A MIDDLEWARE SERVICE FOR MANAGING TIME AND QUALITY DEPENDENT CONTEXT

Tasos Kontogiorgis
Unit of Medical Technology and Intelligent Information Systems
Dept. of Computer Science - University of Ioannina – P.O. BOX 1186, 45110 GR - Greece

Dimitrios Fotiadis
Unit of Medical Technology and Intelligent Information Systems
Dept. of Computer Science - University of Ioannina – P.O. BOX 1186, 45110 GR - Greece

Apostolos Zarras
Dept. of Computer Science - University of Ioannina – P.O. BOX 1186, 45110 GR - Greece
{tasos, fotiadis, zarras}@cs.uoi.gr

ABSTRACT

Nowadays, wearable devices, such as mobile phones, PDAs, etc. gain widespread popularity for communication and data exchange. Consequently, several approaches investigate the problem of their interconnection and communication, under a common middleware infrastructure enabling the development of mobile applications, which form a ubiquitous mobile computing environment. In such an environment, changes are very often and the applications need to be highly adaptive. In other words, the applications must be context-aware. The context of an application may be anything that influences its execution. In this work, we propose a middleware service, which enables reasoning about changes in the context of an application. It supports the adaptation of the services used and the application itself, according to context changes. The proposed service relies on a method for modeling context, which is based on temporal logic, which allows reasoning about time dependencies between context changes and adaptation actions. The reasoning procedure takes into quality properties (e.g. inaccuracy, unreliability, insecurity), characterizing the trustworthiness of the sources, which generate information about context changes.

KEYWORDS

Context-Aware Middleware, Trustworthiness, Ubiquitous Computing.

1. INTRODUCTION

The widespread use of wearable devices like mobile phones and PDAs dramatically changed the way of coping with the requirements of distributed applications. Such devises enable the realization of ubiquitous computing environments, consisting of mobile applications, which provide services to the users anywhere, anytime. Traditional middleware infrastructures like CORBA[1], J2EE[2] and DCOM[3] cannot deal with new *features* characterizing an ubiquitous computing environment and the devises that constitute it. Typical examples of such features are the location in space, battery-dependence, computational-power, memory, data-storage, communication-bandwidth, etc. The values of the aforementioned features constantly change. Consequently, the mobile applications and the middleware services used by them must adapt to those changes. These values constitute the *context* of the mobile applications. By definition [Dey, A., K., 2001], *context* is anything that influences the execution of a mobile application and the middleware services used.

Modeling and managing context information is a critical issue in the development of a mobile application. The representation of context information must be lightweight, flexible, and highly expressive. Moreover, it

[1] http://www.omg.org /technology/documents/formal/corba_iiop.htm
[2] http://java.sun.com/j2ee/
[3] http://msdn.microsoft.com/library/default.asp?url=/library/enus/cossdk/htm/ pgservices_events_5x4j.asp

must enable inductive and deductive reasoning, which results in triggering certain adaptation actions that customize the application and the middleware services. In order to model context while satisfying the previous requirements, in [Rangamathan, A. and Campbell, R. 2003] the authors have proposed the use of first-order predicate logic. In general, first-order predicate logic is a powerful tool for representing facts, events, actions, objects and relations between them. However, it is not expressive enough when dealing with *time dependencies* between the values of context features. This is vital especially in cases of mobile applications which handle critical situations (e.g. accidents, war situations, environmental catastrophes). Moreover, the values of context features cannot be considered always trustful. Consequently, context information must encompass a degree of fuzziness related to the values of the context features.

In this work we propose a middleware service, which enables managing and reasoning about changes in the context of mobile applications. It further enables adapting the applications and the middleware services used by them according to those changes. The proposed service is designed to be generic enough so that it can be incorporated within any specific middleware infrastructure like CORBA, DCOM, etc. It incarnates a context modeling and reasoning approach, which relies on temporal logic. Moreover, the proposed method introduces certain probabilistic features in order to take into account the trustworthiness of the values of context features.

The paper is structured as follows. Section 2 presents work related with the modeling and management of context in ubiquitous mobile computing environments. Section 3 presents the architecture of the proposed service. Section 4 analyses the proposed context modeling method. Section 5 discusses the management of context information and the reasoning procedure.

2. RELATED WORK

Several interesting approaches to context-aware computing have been elaborated so far. However, the issue of reasoning about context has not gained very much attention. More specifically, in [Dey, A. K., 2001] the author identifies requirements for supporting the development of context-aware applications and proposes a tool. The proposed tool does not provide any reasoning capabilities towards enabling the reaction of mobile applications into context changes. In addition, the proposed tool does not allow customizing the middleware services used by the applications according to context changes. In [Hong, J., et al., 2001] the authors go one-step further. Context modeling and management becomes a service that comes along with a specific middleware infrastructure. Similarly, in [Chan, A., T., S. and Chuang, S-N, 2003] a middleware infrastructure for context-aware applications is proposed. The infrastructure further provides services for the application adaptation and migration. The previous are triggered based on simple logical conditions that must hold for the values of certain context features. Time dependencies between those values are not taken into account. In [Rangamathan, A. and Campbell, R. 2003] the authors propose a method for reasoning and reacting to context changes. Their method is based on first-order predicate logic. Hence, it does not take into account time dependencies.

The Solar infrastructure [Chen, G. and Kotz, D., 2002] is mainly targeted to context management. According to this platform, the application may define a graph of operators (e.g., filters, transformers and more complicated aggregators) manipulating context. The operators mediate the flow of context information from the sources to the applications. A specification language is used to describe context features and operator graphs. In [Capra, L., et al., 2003] another interesting context-aware middleware infrastructure is proposed, which includes a micro-economic mechanism for the resolution of conflicts existing in the values of the context features characterizing a class of mobile applications. Such conflicts may prohibit the interoperation between them.

Much work on modeling context has been done in [Henricksen, K, et al., 2002]. They have explored the characteristics of context information in pervasive systems and describe a set of modeling concepts (e.g., static, dynamic and sensed associations, dependencies between associations, etc.) designed to accommodate these. In [Gray, P. and Salber, D. 2001] the authors also concentrate on representations for modeling context and identify quality properties characterizing the trustworthiness of the context itself (e.g. accuracy, timeliness, etc.). They further consider quality properties for the elements that produce the values of context (e.g. reliability, intrusiveness, etc). However, they do not provide any systematic approach for reasoning about the trustworthiness of the context based on the aforementioned quality properties.

In this paper, we built upon the previous approaches and we further contribute with a context modeling method and a reasoning procedure, which takes into account time dependencies between context features. Moreover, our reasoning procedure systematically considers quality properties characterizing the trustworthiness of context information. The aforementioned concepts are realized in a generic middleware service, which can be incorporated within existing middleware infrastructures having limited context-reasoning support.

3. ARCHITECTURE OF THE CONTEXT-AWARE SERVICE

Figure 1 gives the overall architecture of the proposed middleware service. The grey squares represent elements of the service, while the light-grey ones represent elements of an application. In general, we assume that the application conforms to the architectural style imposed by the RM-ODP standard for open distributed processing. As discussed in [Zarras, A., 2004], existing middleware infrastructures like CORBA, DCOM and J2EE follow this style. The architecture of the proposed service is generic enough to cope with applications built on top of any of these platforms.

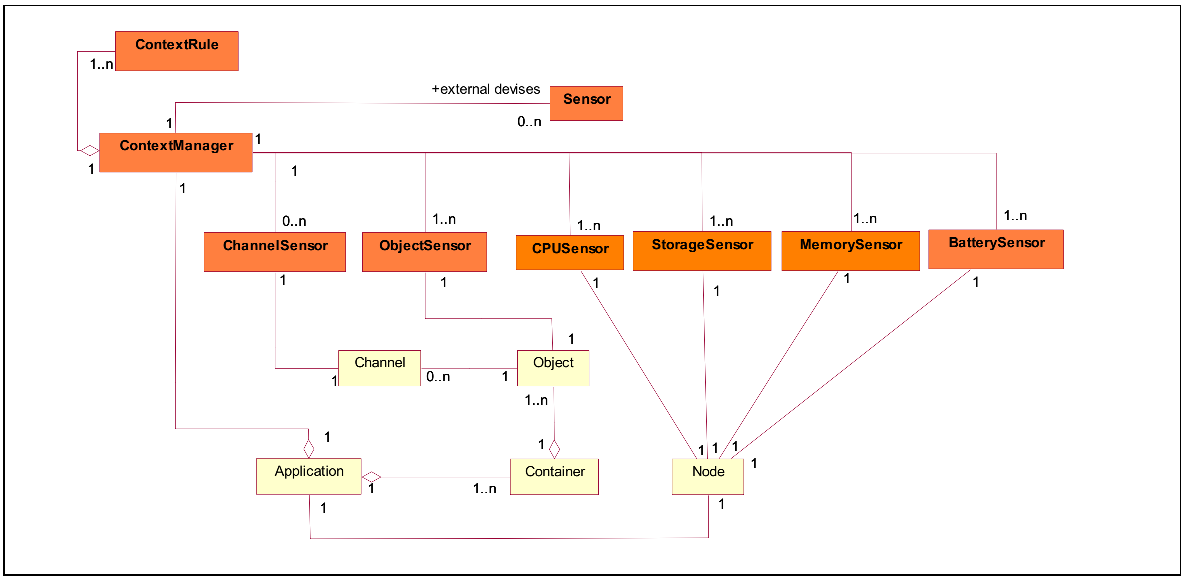

Figure 1. The overall architecture of the context-aware service

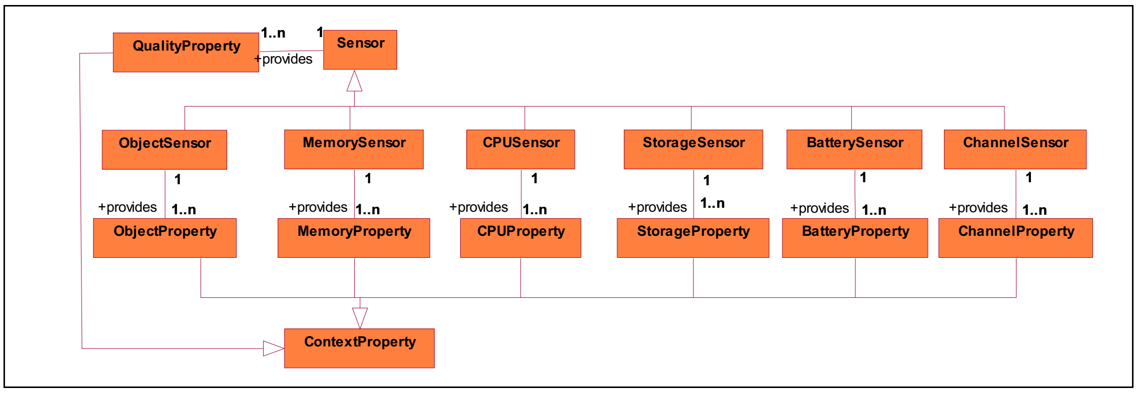

Figure 2. Classification of sensors and context features

The application consists of a set of *containers,* sharing the same processing and storage resources. A node on top of which the application executes provides those resources. A container comprises *objects* that form a single unit for the purpose of deactivation, reactivation, checkpoint, and recovery. Objects provide operations which can be used by others for assessing and modifying the objects' state. Objects belonging to different applications communicate through middleware *channels*. A channel further consists of proxy, skeleton, binder, and protocol objects. Moreover, we distinguish between different types of channels supporting point-to-point communication, multicast, and broadcast. The aforementioned elements are associated with different kinds of sensors shown in Figure 2. In particular, an object is associated with an *object sensor*, which provides information about the values of a number of context features characterizing the object. Table 1 gives a set of features we consider for objects. More specifically, an object may be persistent. Moreover, the object may actually represent a fault tolerant unit, i.e. it may consist of a number of replicated objects, which are coordinated according to three replication policies, namely: active, passive, and semi-active. The object can be transactional supporting the execution of either flat or nested transactions.

A node is associated with four different sensors. The *CPU sensor* provides information regarding context features like the CPU clock rate, and load (Table 1). Similarly, the *memory sensor* provides information regarding features like the total memory size and the memory size that is currently available for use. The *storage sensor* provides information related to the total size of stable storage provided by the node and the size that is currently available for use. Finally, the *battery sensor* provides information about maximum battery operation time and the remaining battery operation time. Moreover, it allows setting the operation of the node either in economy mode, or in normal mode.

A channel is associated with a *channel sensor*, which provides information about the delay, the data rate, the drop rate, and the security mode of the channel (Table 1). The latter may be set to high, low, or medium.

Table 1. Context features

Architectural Element	Context Property	Type	Produced By
Node	CPURate	Float	**CPUSensor**
	CPULoad	Float	
	MemTotal	Int	**MemorySensor**
	MemAvail	Int	
	StorTotal	Int	**StorageSensor**
	StorAvail	Int	
	BattTotalTime	Int	**BatterySensor**
	BattAvailTime	Int	
	Mode	enum{economy, normal}	
Channel	ChanDelay	Int	**ChannelSensor**
	ChanDropRate	Int	
	ChanSecurity	enum{high, low, med}	
Object	Persistence	enum{ON, OFF}	**ObjectSensor**
	Transactions	enum{FLAT, NESTED, OFF}	
	Replication	enum{OFF, PASS, ACT, SEMI}	
	NumOfRepl	Int	
Sensor	Unreliability	0..1	**Sensor**
	Inaccuracy	0..1	
	Insecurity	0..1	

Depending on the application, there may be also a number of external sensors, which are used, for instance, for tracking location, environmental temperature, etc. As shown in Figure 2 all sensors derive from a basic sensor element, which is associated with a number of context features (Table 1), characterizing the quality of the provided information. More specifically, the sensor is characterized by the *unreliability* property, i.e. *the probability that a sensor provides incorrect values due to accidental faults*. Moreover, the sensor is characterized by the *insecurity* property, i.e. *the probability that a sensor provides incorrect values due to intentional/malicious faults*. Finally, we consider the *accuracy* property, i.e. $\text{accuracy} = \left|(\text{sensor value} - \text{actual value})/\text{actual value}\right|$. As it is described in [Gray, P. and Salber, D. 2001] several other quality properties may be considered.

The context manager is the central unit of the proposed middleware service. The manager searches for changes in the context of the application and *adapts* the application and the middleware services it uses, according to those changes. To achieve the previous, the manager is associated with the set of application-related sensors. The changed values of context features are encapsulated into *events* sent by the sensors in the context manager. The way that the middleware and the application adapt in response to context changes is determined by a set of *context rules*, given as input to the context manager during the initialization of the application. *Roughly, a context rule specifies adaptation actions that are triggered by a number of context values, which where reported to the manager by temporally related events*. An action may comprise setting a new value of a context property (in which case a corresponding sensor generates another event, which may trigger a second context rule and so on), or calling a particular operation provided by the application objects.

An alternative design option for the context manager would be querying the sensors for information regarding changes in the context of the application. This approach, however, immediately implies delaying the actions that must take place towards adapting the middleware and the application according to the changes, until the time that the manager discovers them. The previous is certainly a drawback, especially in cases of critical applications that must adapt as fast as possible to the current environmental conditions.

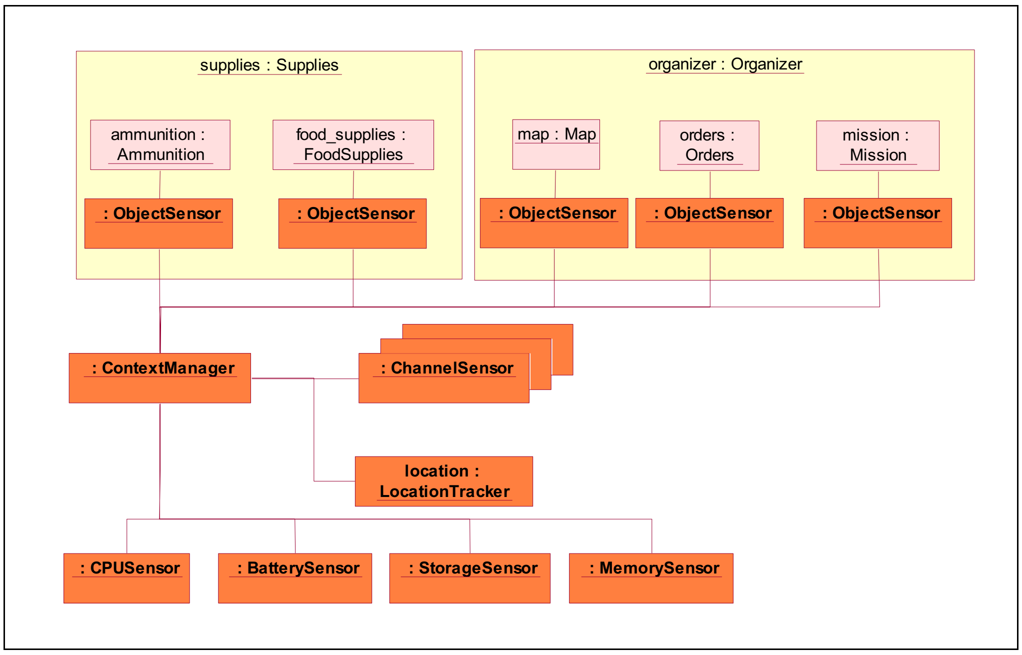

Figure 3. An application

Figure 3 gives an example of a mobile application relying on the proposed middleware service. The purpose of the application is to support the members of a military group that patrol across a hostile territory. Each soldier of the group has a mobile PC on top of which the application executes. The application comprises two main containers. The *supplies* container consists of two objects, namely *ammunition* and *food_supplies*, for managing the personal supplies of the soldier. The *organizer* container encapsulates three objects. The *map* object provides operations for accessing information about the route followed by the group and the position of the campus that hosts the group's members. The *orders* object contains guidelines from the group member which is the immediate superior of the soldier. Finally, the *mission* object provides operations for accessing information related to the main objectives of the group. All the aforementioned objects are *persistent* and *fault tolerant*. The objects, the portable PC and the channels used for communication[4] between the members of the group are associated with corresponding sensors. The overall application further uses an external *location* sensor, which reports to the context manager the current position of the soldier.

[4] Note that the communication channels are not given in the figure to avoid increasing the figure's complexity.

4. CONTEXT RULES SPECIFICATION

As we discussed in Section 1, we use temporal logic to model the context rules which serve as input to the context manager, enabling modeling time dependencies between changes in context features. As we further discuss in Section 5, using temporal logic facilitates managing the context information provided by the sensors to the context manager.

Table 2. Temporal operators for the specification of context rules

Operators		**Semantics**
logical operators	$\wedge, \vee, \neg, \Rightarrow$	Denote the logical *and*, *or*, *not* and *implication,* respectively.
Quantifiers	$\forall, \exists$	Denote the *universal* and *existential* quantifiers, respectively.
past operators	Θ	Denotes the *previous* operator. ΘP states that *P* held at the previous moment in time.
	∇	Denotes the *once* operator. ∇P states that *P* held at some time in the past.
	$\bullet$	Denotes the has *always been operator*. $\bullet$ P states that P held until this time.
future operators	$\oplus$	Denotes the *next* operator. $\oplus$ P states that *P* held at some time in the past.
	$\Diamond$	Denotes the *eventually* operator. $\Diamond$P states that *P* holds at some time in the future.
	$\circ$	Denotes the *henceforth* operator. $\circ$ P states that *P* holds from this time on.

A context rule is a temporal logic formula, which consists of *conditional parts* and *action parts*. A conditional part is defined using the values of certain context features, reported to the manager by temporally related events. The conditional part is related with an action part by a logical implication. The action part describes a number of temporally related actions that must take place when the conditional part holds. An action may result in changing the value of a particular context property or calling an operation on an application object. Both the conditional and the action parts consist of logical expressions, defined using the traditional temporal logic operators [Manna, Z. and Pnueli, A., 1992] denoted by the symbols given in Table 2[5].

$$R1 = \Diamond \text{node.BattAvailTime} \le 60 \Rightarrow \left(\forall c \in \text{supplies} \left| \begin{array}{l} \text{c.Replication} = \text{OFF} \wedge \\ \oplus \left(\begin{array}{l} \text{c.Persistence} = \text{OFF} \wedge \\ \oplus \left(\text{node.Mode} = \text{economy}\right) \end{array} \right) \end{array} \right. \right)$$

(a)

$$R2 = \left(\begin{array}{l} \text{node.Location} = \text{INCAMPUS} \wedge \\ \left(\begin{array}{l} \Diamond\left(\text{node.CPULoad} \ge 2\right) \vee \\ \Diamond\left(\text{node.MemAvail} \le 512\right) \end{array} \right) \end{array} \right) \Rightarrow \left(\forall \text{channel} \in \text{application} \middle| \text{channel.Security} = \text{low} \right)$$

(b)

Figure 4. Examples of context rules

Figure 4(a) gives an example of a context rule based on our example scenario. The conditional part of the rule holds if eventually the battery available time drops below 60 minutes and the corresponding actions comprise setting off the persistence and the replication properties of all the objects included in the supplies container. Moreover, the operation property of the node is set to the economy mode. That way less battery is spent. Note that in the action part of the rule, first we disable the persistence property. Next, we set off the replication property. Performing the previous actions in the reverse order may cause a fatal error in the execution of the application if the fault tolerance service of the middleware relies on the persistency service.

[5] Font limitations force us not to use the standard symbols for some of the operators (e.g. once, henceforth, etc.).

Thus, even in this simple example we can understand the necessity of using temporal operators in the specification of context rules.

Figure 4(b) gives another example of a context rule. The conditional part holds if the node is in the campus that hosts the military group and eventually either the CPU load increases or the available memory drops. In that case, the security mode of every channel of the application is set to low.

5. CONTEXT MANAGEMENT AND REASONING

The context manager of the proposed service consists of three separate parts (Figure 5). The RulesParser performs syntactic and type checks on the context rules given as input by the application. Based on the provided rules it constructs a history table, to store information regarding changes in the values of the context features specified in the rules. These values are encapsulated in the events generated by the sensors.

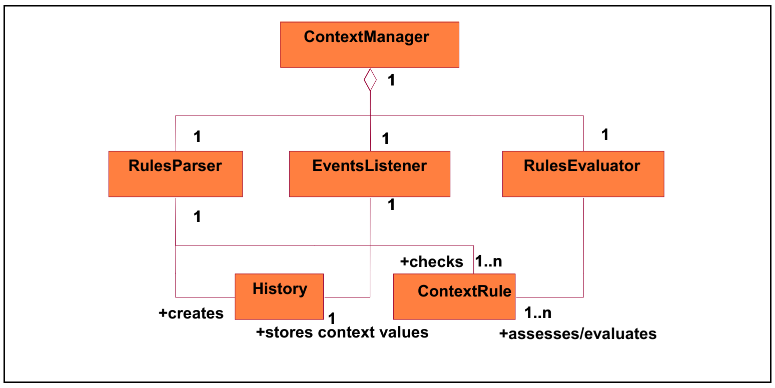

Figure 5. The main parts of the context manager

The EventListener part is responsible for the collection of the values. Moreover, it further determines which values need to be kept in the history table, depending on the context rules. For instance, a value related to a context property, involved in a logical expression, which contains a past operator, is kept in the history table. If the aforementioned operator is Θ, the value is kept only until the arrival of another event from any of the sensors. If, on the other hand, the operator is $\bullet$, the value is kept until the arrival of an event from the sensor that reported the value, reports a new value. Finally, if the operator is ∇, the value should be kept in the history for the lifetime of the application. However, the resources of mobile devices are limited and, the context manager periodically clears from the history values such the one above, based on a timeout set by the application.

The RulesEvaluator part evaluates the truth of the context rules based on the events' arrival. More specifically, for every event encapsulating a value, which refers to a particular context property, the RulesEvaluator looks for rules whose conditional part contains this property. If the conditional part of such a rule holds with respect to the reported value and the values of other context features which are possibly involved in it (the history table is used at this point), the action part of the rule is triggered.

However, the triggering of the action part is preceded by the assessment of the trustworthiness of the sensors which produced the values of the context features involved in this rule. More specifically, a value included in an event generated by a sensor is associated with three probabilities, corresponding to the inaccuracy, the unreliability, and the insecurity properties of the sensor (Table 1). Based on the probabilities of all the values involved in the rule, the RulesEvaluator calculates the total inaccuracy, unreliability and insecurity for this rule. The total values are compared against corresponding thresholds associated with the context rule. If none of the values is greater than a corresponding threshold, the action part of the rule is triggered. For the rule given in Figure 4(b), the total insecurity is calculated as follows:

$$\text{Insecurity}_{\text{R1}} = \text{Insecurity}_{\text{location} \wedge (\mathit{CPUSensor} \vee \mathit{MemSensor})} = \text{Insecurity}_{\text{location}} * \text{Insecurity}_{(\mathit{CPUSensor} \vee \mathit{MemSensor})} =$$

$$\text{Insecurity}_{\text{location}} * (\text{Insecurity}_{\mathit{CPUSensor}} + \text{Insecurity}_{\mathit{MemSensor}} - \text{Insecurity}_{\mathit{CPUSensor}} * \text{Insecurity}_{\mathit{MemSensor}}$$

6. CONCLUSIONS

We propose a middleware service for managing and reasoning about the constantly changing context of mobile applications. The proposed service facilitates the adaptation of the middleware services used by the application and the application itself, according to context changes. More specifically the main contributions of the proposed approach are summarized in the following:

- Context information is modeled using temporal logic. As demonstrated in Section 4, the proposed modeling method is far more expressive, compared to other previously proposed ones, which simply use first-order logic [Rangamathan, A. and Campbell, R. 2003]. It allows modeling time dependencies between changes in context information. It further enables specifying the order of the adaptation actions that must take place upon those changes. Moreover, Section 5 highlighted that the proposed modeling method allows the efficient management of context information, which is important considering the limited resources and processing capabilities of mobile devises.
- Reasoning about context changes and adaptation actions is performed while taking into account properties characterizing the quality of the sensors, which produce context information. Such properties were identified in previously proposed approaches [Gray, P. and Salber, D. 2001]. However, their impact was not incorporated in a systematic way within the reasoning process.

The aforementioned features of the proposed approach are crucial for context-aware applications aimed at handling critical situations (e.g. environmental crises, accidents, war situations, etc.). Currently, we further investigate such cases. Moreover, we examine the applicability of more advanced methods of inexact reasoning in the proposed approach (e.g. certainty factor models, necessity models, fuzzy logic, etc. [Panayiotopoulos, T. and Papakonstantinou, G., 1991]).

REFERENCES

Capra, L., et al., 2003. CARISMA: Context – Aware Reflective Middleware System for Mobile Applications. *In the IEEE Transactions on Software Engineering*, Vol. 29, No. 10, pp. 929-945.

Chan, A., T., S. and Chuang, S-N, 2003. MobiPADS: A Reflective Middleware for Context-Aware Mobile Computing. *In the IEEE Transactions on Software Engineering*, Vol. 29, No. 10, pp. 1072-1085.

Chen, G. and Kotz, D., 2002. Solar: An Open Platform for Context -Aware Mobile Applications. Modeling Context Information in Pervasive Computing Systems. *In Proceedings of the 1st International Conference on Pervasive Computing,* pp. 41-47.

Dey, A., K., 2001. A Understanding and Using Context. *In the Personal and Ubiquitous Computing Journal* , Vol. 5, No. 1, pp. 4-7.

Gray, P. and Salber, D. 2001. Modelling and Using Sensed Context Information in the Design of Interactive Applications. *In Proceedings of the 8th IFIP Conference on Engineering for Human Computer InteractionEHCI*, pages 317–335.

Henricksen, K, et al., 2002. Modelling Context Information in Pervasive Computing Systems. *In Proceedings of the 1st International Conference on Pervasive Computing*, pages 167–180.

Hong, J., I., et al., 2001. An Infrastructure Approach to Context-Aware Computing. *In the special issue on context aware computing of the Human Computer Interaction Journal*, Vol. 16, No. 2-4, pp. 287-303.

Manna, Z. and Pnueli, A., 1992. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag.

Panayiotopoulos, T. and Papakonstantinou, G., 1991. Predicate Logic and Inexact Reasoning. *Engineering Systems with Intelligence*. Kluwer Academic Publishers, pp. 65-71.

Rangamathan, A. and Campbell, R. 2003. An Infrastructure for Context-Awareness Based on First Order Logic. *In the Personal and Ubiquitous Computing Journal*, Vol. 7 , No. 6, pp. 353 – 364 .

Zarras, A., 2004. A Comparison Framework for Middleware Infrastructures. *In the Journal of Object Technology*, Vol. 3, No. 5, pp. 103-123.