Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. V (2010), No. 2, pp. 148-159

Middleware for Smart Environments Management

I. Anghel, T. Cioara, I. Salomie, M. Dinsoreanu, A. Rarau

Ionut Anghel, Tudor Cioara, Ioan Salomie, Mihaela Dinsoreanu, Anca Rarau
Technical University of Cluj-Napoca
Computer Science Department
15 Daicoviciu street, Cluj-Napoca, Romania
E-mail: {ionut.anghel,tudor.cioara,ioan.salomie,mihaela.dinsoreanu,anca.rarau}@cs.utcluj.ro

Abstract: This paper introduces a self-configuring middleware that manages the processes of context information acquisition and representation from smart closed environments, targeting the development of context aware applications. The environment context information is modeled using three sets: context resources, context actors and context policies. The context model artifacts are generated and administrated at run time by a management infrastructure based on intelligent software agents. The self-configuring property is enforced by monitoring the closed environment in order to detect variations or conditions for which the context model artifacts must be updated. The middleware was tested and validated within the premises of our Distributed Systems Research Laboratory smart environment.
Keywords: Self-Configuring, Closed Spaces, Context Awareness, Autonomic Computing, Pervasive Systems

1 Introduction and Motivation

The context aware applications continuously monitor, capture and interpret environment related information in order to adapt their behavior to changes. A context aware system can evolve in two distinct types of environments: open environments and closed environments. An open environment is a large, heterogeneous space with no fixed spatial limitations (e.g. a city or an airport). In an open space the most relevant context information is the location which can be determined through GPS devices. On the other hand, a closed environment is a smart space delimited by predefined fixed boundaries featuring a large number of sensors and intelligent devices (e.g. a smart hospital) where the actor's location is identified by indoor specific techniques such as RFID (Radio-Frequency Identification) [1] or location sensors. These spaces offer a large variety of context information which is usually captured through sensor networks.

Let's consider a scenario in which a context aware application is used to guide the tourists into a museum. The museum is an intelligent closed space in which visitors are identified by RFID readers, while their location and orientation is determined using a sensor network. The tourists, as scenario actors, can use the context aware application if they have wireless capable PDAs on which an application can be downloaded and executed. In the museum, the visitors must follow a set of rules such as the minimum distance to the artifacts, loud limits, etc. The interaction between the tourists and the museum resources is made through requests from the tourists to the application residing on their PDAs. These requests may vary in complexity from simple requests such as information about a specific artifact, to complex requests such as requesting a tour to visit the middle age artifacts. Using this scenario, the following environment resources which provide relevant information and determine the context aware application execution are identified: the resources attached to the tourist such as PDA or RFID tags; the intelligent museum resources such as location sensors, orientation sensors or actuators; the constraints used to drive the tourist-museum interaction such as the minimum distance to the artifacts; the tourist preferences and his mobility.

The resources variety may influence the execution of a context aware application making the context information acquisition and representation processes extremely difficult to manage. During the context information acquisition process, the context information sources (e.g. sensors) can fail or new context information sources may be identified. The context acquisition and representation processes need to be reliable and fault tolerant. A context aware application cannot wait indefinitely for an answer from a temporary unavailable context resource. On the other hand, the payoff for not taking into consideration the new available context resources can be very high.

From our perspective it is necessary to introduce some degree of autonomy to the context acquisition and representation processes in order to achieve an efficient closed space context information management. We address the problems of monitoring and capturing context information by proposing and developing a pervasive self-configuring middleware that manages intelligent closed space environments. The fundamental element of this middleware is the context model which represents the environment context information using three sets: context resources, context actors and context policies. The context model management infrastructure is implemented by intelligent software agents [2] that generate and administrate the context model artifacts at run time. The middleware self-configuring feature is implemented by monitoring and evaluating the environment changes in order to keep the context artifacts updated. The proposed middleware is tested and validated in our Distributed Systems Research Laboratory [3] smart environment infrastructure.

The rest of the paper is organized as follows: in Section 2, a domain research overview is presented; Section 3 presents the middleware conceptual architecture; Section 4 presents the middleware self-configuring feature and a self-configuring algorithm; Section 5 shows how the middleware is used to manage the context representation of a smart laboratory while Section 6 concludes the paper and shows the future work.

2 Related Work

During the middleware development process we have identified the context representation and the autonomic context management as major problems. The rest of this section presents the state of the art related to these research directions by highlighting our approach advantages.

For *context representation*, generic models that aim at accurately describing the system execution context in a programmatic way are proposed [4]. Key-Value models represent context information using a set of attributes and their associated values [5]. Markup models enable structuring context information into a hierarchy. Tags describe context attributes and associated values [6]. Object models structure context into object classes and their implicit relationships [7]. In [8], the concept of multi-faceted entity is defined and used to model the set of context properties. A facet represents the effective values of the context properties to which the context aware application has access. The main drawback of the above presented approaches is the lack of semantic information encapsulated in the context model representation which makes the process of inferring new context related knowledge extremely difficult. In the current work we try to overcome these deficiencies by using our RAP context model [9] to represent the context information. The RAP context model ontology based representation is used by the context aware applications to infer new context related information using reasoning and learning algorithms. The use of ontology representations to model the context related information is also proposed in [10], [11]. In this approaches the context properties are represented as ontological concepts during design time and instantiated with run-time sensor captured values. The main disadvantage of these approaches is the high degree of inflexibility determined by the human intervention in the context representation phase. This problem is solved in our approach by defining an agent based solution that uses the RAP context model set based context representation to evaluate the real world context and to automatically construct the context representation.

In the *context management direction*, the research is concentrated on developing techniques for (i) keeping the context representation consistent with the environment and for (ii) automatic discovery and setup of new context resources. In [12], models for capturing and updating the context information based on the information type are proposed. In this approach the context information is classified in three types: user information, physical information and computational information. Another approach is to define reusable components for updating the context specific data [4]. These components provide stable communication channels for capturing and controlling context specific data. In [13] the authors propose the development of context guided behavioral models, which allow context aware applications to detect only those context data variations that lead to the modification of their behavior. The main disadvantage of these approaches is the lack of efficiency in the context model management process which is rather static and difficult to adapt to context changes. The current approach introduces a degree of autonomy for the context acquisition and representation processes in order to provide an efficient context information management. There is a reduced number of researches regarding the context aware systems self-* management in the literature and they are focused on the self-adaptation problem. Models and algorithms that allow computational systems to execute specific actions according to the context or situation at hand are proposed [14]. Their objective is to associate a certain degree of intelligence to the computational systems for context adaptation. In [15], the authors propose a context adaptive platform based on the closed loop control principle. The novelty of this proposal consists in defining and using the concept of application-context description to represent system knowledge about the context. This description is frequently updated and used for system control allowing it to reconfigure and take adapting decisions. [16], [17] and [18] propose context adaptation models for web services and web service orchestrations based on defining the service behavior in a certain situation using a set of context adapting rules, each rule consisting of a context condition and an associated action. A self-adapting context model that uses a system situation space to represent the system execution context is proposed in [19]. Using learning algorithms, the system may infer the action to be executed when a new situation appears by placing it in a situation space group. The use of the self-configuring autonomic computing paradigm for the development and integration of self-managing components into context sensitive systems is a new research direction. The research efforts concentrate on developing models for managing the automatic discovery, installation and configuration of complex context aware systems and their components [20], [21]. In this paper we make a step forward towards developing context aware self-* management systems by defining a self-configuring algorithm that is used to develop a context aware management middleware for smart environments.

3 The Middleware Conceptual Architecture

The middleware architecture (see Figure 1) defines three main layers: the *acquisition layer* that captures the context information from closed environments, the *context model layer* which represents the context information in a machine interpretable way and the *context model management layer*. In the following we detail each of the three middleware architectural layers.

3.1 The Context Acquisition Layer

The middleware monitors and captures information form the closed environment in order to accurately represent the context. Our approach to the context information acquisition layer (see Figure 2) provides (i) a sensor information acquisition mechanism and (ii) easy access and good visibility of the sensor information to middleware upper layers.

From the middleware perspective we have defined both push and pull types of sensor information acquisition mechanisms. The push mechanism uses event based listeners to determine when information is available in order to make it visible to the middleware upper layers. The pull mechanism is query

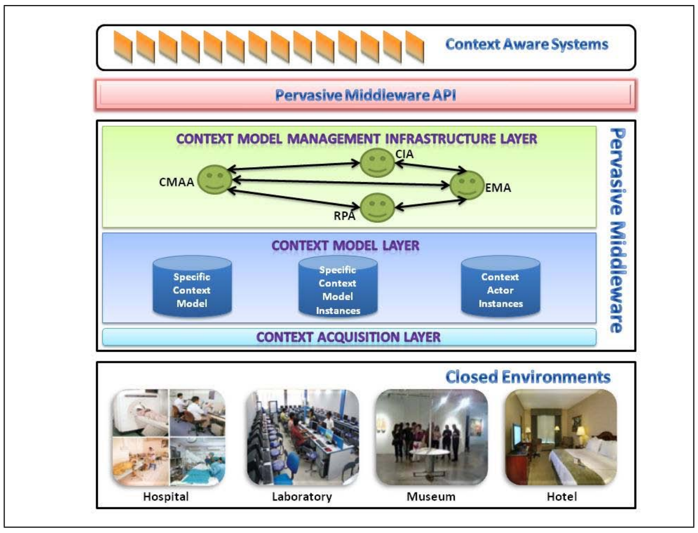


Figure 1: The Pervasive Middleware Conceptual Architecture

based, allowing the sensor information to be provided on demand. The sensor information is made visible to the upper middleware layers by the exposing associated web services.

3.2 The Context Model Layer

To represent a closed environment context information in a programmatic manner we have used the RAP context model [9]. In RAP, the context is defined as a triple: C = $<$R, A, P $>$, where R is the set of context resources that generates and/or processes context information, A is the set of actors which interact with context resources in order to satisfy their needs and P is the set of real world context related policies. The set of context resources R is split in two disjunctive subsets: (i) the set of context resources attached to the real world context environment R_S and (ii) the set of context resources attached to the actors R_A.

In order to provide an accurate representation of a closed environment context, the following representation artifacts are defined (see Figure 3): *specific context model*, *specific context model instance* and *context - actor instance*. The specific context model $C_S = <R_S, A_S, P_S>$ is obtained by mapping the context model onto different closed environments and populating the sets with environment specific elements. A specific context model instance $C_{SI} = <R_{SI}, A_{SI}, P_{SI}>$ contains the set of context resources with which the middleware interacts, together with their values in a specific moment of time t. The context - actor instance $CI_a{}^t = < R_a{}^t, a, P^t >$ contains the set of context resources with which the actor can interact, together with their values in a specific moment of time t.

3.3 The Context Model Management Layer

The context model management infrastructure layer is based on four types of intelligent, cooperative intelligent software agents: *Context Model Administering Agents*, *Context Interpreting Agents*, *Request Processing Agents* and *Execution and Monitoring Agents*. The Context Model Administering Agent (CMAA) is the specific context model manager. Its main goal is the synchronization of the context model artifacts with the system execution environment. It is also responsible for negotiation processes that take place when an actor or resource is joining the context. The Context Interpreting Agent (CIA) semantically evaluates the information of a context instance and tries to find the context instance meaning. The Request Processing Agent (RPA) processes the actor requests. RPA identifies and generates the action plans that must be executed for serving an incoming request. Also, RPA uses the context

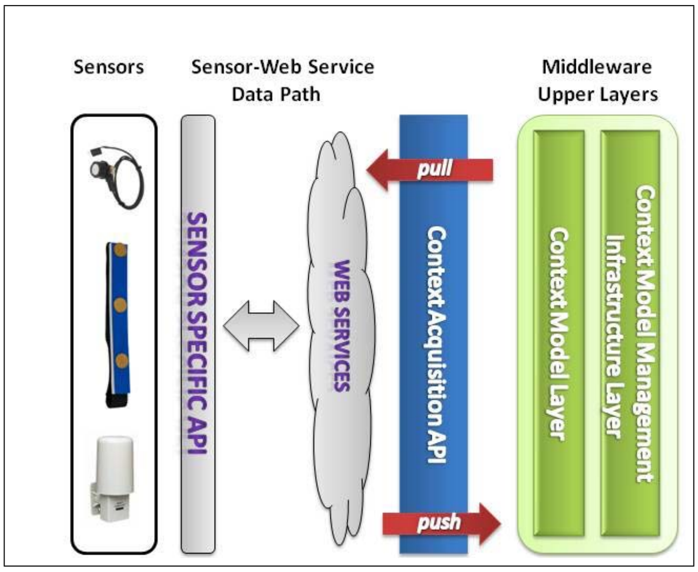


Figure 2: The context acquisition layer

model artifacts to identify the proper plan to be executed by the Execution and Monitoring Agent or for generating a new plan. The Execution and Monitoring Agent (EMA) processes the plans received from RPA agent and executes every plan action using the available services. After mapping action plans onto services, a plan orchestration which can be executed using transactional principles is obtained.

4 The Self-Configuring Feature

In order to provide an efficient, reliable and fault tolerant closed space context information management, the middleware is enhanced with the autonomic computing self-configuring property. This property is enforced by: (i) monitoring the closed environment in order to detect context variations or conditions for which the context artifacts must be updated (section 4.1.) and (ii) executing a self-configuring algorithm (section 4.2.) that updates/populates the context artifacts.

4.1 The Closed Environment Context Variation

We have identified three causes that generate context variation: (1) adding or removing context elements (resources, actors, policies) to/from the closed environment, (2) the actors' mobility within the environment and (3) the changes of the resources' property values (mainly due to changing the sensors' captured values). In the following, we discuss each of these context variation causes to evaluate the context variation degree which determines the condition of executing the self-configuring process.

Adding or removing context elements. During the context information acquisition process, the sources of context information can fail or randomly leave/join the context. These changes generate a context variation that is detected by the context acquisition layer and sent to CMAA which creates context artifacts adapted to the environment. Next, we evaluate the context variation degree generated by context resources ΔR, context actors ΔA and context policies ΔP in relationship with a set of associated thresholds T_R, T_A and T_P respectively. The context resources set variation is generated by adding or removing a context resource *r* (sensor or actuator) to/from the closed environment:

$$\Delta R = \{ R^{t+1} \setminus R^{t} \} \cup \{R^{t} \setminus R^{t+1}\} \qquad (1)$$

In relation (1) $R^{t+1} \setminus R^{t}$ contains the set of context resources that become available and $R^{t} \setminus R^{t+1}$ contains

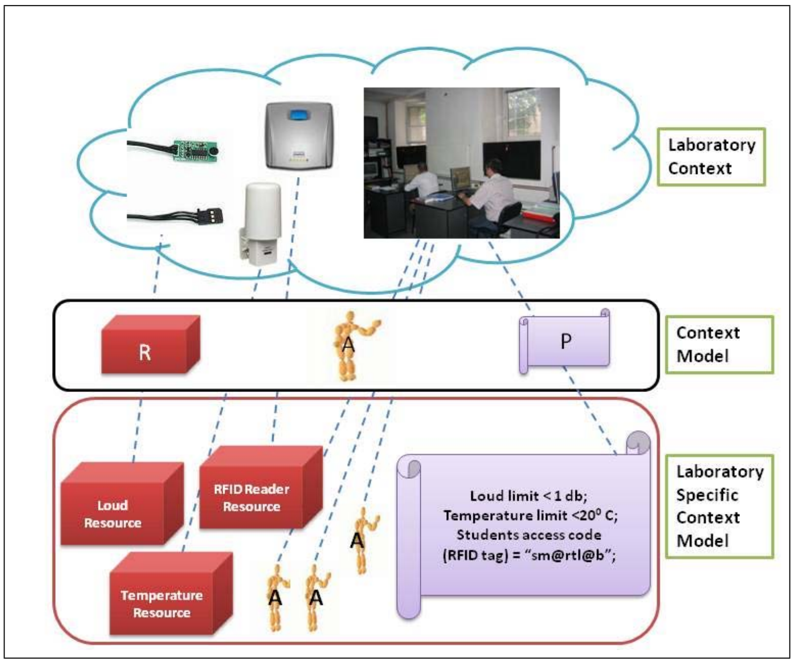


Figure 3: The RAP context model

the set of context resources that become unavailable. If Card(ΔR) $\geq$ T$_R$ a new specific context model is generated by adding or removing the context resources contained in ΔR. Using the same assumptions and conclusions as for context resources, we compute the policy set variation:

$$\Delta \mathrm{P} = \{ \mathrm{P}^{t+1} \setminus \mathrm{P}^{t} \} \mathrm{U} \{ \mathrm{P}^{t} \setminus \mathrm{P}^{t+1} \} \tag{2}$$

For evaluating the actors related context, the fact that each context actor has a set of attached resources was taken into consideration:

$$\Delta \mathrm{A} = \{ \mathrm{A}^{t+1} \setminus \mathrm{A}^{t} \} \mathrm{U} \{ \mathrm{A}^{t} \setminus \mathrm{A}^{t+1} \} \mathrm{U} \{ \mathrm{R}_A{}^{t+1} \setminus \mathrm{R}_A{}^{t} \} \mathrm{U} \{ \mathrm{R}_A{}^{t} \setminus \mathrm{R}_A{}^{t+1} \} \tag{3}$$

The overall real world context variation ΔENV is given by the union of all context elements' variation:

$$\Delta \mathrm{ENV} = \Delta \mathrm{R} \ \mathrm{U} \ \Delta \mathrm{A} \ \mathrm{U} \ \Delta \mathrm{P} \tag{4}$$

CMMA should start the execution of the self-configuring process and should generate a new specific context model when Card(ΔENV) $\geq$ T$_{Self-Configuring}$, where the self-configuring threshold is defined as:

$$\mathrm{T}_{Self-Configuring} = \min(\mathrm{T}_R, \mathrm{T}_A, \mathrm{T}_P) \tag{5}$$

The actor's mobility. Due to their mobility, the actors are changing their environment location and implicitly the set of resources with which they interact. CMAA identifies this variation and generates (i) a new context-actor instance and (ii) a new specific context model instance. In order to evaluate the context variation generated by actors' mobility we use the isotropic context space concept, defined in [9]. A context space is isotropic if and only if the set of real world context resources is invariant to the actors' movement. Usually, a context space is non-isotropic, but it can be split into a set of disjunctive isotropic context sub-space volumes in which the isotropy degree variation is the empty set. Such volume is called context granule. For a given moment of time, an actor can be located only into a single context granule. As a result, the space isotropy variation ΔIZ is non-zero only when an actor a moves between two context granules. The isotropy variation for a context actor is computed as:

$$\Delta \mathrm{IZ}_a = \{ \mathrm{R}_{GC}{}^{t+1} \setminus \mathrm{R}_{GC}{}^{t} \} \cup \{\mathrm{R}_{GC}{}^{t} \setminus \mathrm{R}_{GC}{}^{t+1}\} \quad (6)$$

CMMA continuously monitors the actors' movement in the environment and periodically evaluates the space isotropy variation. If for an actor, the space isotropy variation is a non empty set, $\Delta \mathrm{IZ}_a \neq \emptyset$, then the self-configuring process executed by CMMA generates a new context-actor instance:

$$\mathrm{CI}_a{}^t = \ < \mathrm{R}_a{}^t, \mathrm{a}, \mathrm{P}^t >, \mathrm{R}_a{}^{t+1} = \mathrm{R}_{GC}{}^{t+1} \quad (7)$$

Changes of resources property values. A context resource is a physical or virtual entity which generates and/or processes context information. The resource properties, K(r), specify the set of relevant context information that a resource can provide. For example, the set of context properties for a HotandHumidity sensor is: K(HotandHumidity) = {Temperature, Humidity}. In order to evaluate the context variation generated by the changes of the resource property values, we define a function K_{val} that associates the resource property to its value:

$$\mathrm{K}_{val}(\mathrm{R}) = \{(\mathrm{k}_1,\mathrm{val}_1), \ldots, (\mathrm{k}_n,\mathrm{val}_n)\} \text{ with } \mathrm{k}_1, \ldots, \mathrm{k}_n \in \mathrm{K} \quad (8)$$

If the values captured by the HotandHumidity sensor in a moment of time are 5 degree Celsius for temperature and 60% for humidity, then: K_{val}(HotandHumiditySensor) = {(Temperature, 5), (Humidity, 60%)}. The CMAA agent calculates the context variation generated by changes of resource properties' values ΔRPV as presented in (9). If val^{t+1}- val^{t}=0 then the property value hasn't changed from t to t+1 and that property is ignored when the variation is calculated. As a result, we conclude that a new specific context model instance should be created when Card(ΔRPV) ≥ 0.

$$\Delta \mathrm{RPV} = \mathrm{K}_{val}(\mathrm{R}^{t+1}) - \mathrm{K}_{val}(\mathrm{R}^{t}) = \{(\mathrm{k}_1,\mathrm{val}_1{}^{t+1} - \mathrm{val}_1{}^{t}), \ldots, (\mathrm{k}_n,\mathrm{val}_n{}^{t+1} - \mathrm{val}_n{}^{t})\} \quad (9)$$

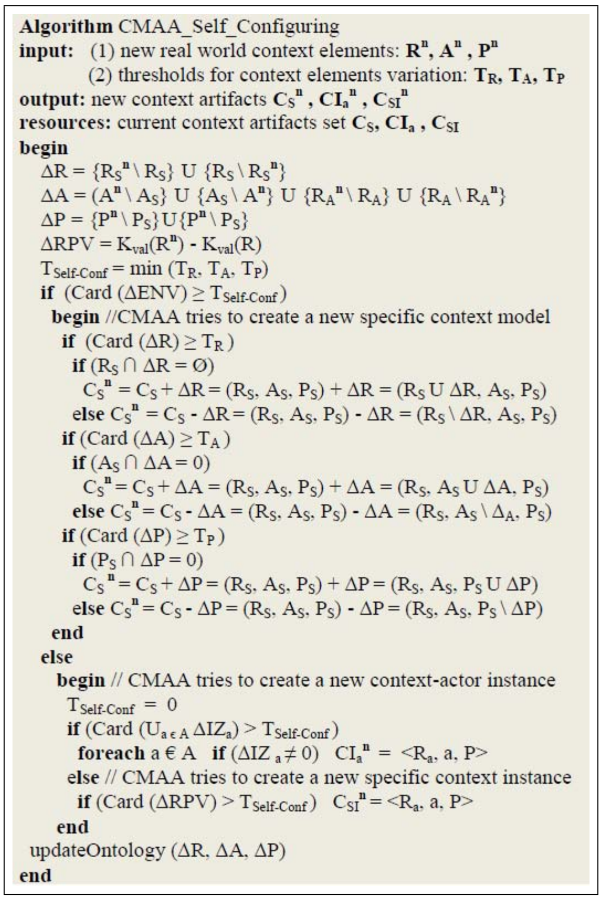
4.2 The Self-Configuring Algorithm

The self-configuring algorithm is executed by CMAA in order to keep the context model artifacts synchronized with the real context (see Figure 4). CMAA features a ticker based behavior by periodically evaluating the context changes. When a significant context variation is determined, the context model artifacts are updated.

5 Case Study

For the case study we have used the intelligent closed environment represented by our Distributed System Research Laboratory. In the laboratory students are marked using RFID tags and identified using a RFID reader. The students interact with the smart laboratory by means of wireless capable PDAs on which different laboratory provided services are executed (submit homework service, print services, information retrieval services, etc.). A sensor network captures information regarding students' location or orientation and also information regarding the ambient like the temperature or humidity. The DSRL infrastructure contains a set of sensors through which the real context information is collected: two HotandHumidity sensors that capture the air humidity and the temperature, four Orientation sensors placed in the four corners of the laboratory that measure the orientation on a single axis, one Loud sensor that detects sound loudness level and one Far Reach sensor that measures distances (see Figure 5).

The sensors are connected using a Wi-microSystem wireless network produced by Infusion Systems Ltd [22]. The middleware is deployed on an IBM Blade-based technology Server Center which maintenance software offers autonomic features like self-configuring of its hardware resources. The context related data captured by sensors is collected through the Wi-microSystem that has an I-CubeX WimicroDig analog to digital encoder as its main part. It is a configurable hardware device that encodes up to



```
Algorithm CMAA_Self_Configuring
input:  (1) new real world context elements: $R^n$, $A^n$, $P^n$
        (2) thresholds for context elements variation: $T_R$, $T_A$, $T_P$
output: new context artifacts $C_S^n$, $CI_a^n$, $C_{SI}^n$
resources: current context artifacts set $C_S$, $CI_a$, $C_{SI}$
begin
  $\Delta R = \{R_S^n \setminus R_S\} \cup \{R_S \setminus R_S^n\}$
  $\Delta A = (A^n \setminus A_S\} \cup \{A_S \setminus A^n\} \cup \{R_A^n \setminus R_A\} \cup \{R_A \setminus R_A^n\}$
  $\Delta P = \{P^n \setminus P_S\} \cup \{P^n \setminus P_S\}$
  $\Delta RPV = K_{val}(R^n) - K_{val}(R)$
  $T_{Self\text{-}Conf} = \min(T_R, T_A, T_P)$
  if (Card ($\Delta ENV$) $\geq T_{Self\text{-}Conf}$)
   begin //CMAA tries to create a new specific context model
    if (Card ($\Delta R$) $\geq T_R$)
     if ($R_S \cap \Delta R = \emptyset$)
      $C_S^n = C_S + \Delta R = (R_S, A_S, P_S) + \Delta R = (R_S \cup \Delta R, A_S, P_S)$
     else $C_S^n = C_S - \Delta R = (R_S, A_S, P_S) - \Delta R = (R_S \setminus \Delta R, A_S, P_S)$
    if (Card ($\Delta A$) $\geq T_A$)
     if ($A_S \cap \Delta A = 0$)
      $C_S^n = C_S + \Delta A = (R_S, A_S, P_S) + \Delta A = (R_S, A_S \cup \Delta A, P_S)$
     else $C_S^n = C_S - \Delta A = (R_S, A_S, P_S) - \Delta A = (R_S, A_S \setminus \Delta_A, P_S)$
    if (Card ($\Delta P$) $\geq T_P$)
     if ($P_S \cap \Delta P = 0$)
      $C_S^n = C_S + \Delta P = (R_S, A_S, P_S) + \Delta P = (R_S, A_S, P_S \cup \Delta P)$
     else $C_S^n = C_S - \Delta P = (R_S, A_S, P_S) - \Delta P = (R_S, A_S, P_S \setminus \Delta P)$
   end
  else
    begin // CMAA tries to create a new context-actor instance
     $T_{Self\text{-}Conf} = 0$
     if (Card ($\bigcup_{a \in A} \Delta IZ_a$) > $T_{Self\text{-}Conf}$)
      foreach $a \in A$  if ($\Delta IZ_a \neq 0$)  $CI_a^n = <R_a, a, P>$
     else // CMAA tries to create a new specific context instance
      if (Card ($\Delta RPV$) > $T_{Self\text{-}Conf}$)  $C_{SI}^n = <R_a, a, P>$
    end
 updateOntology ($\Delta R$, $\Delta A$, $\Delta P$)
end
```

Figure 4: The self-configuring algorithm

8 analog sensor signals to MIDI messages which are real-time wirelessly transmitted, through Bluetooth waves, to the Server Center for analysis and/or control purposes. The Bluetooth receiver located on the Blade computer is mapped as a Virtual Serial Port.

CMAA periodically evaluates the context information changes at a predefined time interval (we use 1 second time intervals for this purpose). If significant variations are detected, the context model artifacts are created or updated using the self-configuring algorithm presented in section 4.2. When the middleware is deployed and starts its execution (t=0), there are no context model artifacts constructed meaning that the R, A and P sets of the context model are empty. After one second (t=1), when two students John and Mary enter the lab, CMAA receives the updated context information from the Context Acquisition Layer and calculates the context elements variation ΔR, ΔA and ΔP as presented in Figure 6. By default the self-configuring thresholds are set to the value 1 ($T_{Self-Configuring}$=T_R=T_A=T_P=1). As a result of evaluating the context variation at t=1, CMAA executes the self-configuring algorithm which adds new concepts/populates the context model artifacts. The new added concepts originate from the context elements set variations ΔR, ΔA and ΔP calculated in Figure 6.

In order to assess the performance of the proposed self-configuring algorithm a simulation editor was developed. The evaluation test cases can be described by adding simulation times together with the corresponding sensor values. We evaluated the memory and processor overloading when executing the self-configuring algorithm in order to update the specific context model instance due to sensor values changes. Using the simulator, we tested our middleware with 100 sensors that change their values at 100 ms and 2000 ms. Even if the sensor values change rate is much higher at 2000 ms, the memory and processor overloading did not show major differences (see Figure 7).

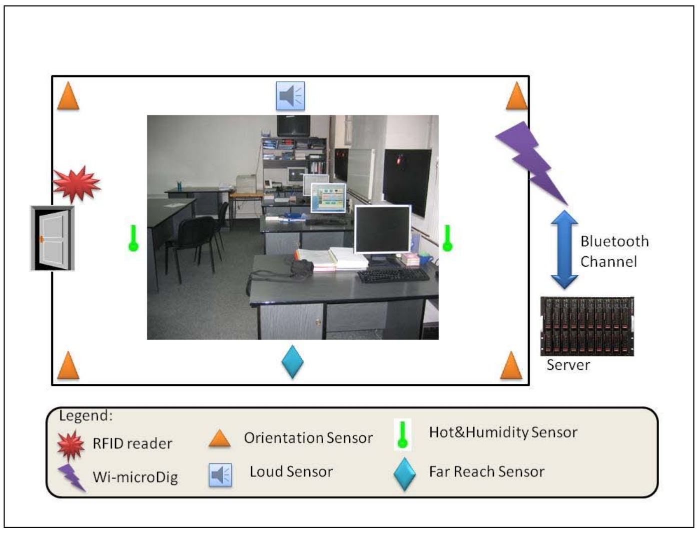
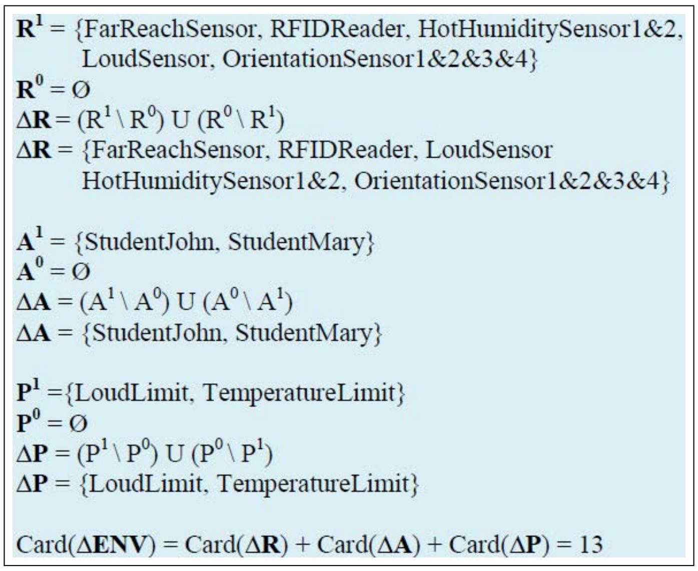


Figure 5: The DSRL infrastructure



$\mathbf{R}^1$ = {FarReachSensor, RFIDReader, HotHumiditySensor1&2, LoudSensor, OrientationSensor1&2&3&4}

$\mathbf{R}^0 = \emptyset$

$\Delta\mathbf{R} = (R^1 \setminus R^0)\ U\ (R^0 \setminus R^1)$

$\Delta\mathbf{R}$ = {FarReachSensor, RFIDReader, LoudSensor HotHumiditySensor1&2, OrientationSensor1&2&3&4}

$\mathbf{A}^1$ = {StudentJohn, StudentMary}

$\mathbf{A}^0 = \emptyset$

$\Delta\mathbf{A} = (A^1 \setminus A^0)\ U\ (A^0 \setminus A^1)$

$\Delta\mathbf{A}$ = {StudentJohn, StudentMary}

$\mathbf{P}^1$ ={LoudLimit, TemperatureLimit}

$\mathbf{P}^0 = \emptyset$

$\Delta\mathbf{P} = (P^1 \setminus P^0)\ U\ (P^0 \setminus P^1)$

$\Delta\mathbf{P}$ = {LoudLimit, TemperatureLimit}

Card($\Delta\mathbf{ENV}$) = Card($\Delta\mathbf{R}$) + Card($\Delta\mathbf{A}$) + Card($\Delta\mathbf{P}$) = 13

Figure 6: DSRL context variation at t=1

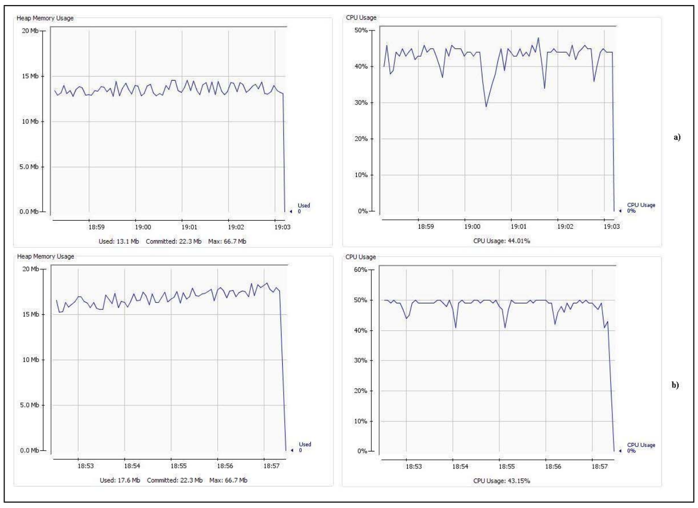


Figure 7: The self-configuring algorithm CPU and memory overloading with 100 sensors at a) t1=100 ms and b) t2=2000 ms

For testing our self-configuring algorithm scalability we have implemented an application that can simulate the behavior of a large number of sensors that randomly generate context information at fixed periods of time. The results show that the self-configuring algorithm implemented by CMAA can generate, synchronize and update the context model artifacts in a reasonable time for up to 20 sensors that change their values simultaneously. It is possible that sensor values change much faster than CMAA can synchronize contexts when the processing time is higher than the CMAA ticker interval.

6 Conclusions

This paper addresses the problem of managing the closed environment context information acquisition and representation processes in a reliable and fault tolerant manner. In order to achieve our goal we have defined a self-configuring middleware that uses a context management infrastructure based on agents, to gather context information from sensors and generate a context representation at run-time. The self-configuring property is enforced at the middleware level by monitoring the execution context in order to detect context variations for which the context artifacts must be updated. As a result, the middleware supports the dynamic configuration of the context artifacts. For future development, we intend to enhance the middleware with self-healing and self-optimizing autonomic features.

Bibliography

[1] G. Roussos and V. Kostakos, rfid in pervasive computing: State-of-the-art and outlook, *Pervasive and Mobile Computing*, Volume 5, Issue 1 pp. 110-131, ISSN:1574-1192, 2009.

[2] I. Dzitac and B. E. Barbat, Artificial Intelligence + Distributed Systems = Agents, *International Journal of Computers, Communications and Control (IJCCC)*, Vol 4(1), pp. 17-27, ISSN: 1841-9836, 2009.

[3] Distributed Systems Research Laboratory, Technical University of Cluj-Napoca, *http://dsrl.coned.utcluj.ro*.

[4] D. Fournier, S. Ben Mokhtar and N. Georgantas, Towards Ad hoc Contextual Services for Pervasive Computing, *Proceedings of the 1st workshop on Middleware for Service Oriented Computing*, pp. 36 - 41, ISBN:1-59593-425-1, 2006.

[5] K. M. Anderson, F. A. Hansen and N. O. Bouvin, Templates and queries in contextual hypermedia, *Proceedings of the seventeenth conference on Hypertext and hypermedia*, pp. 99 - 110, 2006.

[6] D. Raz, A. Tapani Juhola, J. Serrat-Fernandez and A. Galis, *Fast and Efficient Context-Aware Services*, Wiley Series on Communications Networking and Distributed Systems, ISBN-13: 978-0470016688, pp. 5-25, 2006.

[7] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann and W. Retschitzegger, Context-awareness on mobile devices - the hydrogen approach, *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, pp. 292, USA, 2003.

[8] A. Rarau, K. Pusztai and I.Salomie, MultiFacet Item based Context-Aware Applications, *Inernational Journal of Computer and Information Sciences 3(2)*, pp. 10-18, 2006.

[9] I. Salomie, T. Cioara, I. Anghel and M. Dinsoreanu, RAP - A Basic Context Awareness Model, *Proc. Of 4th IEEE Int. Conf. on Intelligent Computer Communication and Processing*, ISBN: 978-1-4244-2673-7, pp. 315- 318, 2008.

[10] G. Specht and T. Weithoner, Context-Aware Processing of Ontologies in Mobile Environments, *7th Int. Conference on Mobile Data Management*, ISBN: 978-1-4244-2673-7, pp. 86, May 2006.

[11] K. C. Lee, J. H. Kim and J. H. Lee, Implementation of Ontology Based Context-Awareness Framework for Ubiquitous Environment, *Int. Conference on Multimedia and Ubiquitous Engineering*, pp. 278 - 282, April 2007.

[12] P. Bellavista, A. Corradi and R. Montanari, Mobile Computing Middleware for Location and Context-Aware Internet Data Serices, *ACM Transactions on Internet Technology (TOIT)*, Volume 6 , Issue 4, pp. 356 - 380, ISSN:1533-5399, 2006.

[13] G. Spanoudakis and K. Mahbub, A Platform for Context Aware Runtime Web Service Discovery, *IEEE Int. Conference on Web Services*, pp. 233-240, ISBN: 0-7695-2924-0, 2007.

[14] C. Klein, R. Schmid, C. Leuxner, W. Sitou and B. Spanfelner, A Survey of Context Adaptation in Autonomic Computing, *Proc. of the Fourth International Conference on Autonomic and Autonomous Systems*, pp. 106 - 111, 2008.

[15] M. Cremene, M. Riveill and C. Martel, A Service-Context Model allowing Dynamical Adaptation, *International Journal of Computers, Communications and Control (IJCCC)*, Vol 1, pp. 163-170, ISSN: 1841-9836, 2006.

[16] F. Seyler, C. Taconet and G. Bernard, Context Aware Orchestration Meta-Model, *Proc. Of the Third Int. Conference on Autonomic and Autonomous Systems*, pp. 17, 2007.

[17] Y. C. Zhou, X. P. Liu, E. Kahan, X. N. Wang, L. Xue and K. X. Zhou, Context Aware Service Policy Orchestration, *Proc of the IEEE International Conference on Web Services*, pp. 936-943, 2007.

[18] I. Y.L. Chen, S. J.H. Yang and J. Zhang, Ubiquitous Provision of Context Aware Web Services, *IEEE International Conference on Services Computing (SCC'06)*, pp.60-68, ISBN: 0-7695-2670-5, 2006.

[19] N. O'Connor, R. Cunningham and V. Cahill, Self-Adapting Context Definition, *First International Conference on Self-Adaptive and Self-Organizing Systems*, pp. 336-339, 2007.

[20] R. Calinescu, Model-Driven Autonomic Architecture, *Proc. of the Fourth International Conference on Autonomic Computing*, pp. 9, 2007.

[21] E. Patouni and N. Alonistioti, A Framework for the Deployment of Self-Managing and Self-Configuring Components in Autonomic Environments, *Proc. of the Int. Symposium on a World of Wireless, Mobile and Multimedia Networks*, pp. 484-489, 2006.

[22] Infusion Systems Ltd, *www.infusionsystems.com*.

Ionut Anghel (February 28, 1983) is an Assistant Professor and PhD Student at the Computer Science Department of the Technical University of Cluj-Napoca. He is currently developing the PhD thesis "Autonomic Computing Middleware". His main research areas are Autonomic systems, Mobile agents and Ontology learning. Also he was involved in four national funded research projects and published several papers in international databases-indexed proceedings and journals in the last 2 years. Ionut also co-authored a book in distributed systems in 2008.