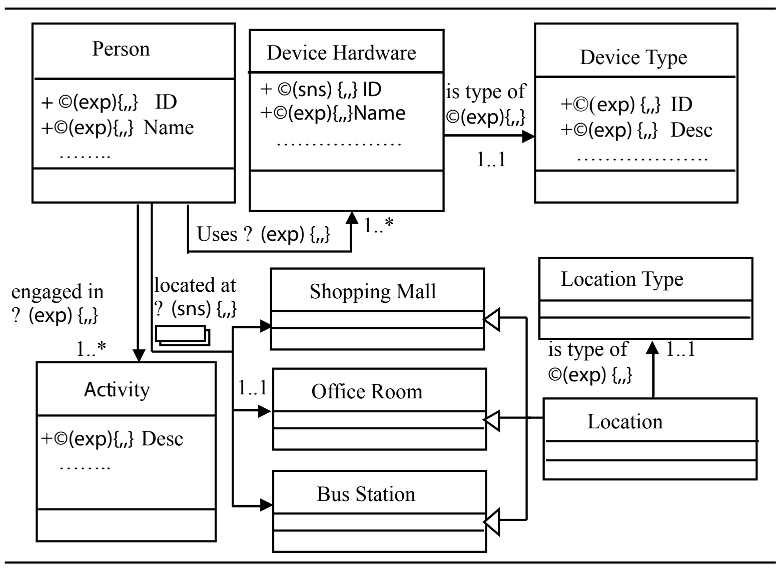
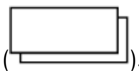
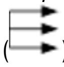


Figure 2. Context-aware UML class diagrams.

Consistently to the UML class diagrams, entities are represented using rectangles with three areas. The top area contains the class name, the middle area lists its properties and the bottom area lists the basic operation it provides. Its relationships with other entities are denoted using arrows (→), which are labeled with the relationship cardinality (e.g. 0..1, 1..1, 1..*) [51]. The special-type *generalization links* (─▷) denote the parent/child class relationships. Each attribute or relationship may be labeled with additional marks that denote the special nature of context information as follows:

1. *Information dynamicity* is denoted using:

- the symbol © for static information
- the symbol ≈ for dynamic information

2. The *acquisition method* is denoted using:
   - (sns) : for sensed information
   - (exp): for explicitly provided information
   - (drv): for derived information
3. The *metadata* associated to the context information is denoted as a series of values, with each value corresponding to a piece of metadata, e.g. {source, timestamp, confidence, frequency, validity period, metric}
4. the *need to record past values* for the information is illustrated through a double rectangle ().
5. the *possibility that the type of a related entity may change* is denoted using an arrow splitting to multiple ends (), one end for each possible type.

The stereotypes denoting the characteristics of attributes and relationships storing context information can be automatically mapped to code, relieving the developer of tedious and repetitive work and minimizing the possibility of errors. The code generation procedure can be summarized as follows:

1. all elements representing context include code to locate their context sources. This code is called when the object is initialized (within the constructor) and when the connection to the current source is broken. Additionally, callback methods are provided for receiving new values from the context sources and storing them accordingly into the context elements,
2. if information regarding information dynamicity (static, dynamic) is specified, then a slot will be created where the relevant dynamicity characterization will be stored, in order to be available to consumers of context information (e.g. adaptation managers). Context consumers, using this information and the appropriate metadata (e.g. confidence), will be able to assess the quality of context information [34],
3. information regarding the acquisition method (sensed, derived, explicitly provided) is used in the process of discovering available sources for the context information element, effectively filtering out sources not providing the designated method,
4. each item in the metadata list is handled according to its particular semantics. More specifically:
   (a) the presence of the *source* metadata element leads to the creation of a slot in which the context information source will be stored,
   (b) if the *timestamp* metadata element is specified, a slot will be created where the time of the context information acquisition will be stored,
   (c) the *confidence, validity period* and *metric* metadata elements, if specified, are retrieved from the context information source and stored in special slots. If the context information source does not make these relevant metadata available, a designer-specified default value is used,
   (d) Finally, the *frequency* metadata element is used to set how often a new value will be obtained from the context source. The code generation procedure creates a dedicated thread [63] to query context sources and store the obtained values accordingly.
5. recording past values is achieved by using a collection object (e.g. a Java or STL/C++ vector) to store all past values of the context element; each past value is accompanied with the timestamp the value was obtained. Set methods are redefined to append new values to the collection, while distinct get methods are provided to obtain the last or a designated range of values,

6. if the type of a related entity may change, the type of the variable storing the relationship is set to the *lowest common ancestor* in the class hierarchy, as determined by the generalization relationships in the class diagram (in the example of Figure 2, the lowest common ancestor of classes *Shopping mall*, *Office room* and *Bus station* is *Location*). If no lowest common ancestor can be found in the class diagram, a suitable generic type (e.g. *Object* in Java) is used. Additionally, callback functions are provided enabling the developer to execute actions when the type of the related object changes.

A UML profile and a plugin extending ArgoUML [2], with capabilities to (a) represent the context information extension and (b) adapt the code generation procedure to the characteristics specified for attributes and relationships representing context information, are currently under development, while an extension to the Eclipse platform is being designed.
Through the tracing of the context information onto extended UML class diagrams, the transformation of the vague concept of context of an m-commerce transaction into distinct data elements suitable for processing by computerized information systems has been achieved. Note that UML class diagrams are merely a means to the end of identifying the needed context data elements; the same goal can be attained using different approaches, such as the Mobl domain specific language [32] or the COPAL language [48], which allow for high-level, declarative language for programming mobile web applications including context-aware aspects.
The next logical step of the process of context utilization is its capture, management, storage and distribution from informational systems which will take into consideration the particularity of m-commerce applications.

5. The architecture for context management of m-commerce applications

5.1. The context information manager

In order to determine the context information needs of an m-commerce application a relevant methodology for extracting these needs should be employed. Benou and Vassilakis [6] have already proposed such a methodology. Since the context information of an m-commerce application has been identified (through Benou and Vassilakis' methodology or any other suitable methodology) and properly modeled (through extended UML class diagrams), the next step for the realization of a context-aware application is the designing of the subsystem that will manage the context.
The process of designing the system that will manage context information is common to all context-aware mobile commerce applications (CAMCAs). Despite the fact that the context that different CAMCAs manage can be quite diverse, a well-defined and extensible 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. The need for extensibility in this context refers to both the context factors the architecture is able to manage and the methods that can be employed for context acquisition. Such a standardized architecture will constitute a useful tool for speeding up the development of context-aware applications [33] and minimizing the probability for errors or omissions; furthermore, it will increase the potential for reusability, since context components developed for some application (e.g. context interpreters) will be able to be incorporated in other applications with few or no changes.
Both the international practice and the state-of-the-art [15, 46, 75] in the areas of pervasive and ubiquitous computing indicate that a context information management subsystem should be able to:

1. *capture* context information from its sources, which are physical and logical sensors, as well as the users. This includes the discovery of the context information sources within its vicinity,

2. *store* context information or parts of it, so that it can be exploited in subsequent situations,

3. *interpret* the context to a higher level of abstraction, which will be more meaningful (and useful) to the application that will use it. As an example, we can consider the interpretation of a *(longitude, latitude)* pair to a representation of the form *"home," "office"* or *"shopping mall"*,

4. *transit* the context information to the application that will use it. Transition should be supported in two modes, i.e. with the initiative being either on the application (request/response or pull paradigm) or on the context management system (pub/sub or push paradigm [52]), since both these modes are considered useful for CAMCAs [11].

In accordance with the above requirements, we will present below the design of the *Context Manager* module (Figure 3), which is further decomposed into the following components: i) the *Context Collection and Interpretation* module, *ii*) the *Context Distribution* module, *iii*) the *Context Storage and iv*) the *Context Discovery Agency*. The *Context Collection and Interpretation* module is responsible for gathering the context information from the various sources of the application environment and interpreting it to a higher level of abstraction. The *Context Storage* module is responsible for storing the context information for subsequent use. The *Context Distribution* module is responsible for distributing the context information to the applications that need it. The *Context Discovery Agency* is responsible for facilitating context information discovery for 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.
In the following subsections we will describe in detail each of the functional components that comprise the Context Manager, as well as the interactions between these components and between the context manager and the adaptation manager component of the m-commerce applications requesting its services.

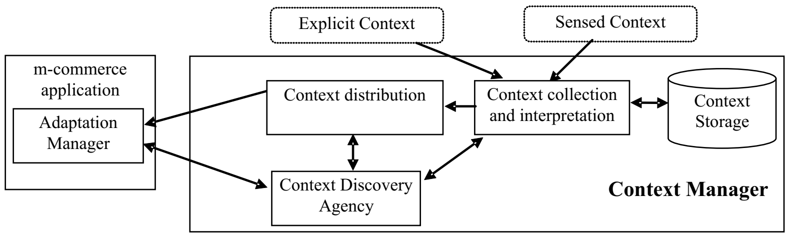


Figure 3. The Context Manager.

5.2. Context wrappers: collecting and distributing context

One of the tasks assigned to the Context Collection and Interpretation subsystem is the collection of 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) [26] 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). Logical sensors include the software modules that retrieve information from the main application database, (e.g. which user is currently logged in, which has been his/her observed behavior up to now, etc). 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 subsequently 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.
Physical sensors typically react to some environmental stimulus and generate numerical outputs which can be retrieved using low-level, device-specific protocols. *Logical sensors* are realized through software APIs, which the interested party may invoke to obtain the desired context information; logical sensors may read the context information values from a single physical sensor or combine values from multiple physical sensors [38]. Context information is made available from logical sensors either through a periodic monitoring process (*polling*) or through an available notification mechanism (e.g. an operating system API which provides notifications when additional storage space is made available). Finally, *user information sensors* – i.e. sensors delivering context information provided by the user (explicitly provided context information, e.g. information about the age or the likings of the user) – do not retrieve the relevant information through sensing mechanisms, but this information is made available through graphical interfaces or through *information integration procedures* (e.g. parsing and processing of XML files, retrieval of information from smart cards and so forth).
Direct incorporation of sensor-dependent code data into applications, usually necessitates low-level coding and leads to tightly-coupled applications with low portability and components with limited reusability [15]. Therefore, in order to decouple the applications from the details of the sensing process, we adopt the *context wrapper* approach. Fig-

ure 4 illustrates the concept of the context wrapper through a UML diagram. This software module undertakes the responsibility of reading context information from its source (through the ContextSourceAPI), so the peculiarities and idiosyncrasies of the particular context source are encapsulated in the ContextSourceAPI. 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 makes the context information available for exploitation through a standardized interface (ContextWrapperAPI), common for all kinds of context information.
Context wrappers can be viewed as the context domain counterpart of *device drivers* found in operating systems: device drivers undertake the task of accepting generic commands from the operating system (e.g. "read a block from disk") or passing data and status information to the operating system (e.g. "these are the requested data" or "the last command has failed in a retryable manner"), while at their other end, they communicate with the device in a device-dependent fashion, such as reading and writing device registers and processing device-generated interrupt signals [63]. The presence of the context wrapper, enables us to handle introductions of new context sources or modifications of existing ones by correspondingly creating a new context wrapper or modifying the existing one, leaving the rest of the CAMCA and the Context Manager system intact.

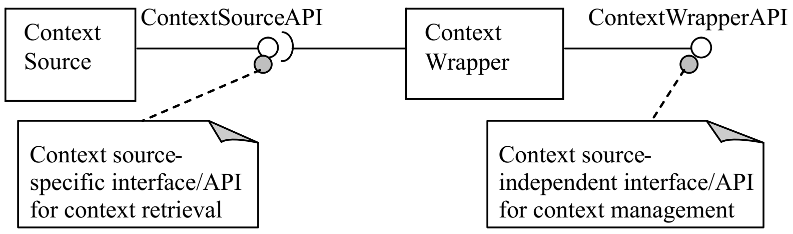

Context Source
ContextSourceAPI
Context Wrapper
ContextWrapperAPI
Context source-specific interface/API for context retrieval
Context source-independent interface/API for context management

Figure 4. A Context Wrapper.

As part of their internal operation, context wrappers may *cache* the last value obtained from the managed sensor in local memory to speed up the processing of the requests posed to them.
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, thus implementing the *pull* paradigm. As a response to such queries, the wrapper may probe the context source for a new value, use the last one retrieved from the context source and cached, if it is deemed valid or even retrieve a value previously stored in the context store,

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. They are sent whenever a subscriber-specified condition is met – e.g. for a wrapper managing a GPS device, a relevant condition could be "the location has changed by 200m or more". The subscription mechanism effectively implements the pub/sub paradigm,

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), which metadata are pertinent to each specific property (e.g. if a wrapper "measures" temperature, an indication whether temperature is measured in Centigrade or Fahrenheit degrees) and the list of the notifications it provides (e.g. for a wrapper measuring temperature, "temperature increased," "temperature dropped," "temperature changed," "temperature above threshold" and so forth),

5. they *register* themselves with the Context Information Discovery Agency. This registration allows the wrapper to be discovered by other software components (context information aggregation wrappers, adaptation managers, etc), as described in Section 5.5, below. 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. This may be practically implemented by having the Discovery Agency periodically broadcast requests for the specific service and automatically register to its repository those context wrappers that will respond to the broadcast. These broadcasts also allow the Context Information Discovery Agency to determine which wrappers remain operational and which have ceased functioning.

According to the above list of offered functionalities, the context wrapper interface depicted in Figure 4 can be refined as shown in Figure 5.
Essentially, context wrappers implement the *context collection* and the *context distribution* of the architecture depicted in Figure 3, with the code liaising with the context source interface (cf. Figure 4, Figure 5) implementing the *context collection* and the code realizing the context source-independent interface/API (and more specifically the ContextQuery and ContextNotification interfaces of Figure 5) being the *context distribution*. More specifically, the ContextQuery and ContextNotification interfaces of Figure 5 implement the distribution of context information to interested parties, while interfaces ContextReflection, ContextDiscoverable and ContextDataStoreCom facilitate aspects of the context distribution operation in the overall architecture.

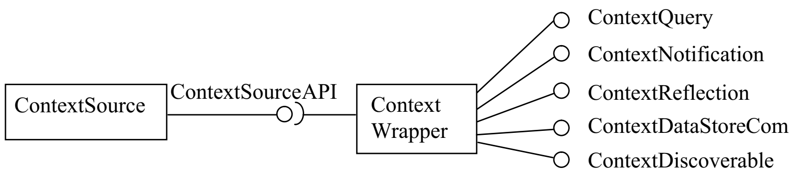


Figure 5. Refined Context Wrapper Interface.

The details of the interfaces through which the wrappers communicate with external software entities (i.e. details on the request response dialogues and notification messages) are described in [7]. 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.
A context wrapper provides information originating from a particular context source, i.e. physical or logical sensor, or the user. In many cases, however, the information required for an entity (person, location or object) is essentially an aggregation of the data elements provided by multiple context information wrappers, which may also need to be combined with additional information from the context information store. Therefore, it is necessary to introduce software components that implement this form of aggregation and which are called *context information aggregators*. Their functionality is similar to that of context wrappers, in the sense that they can respond to queries, produce notifications and store the context information they acquire. These software components can in turn *query* or *subscribe to* other context information wrappers so as to obtain the elements of context information they are interested in. Furthermore, they can retrieve information from the context information store, which may be used together with the data obtained through queries or incoming notifications to produce aggregated context information; the latter will be made available to interested parties for further perusal. Context aggregators are similar to logical sensors, differing only in the aspect that context information is retrieved from context wrappers instead of context source-dependent APIs.

5.3. The context information distribution module

The context information distribution module (i.e. the ContextQuery and ContextNotification interfaces of the context wrapper) undertakes the task of making the context information available to the interested parties (notably the adaptation managers of CAMCs) in a standardized and uniform manner. More specifically, it allows for the distribution of the context information according to both the *request-response* and the *event-triggered* paradigm [8], corresponding to the “pull” and “push” context information distribution [11]. According to the *request-response* (pull) paradigm, context information is given as a response to explicit requests, while according to the *event-triggered* (push) paradigm, the context distributor arranges for sending context information to *subscribers* when certain events occur. The context information distribution module is implemented through the query and notification mechanisms built in the context information wrappers and realized by the ContextQuery and ContextNotification interfaces, respectively, while, as noted above, these interfaces are complemented with interfaces ContextReflection, ContextDiscoverable and ContextDataStoreCom, with the latter three facilitate aspects of the context distribution module'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) 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. Under this scheme, the context consumer (i.e. the adaptation manager module of an m-commerce application) needs to be coded in a manner that can asynchronously receive and process incoming notification messages. Context information wrappers implement both the query and the notification mechanisms through interfaces that are uniform for all wrappers. Uniformity is a key requirement, since in this way applications may easily communicate with the wrappers, regardless of the wrapper implementation details.

5.3.1. The ContextQuery interface

The interface to the query mechanism has the form:
queryContext(timeSpecification, 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, only few of which are needed (e.g. 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 [6], the query mechanism allows the querying party to specify how "fresh" the context information is required to be through the *timeSpecification* designation [7]. The client defines to the wrapper the attributes it requires and the wrapper returns an appropriate reply [7]. 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.

5.3.2. 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 Figure 6). Each such subscription is complemented with a *notification condition* which specifies the circumstances under which the particular subscriber wishes to receive notifications. Besides the current value of the context information element, the condition may refer to the previously observed value, useful for producing notifications when the change has exceeded a certain threshold (e.g. temperature – previousNotificationTemperature > 0.5); it may also refer to temporal information (e.g. produce a notification every hour, regardless of whether the value has changed) or to context information element metadata (e.g. check whether temperature is measured in Celsius or Fahrenheit degrees to set accordingly the notification threshold within the condition).

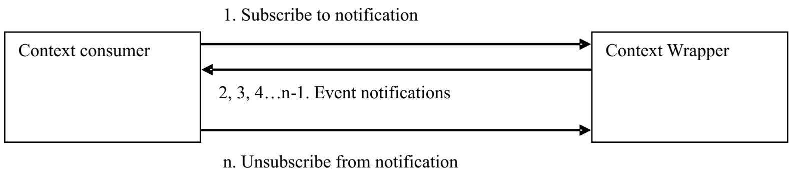

1. Subscribe to notification
Context consumer
Context Wrapper
2, 3, 4…n-1. Event notifications
n. Unsubscribe from notification

Figure 6. Publish/subscribe paradigm.

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 Figure 6). Finally, the context consumer may cancel its subscription through an *unsubscribe* request (flow *n* in Figure 6).

5.3.3. The ContextReflection interface

The ContextReflection interface allows context wrappers to be queried regarding the capabilities they offer and more specifically:

1. which context attributes it provides information on. For each attribute, a list of pertinent metadata is given, describing the attribute (e.g. a human-readable description), the value (e.g. units of measurement) and characteristics specific to the acquisition method (e.g. accuracy, period of value refreshment, minimum and maximum supported values) [7].

2. which notifications it publishes [7].

5.3.4. The ContextDataStoreCom interface

The *ContextDataStoreCom* interface [7] 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 when the algorithm employed by the *QueryAny* method [7] indicates that such a value should be returned.

5.3.5. The ContextDiscoverable interface

The *ContextDiscoverable* interface [7] allows for 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. It encompasses methods to register and unregister the context wrapper to and from the Discovery Agency, as well as methods in order to allow for the context wrapper to be discovered from the Discovery Agency [7].

5.4. Context interpretation

Context interpretation is the sub-module of the Context Collection and Interpretation 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 distribution module 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 [25, 74]. Context interpreters adhere to the context wrapper specifications. They *consume* context from context sources (context distribution module and the data store) and make it available to other context consumers. However, since the input data is gathered from standardized sources, context interpreters' implementation may be greatly simplified since there is no need to write context-source specific code; instead, data gathering may be specified declaratively by simply listing the context sources some pertinent parameters (e.g. whether data will be retrieved according to the push or pull paradigm, what the polling frequency for the pull paradigm is).
According to the specification above, 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 and (iii) the procedure that will perform the interpretation and (iv) the notifications provided, if any.
Context interpreters implement the ContextQuery, ContextNotification, ContextReflection, ContextDataStoreCom and ContextDiscoverable interfaces, thus being context distributors themselves, and providing the services described in Section 5.3.

5.5. The context information discovery agency

The context information discovery agency [7] implements facilities for storing information about the context providers (context information wrappers, context information aggregators, 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.
When a context information provider becomes active, it searches for the context information discovery agency and then registers to it. The details sent with the registration are (i) its ID, (ii) the address it can be reached at (e.g. if the communication is TCP/IP socket-based, the address will include the IP address and the port number), (iii) the attributes

it provides and the related metadata and (iv) the notification services it offers. Additionally, the context information discovery agency can itself register context information providers that have been discovered through a broadcast message. When a context information provider terminates its operation, it informs the discovery agency to remove itself from the context information discovery agency's registry. Context information providers may however terminate their operation abruptly (e.g. due to battery failure) and in these cases they cannot contact the context information discovery agency to perform the registry removal operation. In order to maintain its registry in an up-to-date state, the context information discovery agency periodically checks for the availability of the registered agents and automatically unregisters context information providers that fail to respond to it.

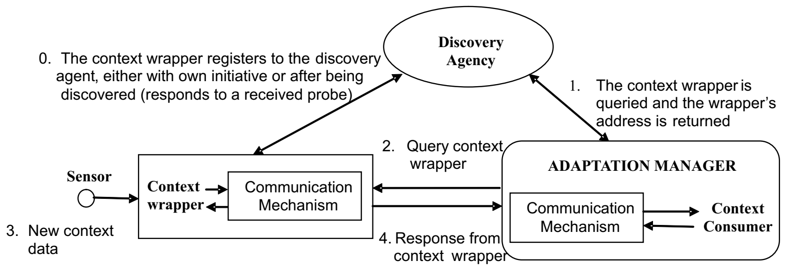

Discovery Agency
0. The context wrapper registers to the discovery agent, either with own initiative or after being discovered (responds to a received probe)
1. The context wrapper is queried and the wrapper's address is returned
2. Query context wrapper
Sensor
Context wrapper
Communication Mechanism
3. New context data
4. Response from context wrapper
ADAPTATION MANAGER
Communication Mechanism
Context Consumer

Figure 7. Example of an adaptation manager querying a context wrapper.

Finally, context information consumers (adaptation managers, context information aggregators and context information interpreters) may invoke the discovery agency to locate the context information providers which make available some particular context information. Figure 7 illustrates the complete message sequence from the point the Context Consumer module of an Adaptation Manager queries the discovery agency for a context wrapper's address, up to the point that it receives the requested messages (note that messages 2, 3 and 4 may be repeated multiple times).

5.6. The context information store

The context information store [7] allows for long-term storage of context information; this 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.

5.7. Implementation issues

Mobile commerce applications may be distinguished into three categories according to their architecture [23]. The first category includes applications that run exclusively on mobile devices and exchange data with a remote server (e.g. J2ME and Windows CE applications). The second category includes applications that run on some server and exchange only messages with the mobile device (typically SMS and MMS applications). The third category includes applications that run within a browser and exchange data with a remote server using a web protocol (HTTP, WAP, etc). According to Quah and Seet [56], the adaptation of these applications essentially comprises of taking into account the values of the context information elements to i) customize the data presented to the user (content adaptation) and/or ii) tailor the application's presentation properties (presentation adaptation) and/or iii) make the suitable modification of the application's functionality (functional adaptation). In order to achieve presentation and functional adaptation, context information must be available either when the application interface is generated (for browser-based applications or message-based applications) or at the location where the application is run (for "desktop-like" applications).
Regarding the first category of m-commerce applications (i.e. applications that run exclusively on the mobile devices), the interface is created at application development time, while the application is run later on the mobile device. On the contrary, for applications falling into the second and third category (message-based and browser-based, respectively), both the application interface generation and the application logic are hosted at the remote server and performed at run-time. Taking into account, however, the resource limitations of current mobile devices, the full-scale management and

exploitation of context information at mobile device-side seems infeasible. Especially if numerous context information elements need to be taken into account and advanced interpretation techniques are required; the need for constantly updating the volatile elements of context information also implies increased communication costs and battery consumption, which are two additional deterring factors for adopting the mobile device-side adaptation. Therefore, the architecture presented here is primarily suitable for mobile applications of the second and third categories (message-based and browser-based, respectively), where the remote server is mainly responsible for most tasks and the mobile device serves mostly as a presentation/user interaction apparatus. The proposed architecture can also be employed in applications falling in the first m-commerce application category (for "desktop-like" applications), provided that the context information elements managed are few and the adaptation tasks do not require extensive resources.

It has to be noted here that context wrappers, which are responsible for capturing and delivering context information, *may* be hosted in mobile devices, in all three application categories. Thus, context information providers that supply information regarding the user (e.g. identity, location) or the mobile device (screen size, input capabilities, etc) will naturally be accommodated in the mobile device. The mobile device may also host context information aggregators that capture data from context wrappers in its proximity (e.g. weather or traffic sensors). Context information from providers hosted in the mobile device will be transmitted to the central server, which will feed it accordingly to the relevant adaptation modules or deposit it in the context information store.

Currently all submodules of context manager are implemented in the Java 1.6 language, (with the exception of the context store which is under development) and have been successfully tested under Linux, Windows XP and Windows 7. Components that may run on mobile devices (context wrappers) have been successfully ported to compile under compiled Java ME SDK [54] under Windows mobile 5.0 (running on an HP iPAQ) and Windows mobile 6.0 (running on a Samsung Omnia) and Android (running on a Sony Ericsson Xperia X10). Specific context wrapper modules have been developed for the GPS receiver and for accessing application preferences (stored in the registry in Windows mobile and through the shared preferences class [1] in Android). The Java language was preferred against languages producing native code (e.g. C++) to promote portability, while Java ME provides a comprehensive set of libraries for accessing context sources on mobile devices; this set of libraries has evolved to cover the technologies introduced in the mobile devices market, accommodating thus the necessary extensibility.

The described architecture was designed and implemented as part of a broader adaptation-enabling approach, which includes, besides the Context Manager, the Adaptation Manager module, i.e. the module that will adapt the main application to the changes of context information. In Section 6, we present a case study which describes the adaptation of an application using the Context Manager presented in this section, and discusses the experiences from this development.

6. Case study — evaluation

In order to assess the effectiveness of using the Context Manager for the development of context-aware browser-based m-commerce applications, we present below a case study application using the ASP.NET framework [19] for the reservation of cinema tickets. In the following, we will only discuss the use of context and the adaptation of two pages, the *TicketsReservation* page and the *MovieSelection* page, since these include usage of all context categories (user, application, environment, computing) and all types of context (sensed, derived, explicitly provided).

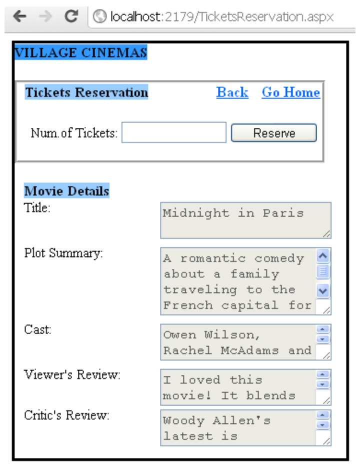
In order to evaluate the effectiveness of Context Manager and considering that the question "*how effective was the usage of Context Manager?*" is rather a general one and hence difficult to answer, we render it concrete by adopting the four specific evaluation criteria proposed in [28] in order to assess its usage: a) *Completeness*: this criterion determines whether Context Manager is sufficiently powerful and extensible to support all CAMCA requirements, b) *Complexity*: this criterion determines how hard it is to write code implementing Context Manager, i.e. it assesses the effort involved in developing the Context Manager itself, as well as the impact on the programmer's productivity, c) *Performance*: this criterion determines whether the Context Manager's architecture and its relevant implementation are good enough to support actual application workloads, i.e. to respond quickly in different usage scenarios (e.g. context sources registration, context queries, sending of notifications, etc), d) *Utility*: this criterion determines whether the architecture of the Context Manager can be used by others, in order to implement relevant modules for a wide range of applications with extensive demands for adaptation.

We proceed with the presentation of the case study application and then we will move on to present the results of our experimental evaluation.

6.1. Case study application development

After analyzing the requirements, we have concluded that for the *TicketsReservation* page the default and adapted functions listed in items (i)-(vii) below will be provided; for each one of them, we also list the context elements that will drive the adaptation procedure, as well as the categories they belong to.
i) The user selects one from the available films screened on that day (through the *MovieSelection* page – cf. Figure 8, sixth view) and the TicketsReservation page is displayed in order to implement his/her tickets reservation. The default behavior of the page (cf. Figure 8 - first view), is to give the capability to the user to define the number of tickets and by pressing a button to call the *onlyReservation* service, through which the reservation of seats takes place. Additionally, in the default behavior of the page, some basic information about the film is displayed and more specifically, the title of the film, a summary of the plot and the cast, as well as additional information about the film, especially information about viewers' reviews and film critics' reviews,
ii) According to the *paymentMode* user preference (which is explicitly provided context and belongs to user domain), there will be a capability for a) only reservation of seats (with a call to the *onlyReservation* service), b) reservation and payment via credit card (through navigation to the *PaymentViaCreditCard* page) and c) reservation and payment via bank account (after navigation to the *PaymentViaBankAccount* page),
iii) According to the *accessMode* user preference (which is explicitly provided context and belongs to user domain), there will be a capability to not display the information regarding viewers' review and film critics' review (cf. Figure 8 - second view),
iv) According to the *musicFriendly* user preference (which is explicitly provided context and belongs to user domain), there will be the capability for additional information regarding the music of the film and, depending on the current value of the *bandwidth* context parameter (which is sensed context that belongs to computing domain), the related information will be displayed either as text (cf. Figure 8 - third view) or as image (cf. Figure 8 - fourth and fifth views). Furthermore, according to the current value of the *bandwidth* context parameter, the resolution of the image will be adjusted (cf. Figure 8 - fifth view),
v) According to the user preferences (which are explicitly provided context and belong to user domain), the proper language for displaying the film's title and additional information will be selected (cf. Figure 8 - fifth view), as well as the values for font size, font weight (bold or not bold) and background color of the title and the data areas of the page will be determined (cf. Figure 8 - all views),
vi) According to the *cinemaCritic* element of the user context (which indicates whether a user is a cinema critic and is explicitly provided context that belongs to user domain) and the *moviePremiereStatus* element of the application-specific context (which indicates whether a particular film screening is a premiere), a message will appear on *TicketReservation* page informing the user of a 50% discount on tickets price (cf. Figure 8 - second view).
vii) Also, according to the *accessMode* user preference and the current value of the *bandwidth* context parameter, the system, instead of simply displaying a sole input box where the user may input the desired number of tickets, can display a table where the seating zones of the cinema theatre are listed along with the available seats of each zone (cf. Figure 8 - third view).
Views of the adapted page are shown in Figure 8.
Regarding the default behavior of the *MovieSelection* page (cf. Figure 8 - view (i) of this page), it will display information about the films of the current day and more specifically it will display in a table, information about the title of each film, its category, the cinema hall in which it will be screened, the hall type (indoor, outdoor) and the start time of the film. The adaptive operations which this page
offers (according to the presentation language and other presentation-related properties such as font weight, font size and background color), are the same as those which are offered by the *TicketsReservation* page and will not be further discussed. The additional adaptive operations which this page offers are the following:

1. If the *badWeather* element of the environmental context is evaluated to "true", the films which have been scheduled to be screened in outdoor cinema halls will be excluded from the table of the *MovieSelection* page displaying the offered films (cf. Figure 8 - view (iii) of this page). The *badWeather* element is an interpreted context element provided by the Context Manager module, and its value is derived after processing the values of the *temperature* and *rainProbability* context elements, with the first obtained through a web service offering current meteorological data, and the second provided by a web service providing weather forecasts. Both *temperature* and *rainProbability*


localhost:2179/TicketsReservation.aspx

VILLAGE CINEMAS

Tickets Reservation Back Go Home

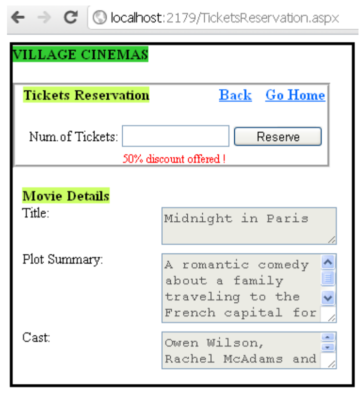
Num. of Tickets: Reserve

Movie Details

Title:	Midnight in Paris
Plot Summary:	A romantic comedy about a family traveling to the French capital for
Cast:	Owen Wilson, Rachel McAdams and
Viewer's Review:	I loved this movie! It blends
Critic's Review:	Woody Allen's latest is

(a) Default page behavior: on click of the 'Reserve' button, the *onlyReservation* service is called.

***TicketsReservation* page view**


localhost:2179/TicketsReservation.aspx

VILLAGE CINEMAS

Tickets Reservation Back Go Home

Num. of Tickets: Reserve

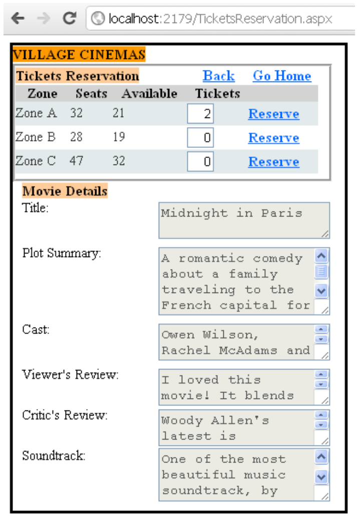
50% discount offered !

Movie Details

Title:	Midnight in Paris
Plot Summary:	A romantic comedy about a family traveling to the French capital for
Cast:	Owen Wilson, Rachel McAdams and

(b) Adapted page behavior: a content unit (viewers and critics' review) is hidden, colors are changed, a discount offer message is shown and on click of the 'Reserve' button, the user is navigated to the *PaymentViaCreditCard* page.

***TicketsReservation* page view**


localhost:2179/TicketsReservation.aspx

VILLAGE CINEMAS

Tickets Reservation Back Go Home

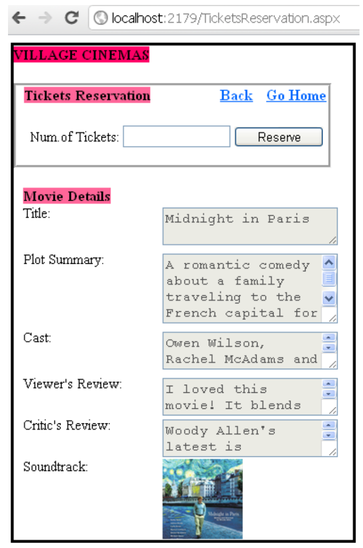
Zone	Seats	Available	Tickets	
Zone A	32	21	2	Reserve
Zone B	28	19	0	Reserve
Zone C	47	32	0	Reserve

Movie Details

Title:	Midnight in Paris
Plot Summary:	A romantic comedy about a family traveling to the French capital for
Cast:	Owen Wilson, Rachel McAdams and
Viewer's Review:	I loved this movie! It blends
Critic's Review:	Woody Allen's latest is
Soundtrack:	One of the most beautiful music soundtrack, by

(c) Adapted page behavior: two content units not belonging to default page behavior (reservation for specific seating zones and soundtrack) are shown, colors are changed and on click of the 'Reserve' button the user is navigated to the *PaymentViaBankAccount* page.

***TicketsReservation* page view**


localhost:2179/TicketsReservation.aspx

VILLAGE CINEMAS

Tickets Reservation Back Go Home

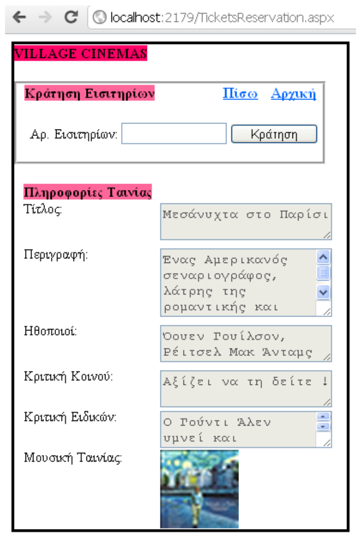
Num. of Tickets: Reserve

Movie Details

Title:	Midnight in Paris
Plot Summary:	A romantic comedy about a family traveling to the French capital for
Cast:	Owen Wilson, Rachel McAdams and
Viewer's Review:	I loved this movie! It blends
Critic's Review:	Woody Allen's latest is
Soundtrack:	

(d) Default page behavior: on click of the 'Reserve' button, the *onlyReservation* service is called.

***TicketsReservation* page's view**


localhost:2179/TicketsReservation.aspx

VILLAGE CINEMAS

Κράτηση Εισιτηρίων Πίσω Αρχική

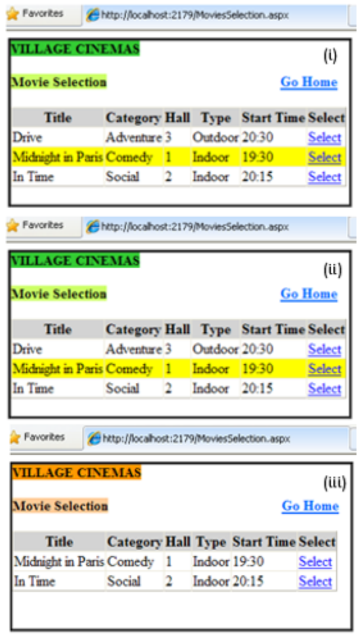
Αρ. Εισιτηρίων: Κράτηση

Πληροφορίες Ταινίας

Τίτλος:	Μεσάνυχτα στο Παρίσι
Περιγραφή:	Ένας Αμερικανός σεναριογράφος, λάτρης της ρομαντικής και
Ηθοποιοί:	Όουεν Γουίλσον, Ρέιτσελ Μακ Άνταμς
Κριτική Κοινού:	Αξίζει να τη δείτε !
Κριτική Ειδικών:	Ο Γούντι Άλεν υμνεί και
Μουσική Ταινίας:	

(e) Adapted page behavior: a content unit (viewers and critics' review) is hidden, colors are changed, a discount offer message is shown and on click of the 'Reserve' button, the user is navigated to the *PaymentViaCreditCard* page.

***TicketsReservation* page's view**


Favorites http://localhost:2179/MoviesSelection.aspx

VILLAGE CINEMAS (i)

Movie Selection Go Home

Title	Category	Hall	Type	Start Time	Select
Drive	Adventure	3	Outdoor	20:30	Select
Midnight in Paris	Comedy	1	Indoor	19:30	Select
In Time	Social	2	Indoor	20:15	Select

Favorites http://localhost:2179/MoviesSelection.aspx

VILLAGE CINEMAS (ii)

Movie Selection Go Home

Title	Category	Hall	Type	Start Time	Select
Drive	Adventure	3	Outdoor	20:30	Select
Midnight in Paris	Comedy	1	Indoor	19:30	Select
In Time	Social	2	Indoor	20:15	Select

Favorites http://localhost:2179/MoviesSelection.aspx

VILLAGE CINEMAS (iii)

Movie Selection Go Home

Title	Category	Hall	Type	Start Time	Select
Midnight in Paris	Comedy	1	Indoor	19:30	Select
In Time	Social	2	Indoor	20:15	Select

(f) Adapted page behavior: two content units not belonging to default page behavior (reservation for specific seating zones and soundtrack) are shown, colors are changed and on click of the 'Reserve' button the user is navigated to the *PaymentViaBankAccount* page.

***MovieSelection* page's view**

Figure 8. The Adapted *TicketsReservation* and *MovieSelection* pages views according Context Information.

context elements belong to environment domain, with the first being sensed and the second interpreted context element,

2. According to the *userDistance* context element which belongs to user domain and indicates the current distance of the user from the cinema (interpreted form the current user's location and the cinema's location, with the first captured by a GPS sensor and therefore sensed context and the second to be explicitly provided context which belongs to application-specific domain) and the *remainingTimeForMovieStart* element which is application-specific context element (interpreted from *currentTime* which belongs to environment domain and *movieStartTitme* which belongs to application specific domain, with the first sensed and the second explicitly provided context element), some rows on the table of the *MovieSelection* page displaying the day's films may will appear with a yellow background color (cf. Figure 8 - view (ii) of this page). More specifically, if the user's distance from the cinema is greater than 1,500 meters and the film will start in less than 15 minutes, the relevant row on the table will appear with a yellow background color.

Screenshots of the adapted *MovieSelection* page are shown in Figure 8.

6.2. Proposed architecture evaluation

After the description of our case study application, we return to the criteria set at the beginning of the paragraph to present the results of our experimental evaluation.
Completeness: The use of all categories of context information by the application (user context, computing context, environment context and application-specific context) and all types of context information (sensed, interpreted, explicitly provided) has shown that the provided types of context information from Context Manager efficiently supported the application demands in terms of context information and allowed the implementation of the adaptation for a wide spectrum of functions. The types of adaptation presented in the case study application, using context information provided by the Context Manager, are representative and suggest that, by using such information, other types of adaptation may be implemented, such as all context information usage scenarios sourced from the bibliography ([13, 17, 21, 30, 36, 44, 48, 75]). Additionally, the proposed architecture also includes the potential to use services offered by third parties as context information sources (e.g. a web service giving *rainProbability*), the use of context provided by distributed nodes and the use of context information originating from remote clients (e.g. sensors, services, mobile devices). The support of these context source categories is necessary for developing CAMCAs due to the user's mobility, the need for the use of environmental parameters and, more generally, context information derived from remote locations. Finally, the ability to manage metadata (e.g. *confidence* for the *temperature* or *rainProbability*) allows for incorporating more effective adapted functions in the application. For example, if the *confidence* characteristic for the *badWeather* context element has a value lower than a certain threshold, an adaptation rule can be defined specifying that films screened in outdoor theatres will not be excluded from table presented to the user, but a message will appear instead, informing the user that "*In case of bad weather, outdoor theatres will be closed. Reservation will be still valid for an indoor theatre. In such a case, users will be notified through an SMS*").
Complexity: The standardization of software artifacts of the Context Manager has allowed their easy development by distinct development teams (e.g. a team may implement the various kinds of context wrappers while another team may implement other modules, such as Discovery Agency); naturally, an integration test will be required after the distinct modules have been implemented and individually tested. The implementation time for the Context Manager was approximately 480 hours for 8,329 noncommenting source statements (NCSS), which results in a measured productivity of 17.35 NCSS/hr, which is within the acceptable range reported by the literature [22, 58]. The use of the Context Manager for the development of the aforementioned application does not pose an additional time burden, given the uniform and standardized interfaces which it offers. The adaptive application programmers have been found to quickly learn the API provided by the context manager, usually after having developed one or two simple applications; their productivity regarding the functionalities of sensing, collecting, interpreting and managing context has been measured to increase by 60%-70%, as compared to the productivity observed when the proposed context manager is not used.
Performance: The proposed architecture has been tested regarding its performance, in order to evaluate its ability to withstand different workloads. In the following, we present the results of our performance evaluation tests regarding the functionalities of (i) registering a context provider to the discovery agency, (ii) the time to perform a context query to a context wrapper attached to a physical device, returning the last value sensed (as opposed to requesting from the

context wrapper to fetch a new value from the physical sensor; this setup was chosen since the time to obtain a value from a physical device greatly varies among devices and physical interface speeds), (iii) the overhead imposed by a context interpretation mechanism, and (iv) the time needed to process an event and notify registered parties.
Regarding the test hardware/software configuration, the *context distribution*, *context discovery agency* and *context collection and interpretation* modules ran on a server equipped with 4 GB of memory and a Pentium dual-core processor running at 2.60GHz, which operated under Debian Linux version 6. In all cases we performed the tests used synthetic workloads, generated using the *Tsung* load testing tool [69], which ran on PCs equipped with 2GB of memory and a Pentium dual-core processor running at 2.60GHz, which operated under Debian Linux. One to twenty different machines were used in parallel for workload generation, depending on the number of concurrent clients simulated in the particular test; the maximum number of clients hosted in a single workstation was limited to 50, to ensure their efficient operation. The *thinktime* feature of Tsung was used to introduce delays between consecutive requests from the same simulated client when such a delay was needed; for instance in the performance test regarding the *query* interface, we used a *thinktime* with a uniform distribution U(32,63) (i.e. with minimum value equal to 32 and maximum value equal to 63), since it has been reported that most "user think times" fall in this range (e.g. [3]).
The client machines were interconnected to the server using a 1GB Ethernet switch. In both the client machines and the server machines we used the *netem* tool [66] to introduce packet delays in the client-to-server communications, emulating thus different network speeds (WLAN 802.11b with a bandwidth of 11Mbps; UMTS/HSDPA with a bandwidth of 7.2 Mbps; and Bluetooth 2.0 + EDR with a bandwidth of 3Mbps).
Finally, all experiments were run for 2 hours, to ensure that the simulation reaches a steady-state producing thus reliable results.

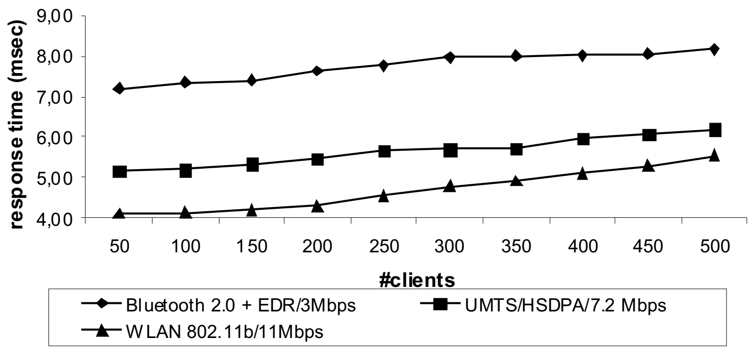


Figure 9. Performance test results for context source registration to the discovery agency.

Figure 9 depicts the performance test results for the benchmark concerning the registration of context sources to the discovery agency. In this experiment, a *thinktime* of 1 minute was set, to simulate the periodic polling of the discovery agency to the context sources to check if they are operative. As can be seen from the diagram, the worst-case response time (when 500 context sources register over Bluetooth communication links) is approximately 8msec. UMTS and WLAN networks expectedly offer better response times.
Figure 10 depicts the performance test results for the benchmark concerning the querying of a context wrapper for the last value sensed (and retained in its memory). In this experiment, a *thinktime* following the uniform distribution U(32,63) was set, to simulate the client page request rate. The worst-case response time (when 500 context sources register over Bluetooth communication links) is approximately 6.8msec, while the respective worst-case response times for UMTS and WLAN are 5.16msec and 4.61msec respectively.
Figure 11 illustrates the measured overheads incurred from using a context interpreter, mapping Celsius degrees to Fahrenheit degrees. Clearly, this interpretation is a trivial one, the objective however of the benchmark is to quantify the overhead of using the interpretation mechanism, and not the time needed to run some arbitrary interpretation algorithm, which is dependent on the algorithm's complexity and implementation efficiency. In this experiment, a *thinktime* following the uniform distribution U(32,63) was set, to simulate the client page request rate. Note that these results are independent of the client network infrastructure, since the interpreter runs on the machine hosting the Context Manager.

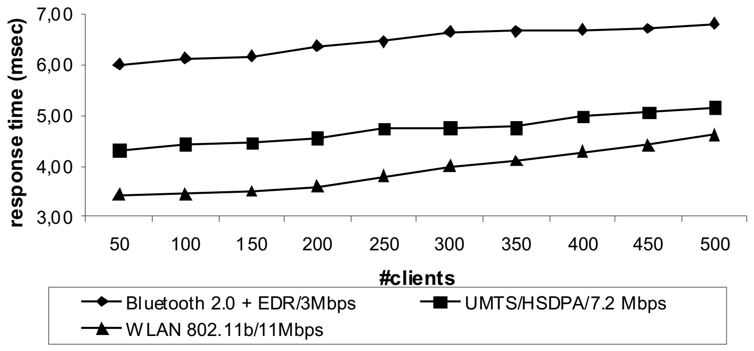


Figure 10. Performance test results for querying a context wrapper.

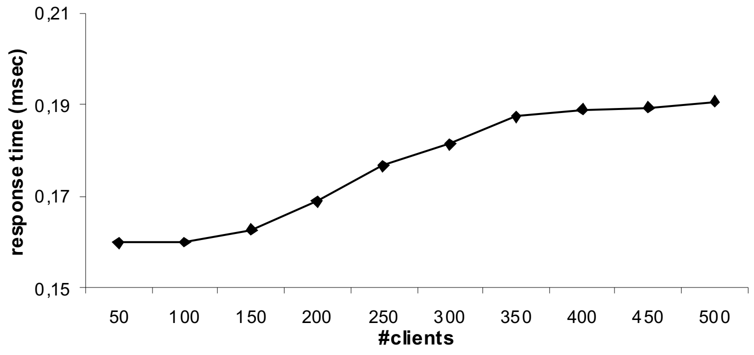


Figure 11. Context interpretation mechanism overhead.

The worst-case response time (when 500 clients request a page involving an interpretation) is approximately 0.2msec.

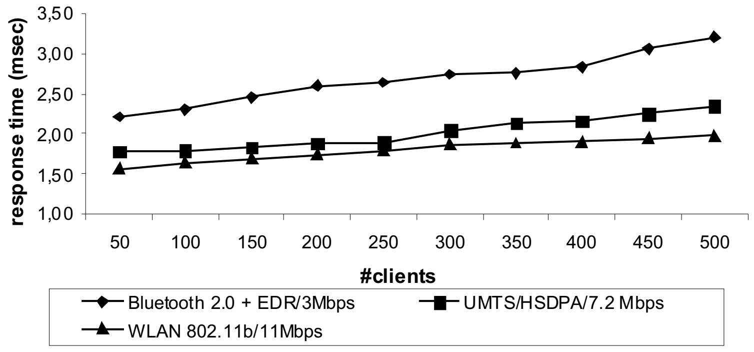


Figure 12. Event processing and notification subscriber update.

Figure 12 depicts the benchmark results regarding processing an event and notifying the registered parties, i.e. it concerns the performance of the "push" paradigm for transiting context information. The scenario, evaluated regarding the notification mechanism, was a sudden rain in the area surrounding a shopping mall (detected by a weather sensor in the mall's vicinity), which triggered the notification of context interpreters employed by a number of applications

delivered by retail stores in the shopping mall (stores supplying umbrellas or raincoats, cinemas, parking areas, etc) as well as the mall information application operated by the mall management. Individual mall customers are notified appropriately from the m-commerce application they use, through an area of their web page which employs the Ajax push engine [65] to receive incoming messages from the m-commerce application server. The measured times illustrated in the figure pertain to the time needed to notify all clients (#clients in the above diagram).
Utility: Under the light of the experiences amassed from developing a number of CAMCAs using the proposed approach, we can state that the applications designed using the WebML model [12] (or any similar model), are adequately standardized and may be adapted using the same pattern; hence, the Context Manager architecture is able to support a wide array of applications. Also, the provision of interpreted context information (e.g. *badWeather*) has relieved the developer of the main application of the additional work required to transform the context information to higher abstraction levels that are needed for the application under development. This provides standardized software artifacts, which may be used by other applications, simplifying and accelerating their development. The isolation of the sensing mechanisms from distribution of the context information facilitates the replacement of any context provider. Lastly, the encapsulation of the capturing, management and distribution of the context information by a separate module (i.e. the Context Manager) has relieved consumers (e.g. adaptation managers) of the need to liaise with multiple distinct context sources (potentially accessible through diverse interfaces).

7. Related work and discussion

Insofar, numerous researchers have proposed software systems that aim at managing context information. Each approach depends on special requirements related to the location of sensors (local or remote), the number of users (one user or many) and the resources available at the user access devices, which may be high-end-PCs or small mobile devices [4]. Thus based on these requirements, three types of architectures for context management can be distinguished [13]:
(i) Direct sensor access: In this approach, sensors embedded in the mobile device are used. The client software collects the context information directly from the sensors, which implies that drivers for the sensor devices are hardwired into the application. Besides severely limiting the application portability and expandability due to the presence of drivers within the application [57], this approach is not suitable for distributed systems because it cannot manage well concurrent access to sensor devices that may be needed by multiple applications (e.g. a GPS-based driving aid and a location-based e-commerce application both requiring access to the GPS device). Examples of such systems are Active Badge [72] and Context-Aware Pocket PC [35],
(ii) Middleware-based: This approach introduces a layered architecture aiming to hide low-level sensing details. Once the middleware has been standardized, any changes in the available sensing devices or any other such details do not necessitate any modification to the application, thus application portability is promoted. The middleware may be hosted in a specific computer or may be distributed in multiple devices, following the P2P paradigm. The system proposed for the Odyssey project [53] is an example of a system following this approach,
(iii) Context server: This approach introduces a central component termed *context server*, which gathers context information from all sensors. Clients communicate only with this server to obtain context information; therefore, they are offloaded from low-level sensing details and resource-intensive operations.
In the following paragraphs, we review related work from the three architectural types described above.

7.1. Direct sensor access-based approaches

Ortiz et al. [55] deals with the issue of adaptation of web services and the mobile client without using a Context Manager for the management and processing of context, and targets exclusively to mobile applications executed on the client (e.g. J2ME applications). The context elements used cover a small subset of context (e.g. device type, platform type, certain user preferences) and are relayed to the server executing the (business logic) web services, by adding parameters to the SOAP message's header. The suggested architecture is suitable for client-side applications which consume web services and with a limited need for adaptation, since few context parameters may be used and part of the adaptation takes place on the resource-constraint client. The absence of a Context Manager module in this approach does not allow (i) context processing (interpreted context, aggregated context), (ii) the utilization of context derived from sources other than the device (e.g. a weather service) without the selection of context provider to be hardwired inside the application, since context source rebinding mechanisms consume additional resources to identify non-operational context sources and locate potential alternatives; this is not feasible to be performed on most mobile devices, (iii) the use of metadata for assessing the quality of context information (iv) use of a context storage in order to reduce the size of data sent from the client to the server (e.g. user preferences) and (iv) the facilitation of transparent use of the context even if the context source in unavailable (e.g. the client device cannot communicate with the GPS satellites; the last position obtained from the GPS system could of course be used, but this will be done by writing extra code in the client, as opposed to an automatically performed action by the Context Manager to retrieve the last value from the context store).

7.2. Middleware-based approaches

The Context Toolkit [17] 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. This architecture is middleware-based and is mostly oriented to sensor-based applications. It has a limited suitability for m-commerce applications, it doesn't use a central management server but uses a Discovery Agent. The extent of context managed is limited. Other examples of peer-to-peer systems are the MANIP system [49] and the SALES system [14], although the latter additionally uses a centralized component.
The architecture proposed in the Hydrogen project [36] 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. This architecture is middleware-based and uses a central management server located on the mobile device, without the use of a Discovery Agent. The framework runs on the mobile device and mostly addresses stand alone applications over wireless LANS (including Bluetooth) for context exchange between devices. The extent of context managed is limited due to the scarceness of resources on the mobile device. The considered context mainly includes time, location, device characteristics and personal properties.
The ESCAPE framework [68] has been developed in order to support the management of context information in emergency situations such as natural disasters. Although this system has been developed to support an m-business application rather than an m-commerce application, we discuss it as another peer-to-peer paradigm in the wider area of m-commerce. This specific architecture has been developed in order to support a certain model of process implementation, in which front-end *teams of individuals* are active on certain *sites* (e.g. a village) where the *situation* occurs, in order to conduct situation *responses* (e.g. rescue subjects) and which are supported by back-end teams. The architectures consist of two components: i) the *Context Information Management Service (CIMS)* executed on each individual's device, which collects and processes context information and ii) the *Situation Context Information Management Service (SCIMS)* executed on the back-end system and which collects all context data related to a situation. The context data collected on the device of each team individual are forwarded to the team manager's device, which in turn forwards them to the back-end system. In this specific architecture using Service Discovery, a CIMS can publish information about itself to other devices by exploiting multicast service discovery based on SLP (Service Location Protocol). The architecture uses a peer-to-peer (P2P) data exchange model, which is suitable for each situation demanding the exchange of context information among individuals on a *site*, but which is unsuitable for m-commerce applications requiring a server-centric approach, since the utilization of context information takes place on the server. Moreover, the component structure of the architecture is oriented towards this specific use without the capability of generalizing for m-commerce applications. For instance, each CIMS is described by a triple *(teamID, individualID, serviceURL)*, while in m-commerce applications *teamID* normally does not apply, and moreover this representation cannot accommodate types of context such as application-specific context parameters. Additionally the proposed XML schema for context representation is specifically targeted to emergency situations, handling context provenance in the situation/site/response/team/individual model, but without the provision for incorporating useful extensions such as context metadata. Finally, CIMSs require to run multiple components (query and subscription, data aggregation and publishing, context interpretation and storage, service discovery, SOAP servers, etc), which is a resource-demanding setting, and this further limit the applicability of this approach for m-commerce applications that need to reach a wide public, without imposing restrictions on the access devices.

7.3. Context server-based approaches

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, i.e. the context server paradigm. This approach has been proposed by Korpipää et al. [44] 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. This architecture is middleware-based and is suitable for applications running on the mobile device. It uses a central management server which runs on the mobile device without a Discovery Agent. The extent of the context managed as well as the extent of applicable context processing are limited, due to the scarceness of resources on the mobile device
The SOCAM architecture (Service-oriented Context-Aware Middleware) [30] also employs a centralized server termed *context interpreter,* which collects data from distributed context providers and offers it, in processed format, to client applications. This architecture is middleware-based and is mainly oriented to smart spaces, e.g. smart vehicles. It uses a central management server which runs on a resource-rich stationary computer and uses a Discovery Agent. The extent of the context managed is limited and partitioned into different domains.
Another centralized middleware-based approach that has been designed for context aware mobile applications is the one proposed by Fahy and Clarke [21] 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. This architecture is middleware-based and is mainly oriented to smart spaces, e.g. presentation areas. It also uses a central management server which runs on a resource-rich stationary computer but maintains a copy of the context in the mobile device and uses a Discovery Agent. The extent of the context managed is limited, mostly including sensor data and location from GPS.
Copal [48] introduces a runtime middleware and a programming model for context provisioning. It provides a loosely-coupled and modularized architecture, whose main components are i) the *Device Services* providing context which are, in essence, the context Publishers that register themselves to *Publisher Registry*, ii) the *Context-aware services*, which are essentially context Listeners that core COPAL has to notify when specific events occur, iii) the *core COPAL*, and iv) the *Plugins*, which are optional architectural components and expand the functionality of core COPAL, e.g. through Localization and QoC (Quality of Context), possibly adding attributes of source location and QoC [47]. The main components in core COPAL are i) *Context Type*, a central component, which represents the different context elements provided by Publishers, ii) *Context Query*, which provides the context events, and iii) *Context Processor*, which offers a wide range of context processing, such as filtering, abstraction, differentiation, enrichment, peeling. The use of COPAL-DSL (Domain Specific Language) to generate code skeletons and deployment artifacts from context components indeed reduces implementation effort and enhances automation of context-aware service development. The advantages of this architecture include the capability for integrating new context sources, the use of metadata for the assessment of context information quality and the provision of interpreted context. However, there are also some disadvantages in relation to the suggested architecture stemming from the fact that it is oriented to the management of sensor-derived context, not directly supporting explicitly provided context and application-specific context. The architecture supports the event model (publish-subscribe mechanism) and not the query mechanism (request-response) with the exception of historical data. The aforementioned disadvantages of the COPAL architecture do not render it suitable for use *as is* from m-commerce applications.
CoBrA (Context Broker Architecture) [13] 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.* This architecture is middleware-based and is mainly oriented to smart spaces, e.g. intelligent rooms. It uses a central management server which runs on a resource-rich stationary computer, without a Discovery Agent. The extent of the context managed is relatively large. Other systems oriented to smart spaces are [10, 27, 40].
The *Location-based Publish/Subscribe Service (LPSS)* is also a noteworthy approach. *LPSS* has been implemented in the context of developing the Pervaho platform [20], which essentially is an implementation of a context-aware

public-subscribe mechanism, on top of the Java ME platform. The particular system employs a central server for the management of the context and extends the classic content-based and topic-based publish/subscribe mechanism incorporating environment and application-specific context. Although the comprehensiveness of the context information elements used in this approach was limited (only the location of the user and some of his/her preferences were considered), performance evaluation has shown that the time needed to process new subscriptions and publications, as well as the delivery of "matching information" within most mobile settings remain within acceptable limits.
The ContextServ platform [61] has been developed in order to support the rapid and flexible development of context-aware web services. One of its main components is the Context Manager which provides two kinds of context: *atomic* (low level) and *composite* (processed or aggregated) context. The ContextServ platform also contains the *ContextUML modeler* and the *RubyMDA transformer*. The *ContextUML modeler* offers a graphical user interface, allowing service developers to specify context-aware web services using Context UML language [60], through the *context binding* and *context triggering* mechanisms. Once a context-aware web service has been defined using the *ContextUML modeler*, the *Ruby MDA transformer* converts the service model into executable web service specifications, including BPEL and WSDL specifications and deploys the BPEL process to the web server exposing it thus as a web service. This particular process focuses more on the use of the Context Manager for the provision of context-aware web services and the automation of their production process instead of providing a generalized framework for supporting applications (the subject of our paper). From the description of Context Manager in [61], it follows that Context Manager supports low-level context as well as interpreted and aggregated context, however no details are given about the architecture of the corresponding components managing it. Regarding the representation of context, it seems to be using individual values (atomic or composite), and this representation (individual values) is not suitable for applications demanding a large number of context elements. No standardized mechanisms for context retrieval (e.g. query mechanism) are listed, it does address the issue of metadata, it has limited triggering mechanisms and it does not have a discovery agency for the location of context services and for the facilitation of the selection of context provided by several different context providers. In addition, it does not include a Context Data Storage, which is a necessary component for using historical context as well as in cases when it is not possible to communicate with a context provider. Lastly, this architecture focuses on processing context provided by sensors, not covering thus context categories such as explicitly provided context.

7.4. Discussion

The context management systems overviewed in the above subsections 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 and (c) the spectrum of operations they offer for context management. 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 manage context originating from physical sensors and 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 Wi-Fi, Bluetooth and Ethernet networks, as opposed to GPRS/UMTS networks used in other settings. From the analysis of the aforementioned propositions for context management in the field of pervasive and ubiquitous computing, we can notice that i) they are either smart space-oriented (for places such as smart homes, smart vehicles, smart university campus, etc), with the main goal of supporting context provisioning to service level and utilization of context originating from physical sensors, mostly, without standardized discovery or expansion capabilities to m-commerce applications (for the use of other types of context information and the support of other types of adaptation such as presentation adaptation which are required for m-commerce applications, in which the presentation is of particular importance), or ii) the context management system is executed on the mobile device, with the result of the context management functions developed being very limited (lack or limited interpreted or aggregated data, stored data, metadata), due to the scarceness of resources on the mobile device. Also, iii) they manage a small portion of the context and use specific context models that suit to their needs (e.g. key-value models [17], XML models [43] or domain-oriented ontologies [13] such as space ontologies) and specific context processing operations, due to the fact that context data processing is strictly defined on the basis of specific context-aware scenarios [31] that the system is going to support. Therefore, each of the aforementioned systems can only handle a restricted set of specific scenarios with specific and limited demands for context information.
In reference to context management and its utilization from mobile commerce applications, we outline and assess some representative examples in the following paragraphs.

The Delivery Context Ontology [9] is an important source of information that can be exploited to create context-aware applications, and provides a formal model of the characteristics of the environment in which devices interact with the Web or other services. The Delivery Context includes the characteristics of the Device, the Runtime Environment, the Network providing the connection and the Physical Environment. The Ontology is formally specified in the Web Ontology Language (OWL) [73]. It defines a normative vocabulary of terms (classes, properties and instances) that models the different Properties, Aspects and Components of a Delivery Context. The vocabulary developed in this ontology facilitates the interoperability of the applications and is mainly suitable for the implementation of applications using web services of different manufacturers; however, it is very specific and demands from the designer adherence to this degree of analysis of context elements and use of certain context classes. In addition, no methodology is recommended for the definition of context demands for each application and the context demands will have to be adopted as recommended by the particular ontology. Also, it does not use the concept of metadata of context elements (with the exception of measurement units representation), which is an essential element for the assessment of available context information [34]. Another disadvantage of this approach is that it does not deal with application-specific context, as it is not intended to model properties which can be application -or domain- dependent. The particular suggestion of Delivery Context Ontology comprises a noteworthy approach of context modeling and not a suggestion for capturing, managing and distributing context.
From the analysis of approaches to context management in the field of mobile commerce presented above, it follows that there has been no proposal of a well-rounded architecture, which:

1. manages all types of context (computing, user, environment, application-specific), regardless of the context source they are derived from (physical sensors, logical sensors, users, etc.) and in a domain-independent way: most of the times only few simple context elements are used, while at the same time enabling the application designer to design the context model in accordance with the current needs of its application,

2. collects the context information in a comprehensive and uniform way, through standardized interfaces, facilitating the task of developers (who only need to learn a single context acquisition interface) and promoting component reusability across applications.

3. exploits the full potential of context sources: in most approaches, only simple context element values are drawn from a service, even though the context source can offer more than one context attribute and their metadata,

4. offers the capability for advanced techniques of interpretation or aggregation of context information: many approaches simply use the context values as drawn from their sources.

5. offers the capability to use and exploit metadata of context information in order to assess the quality of the provided context information: with the exception of COPAL [48], this capability is either completely absent or is very limited,

6. offers the capability to store the context information in order to disconnect context production from context consumption and to support context data availability in cases in which the context provider is unavailable or the network is malfunctioning (low bandwidth, inability to access a context service). Only the ESCAPE framework offers an on-device "lightweight data storage" facility and the capability to later offload context data to a central server for post-situation studies, while COPAL [48] supports context persistence through a plugin,

7. can readily accommodate new context information elements, simply by plugging-in the context sources. On the contrary, many of the approaches proposed in the literature handle only a very limited and fixed set of context information elements.

Recapitulating, mobile commerce applications exhibit a very important lack of an architecture that efficiently manages context, does not place limitations on the range and number of context parameters used (allowing thus for comprehensive adaptation) and provides capabilities for context processing (e.g. interpretation and aggregation) and storage.
Our proposal 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, apart from addressing the disadvantages mentioned in the previous paragraphs, is suitable for mobile commerce applications for the following reasons:

1. A single software component (context manager) will manage issues stemming from concurrent access to sensors,
2. Centralized management of context guarantees context data availability and relieves the mobile devices from the burden of managing context themselves. If we consider the large number of context sources and the production rates usually associated with real physical phenomena (some context aspects change very often, and their associated sources can produce data with very high rates), the context data processing and dissemination performed by mobile devices could consume a considerable portion of the available resources, thus affecting the final experience perceived by mobile users. 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,
3. The user interface (data, presentation properties, functionality) of web-based m-commerce applications is created 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 aggregators/interpreters.
4. The use of distributed context wrapper components allow for capturing of context from remote locations (mobile devices, weather and traffic sensors, etc),
5. 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. Specific components (mostly context source to context wrapper communication) can however be implemented using more lightweight technologies (e.g. RPC [67] or even proprietary protocols) to better suite sensors with limited resources,
6. The adopted approach for context management allows for hiding the low-level sensing details from all context consumers (aggregators/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 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 commerce application development and reducing the possibility of bugs,

Additionally, the proposed architecture is general enough to support the context needs of a wide range of m-commerce applications, while the implemented components for one application could be used from other applications without or with short-range modifications.
Table 1 summarizes the differences and overlaps between existing approaches and our proposed architecture for Context Management.

8. Conclusions

The study of the behavior of the user of m-commerce applications coupled with the study of his/her environment allows us to delimit and specify the context information that is of value for a particular m-commerce application. In the m-commerce domain, the exploitation of this information for delivering innovative and enhanced services offers a competitive advantage for attracting new customers and maintaining existing ones. However, we need to appropriately design the subsystems that will manage, distribute and exploit the context information for m-commerce applications.
The design of a 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, reusability and speed of application development. In this paper, we have presented a high-level software architecture for context information management and distribution, 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. In this paper, we have also presented the main differences and advantages of the proposed architecture against other proposals in the literature, while the case-study application and its relevant evaluation, presented in Section 6, show the effectiveness of Context Manager's usage and the accommodation of all relevant requirements for context information stemming from the study of special m-commerce applications characteristics.

Table 1. Differences and overlaps between existing approaches and proposed architecture.

Property / Context Management System	*Use of middleware*	*Use of a central management Server*	*Use of a resource-rich stationary computer as Server (storage, interpretation, distribution)*	*Use of a Discovery Agent*	*Comprehensiveness of managed Context*	*Suitability for m-commerce applications*
Proposed Approach	Yes	Yes	Yes	Yes	Yes	Yes
Context Toolkit [17]	Yes	No	No	Yes	Only data directly from sensors	Limited, mostly oriented to sensors-based applications
Korpipää, Context Management System [44]	Yes	Yes	No, the central management server runs on the mobile device	No	Limited because, the central management servers runs on the mobile device	Yes, but directed to applications running on the mobile device, not for browser-based ones
SOCAM [30]	Yes	Yes	Yes, needs one server for each domain (e.g. vehicle domain)	Yes	Limited and partitioned in different domains.	Suitable for smart spaces with limited extent e.g. smart vehicles
CASS [21]	Yes	Yes, it maintains however a copy of the context in the mobile devices.	Yes	No	Limited, mainly context from sensors and location from GPS	Mainly oriented to smart spaces with limited extent e.g. smart presentation areas
CoBrA [13]	Yes	Yes	Yes	No	Yes	Suitable for smart spaces with limited extent e.g. intelligent rooms
Hydrogen [36]	Yes	Yes	No, located on the mobile device.	No	Limited, due to the scarceness of resources on the mobile device. Mainly considers time, location, device characteristics and personal properties	The framework runs on the mobile device and mostly addresses stand alone applications over wireless LANS (including Bluetooth) for context exchange between devices
Copal [48]	Yes	Yes	Yes, but with limited capabilities	Yes	No, only context from sensors	Suitable for smart spaces with very limited extent
ESCAPE framework [68]	Yes, but uses the peer-to-peer model	Yes but limited: the bulk of context management is performed on the mobile device	Yes, but with limited capabilities	Yes, but with limited capabilities	No, context for the specific emergency domain	Suitable for emergency situations

References

[1] Android developers. Using Shared Preferences. http://developer.android.com/guide/topics/data/data-storage.html#pref, 2011
[2] ArgoUML, http://argouml.tigris.org/, 2011
[3] Arlitt M., Characterizing Web User Sessions, ACM SIGMETRICS PERF E R, 8(2), 50–63, 2000
[4] Baldauf M., Dustdar S., Rosenberg, F., A Survey on Context- Aware Systems, INT J AD HOC UBIQ CO, 2(4), 63-277, 2007
[5] Benou P., Bitos V., Developing Mobile Commerce Applications, J ELECTRON COMM ORGAN, 6(1), 63-78, 2008
[6] Benou P., Vassilakis C., The Conceptual Model of Context for Mobile Commerce Applications, J ELECTRON COMM RES, Springer-Verlag, 10(2), 130-165, 2010
[7] Benou P., Vassilakis C., technical report TR-SSDBL-11-001, Department of Computer Science and technology, University of Peloponnese, http://sdbs.cst.uop.gr/?q=node/261, 2011
[8] Biegel G., Cahill V., A framework for developing mobile, context-aware applications, In Tripathi, A., Iftode, L., Nahrstedt, K., Nixon, Patrick (eds), Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communication (14-17 Mar. 2004 Orlando FL USA), IEEE Computer Society Press, 2004, 361–365
[9] Cantera Fonseca J.M., Lewis R., W3C - Delivery Context Ontology, http://www.w3.org/TR/dcontology/, 2009
[10] Caveney D., Cooperative Vehicular Safety Applications, IEEE CONTR SYST MAG, 30(4), 38-53, 2010
[11] Ceri S., Daniel F., Matera M., Model-Driven Development of context-aware web applications, ACM T INTERNET TECHN, 7(1), 2007
[12] Ceri S., Fraternali P., Bongio A., Brambilla M., Comai S., Matera M., Designing Data-Intensive Web Applications (Morgan Kaufmann Publishers, 2003)
[13] Chen H., An Intelligent Broker Architecture for Pervasive Context-Aware systems, PhD Thesis, University of Maryland, Baltimore County, 2004
[14] Corradi A., Fanelli M., Foschili L., Adaptive Context Data Distribution with Guaranteed Quality for Mobile Environments, Proceedings of the IEEE International Symposium on Wireless Pervasive Computing (5-7 May 2010 Modena Italy), IEEE Computer Society Press, 2010, 373-380
[15] Devaraju A., Hoh S., Hartley M., A context gathering framework for context-aware mobile solutions, In Chong P.H.J., Cheok A.D. (Eds.) Proceedings of the 4th international Conference on Mobile Technology, Applications, and Systems and the 1st international Symposium on Computer Human interaction in Mobile Technology (10-12 Sep. 2007 Singapore Singapore), ACM, 39-46, 2007
[16] Dey A., Abowd G., Towards a Better Understanding of Context and Context-Awareness. Technical Report 99-22, Georgia Institute of Technology, 1999
[17] Dey A., Abowd G., A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, HUMAN COMPU, 16(2), 97–166, 2001
[18] Dunlop M., Brewster S., The Challenge of Mobile Devices for Human Computer Interaction, PERS UBIQUIT COMPUT, 6(4), 235-236, 2002
[19] Esposito D., Programming Microsoft ASP.NET, Microsoft Press, ISBN: 0735643385, 2006
[20] Eugster P., Garbinato B., Holzer A., Pervaho: A specialized middleware for mobile context-aware applications, J ELECTRON COMM RES, Springer-Verlag, 9(4), 245-268, 2009
[21] Fahy P., Clarke S., CASS – a middleware for mobile context-aware applications, Proceedings of the Workshop on Context Awareness (6-9 Jun.2004 Boston USA), ACM, 2004, 304-308
[22] Ferguson P, Leman G., Perini P., Renner S., Seshagiri G., Software Process Improvement Works!, techical report CMU/SEI-99-TR-027, Software Eng. Inst., Carnegie Mellon Univ, 1999
[23] Fertalj K., Horvat M., Comparing architectures of mobile applications, Proceedings of the 5th WSEAS International Conference on Automation Information (15-17 Nov. 2004 Venice Italy), 2004, 946-952
[24] FIPA Gateways TC, FIPA Device Ontology Specification, http://www.fipa.org/specs/fipa00091/PC00091A.html, 2001
[25] Frank K., Rockl M., Nadales V., Robertson P., Pfeifer T., Comparison of exact static and dynamic Bayesian context inference methods for activity recognition, In Muhtadi J.A., Passarella A. (eds.). Proceedings of the 8th IEEE International Conference on Pervasive Computing and Communications Workshops (29 Mar.- 2 Apr. 2010 Mannheim

Germany), IEEE Computer Society, 2010, 189-195

[26] Gellersen H., Schmidt A., Beigl M., Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts, MOBILE NETW APPL, 7(5), 341 –351, 2002

[27] Gorgorin C., Gradinescu V., Diaconescu R., Cristea V., Iftode, L., Adaptive Traffic Lights using Car-to-Car Communication, In Heath R., Bottomley G. (eds.), Proceedings of the 2007 IEEE 66th Vehicular Technology Conference (30 Sep. – 3 Oct. 2007 Baltimore USA), IEEE Computer Society, 2007, 21-25

[28] Grimm R., One.world: Experiences with a pervasive computing architecture, IEEE PERVAS COMPUT., 3(3), 22-30, 2004

[29] Gu T., Hang X., Wang X., Pung H. K., Zhang D. Q., An Ontology-based Context Model in Intelligent Environments, In McDonald A. B. (ed.), Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference (18-21 Jan. 2004 San Diego USA), 2004, 270-275

[30] Gu T., Pung H. K., Zhang D. Q., A Service-Oriented Middleware for Building Context-Aware Services, J NETW COMPUT APPL, Elsevier, 28(1), 1-18, 2005

[31] Gupta A., Karla A., Boston D., Borcea C., MobiSoC: a middleware for mobile social computing applications, MOBILE NETW APPL, 14(1), 35-52, 2009

[32] Hemel Z., Visser E. Declaratively Programming the Mobile Web with Mobl. In Videira-Lopes C., Fisher K. (Eds.): Proceedings of the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2011, part of SPLASH 2011 (22-27 Oct. 2011 Portland, OR, USA). ACM 2011, 695-712

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

[34] Henricksen K., Indulska J., Rakotonirainy A., Modeling Context Information in Pervasive Computing Systems, In: Mattern F., Naghsineh M. (ed.), In Mattern F., Naghshineh, M. (eds), Proceedings 1st International Conference on Pervasive Computing (26-28 Aug. 2002 Zurich Switzerland), Springer Verlag, 2002, 167–180

[35] Hinckley K., Pierce J., Sinclair M., Horvitz E., Sensing techniques for mobile interaction, In Ackerman M., Edwards K. (eds). Proceedings of the 13th annual ACM symposium on User interface software and technology (06 – 08 Nov. 2000, San Diego, CA, USA), ACM New York, NY, USA 2000, 91-100

[36] Hofer T., Schwinger W., Pichler M., Leonhartsberger G., Altmann, J., Context-awareness on mobile devices – the hydrogen approach, In Sprague R.H. (ed), Proceedings of the 36th Annual Hawaii International Conference on System Sciences (6-9 Jan. 2003, Island of Hawaii), IEEE Computer Society, 2002, 292–302

[37] Hoffer J., Prescott M., McFadden F., Modern Database Management - 7th Edition, Prentice Hall, 7th edition, 2004

[38] Hoh S., Devaraju A., Wong C., A Context Aware Framework for User Centered Services, In Khong C.W., Wong C.Y., Niman B. (eds), Proceedings of the 21st International Symposium Human Factors in Telecommunication (27-28 Mar, 2008, Kuala Lumpur, Malaysia), Prentice Hall, 2008

[39] Honle N., Kappeler P., Nicklas D., Schwarz T., Grossmann M., Benefits of Integrating Meta Data into a Context Model, Proceedings 3rd IEEE International Conference on Pervasive Computing and Communications Workshops (8-12 Mar. 2005, Kauai Island, HI, USA), IEEE 2005, 25-29

[40] Iftode L., Smaldone S., Gerla M., Misener J., Active Highways, Proceedings of the IEEE Symposium on Personal, Indoor and Mobile Radio Communications (15-18 Sep. 2008, Cannes, French Riviera), IEEE 2008, 1-5

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

[42] Kappel G., Proll B., Retschitzegger W., Schwinger, W., Customisation for Ubiquitous Web Applications – A Comparison of Approaches. International Journal of Web Engineering and Technology, 1(1), 79-111, 2003

[43] Knappmeyer M., Baker N., Liaquats S., Tonjes R., A context provisioning framework to support pervasive and ubiquitous applications, In Barnaghi P., Moessner, K., Presser M., Meissner S. (eds.), Proceedings of the 4th European conference on Smart sensing and context (16-18 Sep., Guildford, UK,), Springer-Verlag Berlin, Heidelberg, 2009, 93-106

[44] Korpipää P., Mäntyjärvi J., Kela J., Keränen H., Malm E. J., Managing Context Information in Mobile Devices, PERVASIVE COMPUT., 2(3), 42-51, 2003

[45] Koukia S., Rigou M., Sirmakessis S., The Role of Context in m-Commerce and the Personalization Dimension, In Toyoaki N., Zhongzhi S., Ubbo V., Xindong W., Jiming L.,
Benjamin W., William C., Yiu-Ming C. (eds), Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology (18-22 Dec 2006, Hong Kong, China), IEEE 2006, 267-276

[46] Kranenburg H., Bargh M.S., Iacob S., Peddemors A., A context management framework for supporting context-aware distributed applications, IEEE COMMUN. MAG., 44(8), 67-74, 2006

[47] Krause M., SnHochstatter I., Challenges in modelling and using quality of context (qoc), Mobility Aware Technologies and Applications, Magedanz T. (ed), LNCS Springer Verlag 3744, 324–333, 2005

[48] Li F., Sehic S., Dustdar S., COPAL: An Adaptive Approach to Context Provisioning, In Chilamkurti N., Lian S., Mišić J., Taleb T. (eds), Proceedings 6th International Conference on Wireless and Mobile Computing, Networking and Communications (11-13 Oct, Niagara Falls, Ontario, Canada), IEEE, California, 2010

[49] Macedo F., Dos Santos L., Nogueira S., Pujole G., A Distributed Information Repository for Autonomic Context-Aware MANETs, IEEE TRANS NETW SERVICE MANAG, 6(1), 45-55, 2009

[50] March S. T., Smith G. F., Design and natural science research on information technology, Decision Support Systems, 15, 251-266, 1995

[51] Miles R., Hamilton K., Learning UML 2.0, O'Reilly Media, Inc, 2006

[52] MÃijhl G**.**, Fiege L**.**, Pietzuch P., Distributed Event-Based Systems, Springer, 1st edition, 2006

[53] Noble B., Satyanarayanan M., Narayanan D., Tilton J., Flinn J., Walker K., Agile application-aware adaptation for mobility, In Waite W. (ed), Proceedings of the sixteenth ACM Symposium on Operating Systems Principles (5-8 Oct 1997, Saint Malo, France), ACM New York, NY, USA 1997, 276-287

[54] Oracle, Java for mobile devices, http://www.oracle.com/technetwork/java/javame/javamobile/overview/getstarted/index.html, 2011

[55] Ortiz G., Garcia de Prado A., Improving Device-Aware Web Services and their Mobile Clients through an Aspect-Oriented, Model-Driven Approach, INFORM SOFTWARE TECH, 52(10), 1080-1093, 2010

[56] Quah J., Seet V., Adaptive WAP Portals, ELECTRON COMMER R A, 7(4), 337-385, 2007

[57] Pascoe J., The Stick-e Note Architecture: Extending the Interface Beyond the User, In Moore J., Edmonds E., Puerta, A. (eds.), Proceedings of International Conference on Intelligent User Interfaces (06 – 09 Jan 1997, Orlando, FL, USA), ACM New York, NY, USA 1997, 261-264

[58] Prechelt L., An Empirical Comparison of Seven Programming Languages, COMPUTER, 33(10), 23–29, 2000

[59] Rakotonirainy A., Loke S., Fitzpatrick G., Context-Awareness for the Mobile Environment, In Turner T., Szwillus G. (eds.), Proceedings of the Conference on Human Factors in Computing Systems (April 01 - 06, 2000, The Hague, Netherlands), ACM New York, NY, USA 2000

[60] Sheng Q. Z., Benatallah B., ContextUML: A UML-Based Modeling Language for Model-Driven Context-Aware Web Service Development, In Chang E., Brookes, W. (eds.), Proceedings of the 4th International Conference on Mobile Business (ICMB'05) (11 – 13 Jul 2005,,Sydney, Australia), IEEE Computer Society 2005

[61] Sheng Q. Z., Pohlenz S., Yu J., Wong H. S., Ngu A.H.H., Maamar Z., ContextServ: A platform for rapid and flexible development of context-aware Web services, In Atlee J.M., Inverardi, P. (eds.), Proceedings of the 31st International Conference on Software Engineering (16-24 May 2009, Vancouver, BC, Canada), IEEE Computer Society Washington, DC, USA 2009, 619-622

[62] Strang T., Linnhoff-Popien C., A Context Modeling Survey, In Davies N., Mynatt E., Siio I., (eds), Proceedings of the 1st International Workshop on Advanced Context Modelling, Reasoning and Management (7-10 Sep 2004, Nottingham, UK), Springer 2004

[63] Tanenbaum A.S., Modern operating systems (3rd edition), Pearson education international, Upper Saddle River, NJ. ISBN: 0-13-813459-6, 2009

[64] Tarasewich P., Designing mobile commerce applications, COMMUN ACM, 46(12), 57-60, 2003

[65] The APE project, Ajax Push Engine, http://www.ape-project.org/, 2011

[66] The Linux foundation, netem, http://www.linuxfoundation.org/collaborate/workgroups/networking/netem, 2011

[67] Thurlow R., RPC: Remote Procedure Call Protocol Specification Version 2, http://tools.ietf.org/html/rfc5531, 2009

[68] Truong H. L., Juszczyk L., Manzoor A. Dustdar, S., ESCAPE - An Adaptive Framework for Managing and Providing Context Information in Emergency Situations, In: Kortuem G., Finney J., Lea R., Sundramoorthy V. (Eds.), In: Proceedings of Smart Sensing and Context 2007 (EuroSSC 07) (23-25 Oct 2007, Lake District, UK), Springer Berlin/Heidelberg, 207-222, 2007

[69] Tsung open-source multi-protocol distributed load testing tool, http://tsung.erlang-projects.org/, 2011

[70] Venkatesh V., Ramesh V., Web and wireless site usability: Understanding differences and modeling use, MIS QUART, 30(1), 181-205, 2006

[71] Venkatesh V., Ramesh V., Massey A. P., Understanding usability in mobile commerce, COMMUN ACM, 46(12),

53-56, 2003
[72] Want R., Hopper A., Falcao V., Gibbons J., The Active Badge Location System, ACM T INFORM SYST, 10(1) 91-102, 1992
[73] Web Ontology Language, http://www.w3.org/2004/OWL/, 2011
[74] Ye J., Coyle L., Dobson S., Nixon P., Ontology – based models in pervasive computing systems, KNOWL ENG REV, 4, 315-347, 2007
[75] Zheng D., Wang J., Jia Y., Han W. H., Zou P., Middleware Based Context Management for the Component-Based Pervasive Computing, Lecture Notes in Computer Science, 4610/2007, 71-81, 2007