A Self-configuring Middleware Solution for Context Management

Tudor Cioara, Ionut Anghel, Ioan Salomie

Computer Science Department, Technical University of Cluj-Napoca, 15 Daicoviciu Street, 400020 Cluj-Napoca, Romania
{Tudor.Cioara, Ionut.Anghel, Ioan.Salomie}@cs.utcluj.ro

Abstract. This paper proposes a self-configuring middleware that uses a context management infrastructure to gather context data from various context sources and generate/update a run-time context representation. The high demand for reducing the context representation management complexity and ensuring a high tolerance and robustness, lead us to considering the self-configuring autonomic computing paradigm for the context acquisition and representation processes. The middleware defines three main layers: the acquisition layer that captures the context data from real world contexts, the context model layer that represents the context data in a programmatic manner and the context model management infrastructure layer. The middleware continuously monitors the real context to detect context variations or conditions for updating the context representation. The proposed middleware was tested and validated within the premises of our Distributed Systems Research Laboratory smart environment.

Keywords: Autonomic Context Management, Self-Configuring, Middleware, Context Model

1 Introduction and Related Work

An important challenge in developing context aware systems is the dynamic nature of their execution environment, which makes the process of context information acquisition and representation extremely difficult to manage. During the context information acquisition process, the sources of context information (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. For example, a context aware system cannot wait indefinitely for an answer from a temporary unavailable context resource. On the other hand, many times the payoff for not taking into consideration the new available context resources can be very high. To provide an efficient context information management, it is necessary to introduce some degree of autonomy for the context acquisition and representation processes.

Another important challenge in the context aware systems development is the task of assigning the context management responsibility. Current approaches put the system developers in charge with the context management process, making system

development extremely complicated. Our vision is that a third party context management infrastructure must deal with the processes of context information acquisition and representation.

This paper offers a solution for these challenges by introducing a self-configuring middleware that uses a context management infrastructure to gather context information from various context sources and generate a run-time context representation. Therefore, the context management processes are transparent for the context aware systems developers, allowing them to concentrate on designing and implementing the system desired functionality.

The research related to the autonomic context management is focused on two major directions (i) the development of models and tools for acquiring and formally representing the system execution context and (ii) the development of models and techniques for analyzing, processing and managing the context representation without human intervention.

The most important research problems related to *context information acquisition* are to identify the features defining the system execution context [1] and to define models for capturing context features specific data [2]. In the domain literature [3, 4], several system execution context features are considered such as: spatiotemporal (time and location), ambiental and facility (the system devices and their capabilities), user-system interaction, system internal events, system life cycle, etc. Regarding *context representation*, generic models aiming at accurately describing the system execution context in a programmatic manner are proposed. In [5], the authors propose the use of key-value models to represent the set of context features and their associated values. Markup and object oriented models [6, 7] are also used to structure and represent the context information. In [8], context features are represented as ontological concepts in design time and instantiated during run-time with sensor captured values. The main drawback of these approaches is the lack of semantic information encapsulated in the context representation which makes difficult the process of inferring new context related knowledge. Our paper overcomes these deficiencies by using the set-oriented and ontology based RAP context model [9] to represent the context information in a programmatic manner. The set representations of the RAP context model are used by the context management middleware to detect the context changes while the ontology representation is used to infer new context related information through reasoning algorithms.

In the *context management* research direction, the efforts are concentrated on developing models and techniques for: (i) keeping the context representation consistent with the real context and (ii) processing and analyzing the context representation for inferring new context related knowledge and evaluate the context changes. To ensure the consistency of context representation, models and tools that allow for the automatic discovery, installation and configuration of new context information sources are proposed. In [10], the authors describe models for capturing and updating the context information based on the information type. Fournier [11] defines reusable components for updating the context specific data. These components provide stable communication channels for capturing and controlling context specific data. In [12], the development of context guided behavioral models, allowing the detection of only those context data variations that lead their behavior modification, is discussed. The main disadvantage of these approaches is the lack of

efficiency for the context management process that is rather static and difficult to adapt to context changes. There is a high demand for reducing the context model management complexity while ensuring a higher tolerance and robustness, leading to the consideration of the self-configuring autonomic computing paradigm [13]. The specification and representation of configuration, discovery and integration requirements of resource components have been identified as main research problems [14]. In [15], a model for self-configuring the new added components based on policies is proposed. The self-configuring policies are stored into a repository, which is queried when a new component is added. In [16], the authors present an autonomic 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 the context information. This description is frequently updated and used for self-configuring and taking adapting decisions. For the *context processing and analyzing* research direction, models and techniques that aim at determining and evaluating the context changes are proposed. These models are strongly correlated with the context representation model. In [17], fuzzy Petri nets are used to describe context changing rules. Data obtained from sensors, together with user profiles and requests represent the input data for the reasoning mechanism. Context analyzing models based on reasoning and learning on the available context information are presented in [19, 20]. Context changing rules can be described using natural language [18] or first order logic and evaluated using reasoning engines.

The main contribution of our approach is the definition of a self-configuring middleware targeting an efficient and autonomic context management. The fundamental element of this middleware is our RAP context model which uses the concepts of context *Resources, Actors* and *Policies* to formally represent specific context data. The context model management infrastructure is implemented by using BDI (Believe, Desire, Intention) agents [21] 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 updated the context artifacts. The proposed middleware was tested and validated using our Distributed Systems Research Laboratory [22] as a smart space infrastructure.

The rest of the paper is organized as follows: in Section 2, the middleware architecture is presented; Section 3 details the self-configuring enhanced middleware; Section 4 shows how the middleware is used to manage the context representation of an intelligent laboratory environment while Section 5 concludes the paper and shows the future work.

2 The Middleware Architecture

The middleware architecture defines three main layers (see Fig. 1): the *acquisition layer* that captures the context information from real world contexts, the *context model layer* that represents the context information in a machine interpretable manner and the *context model management infrastructure layer*. In the following sections, we detail each of the three middleware architectural layers.

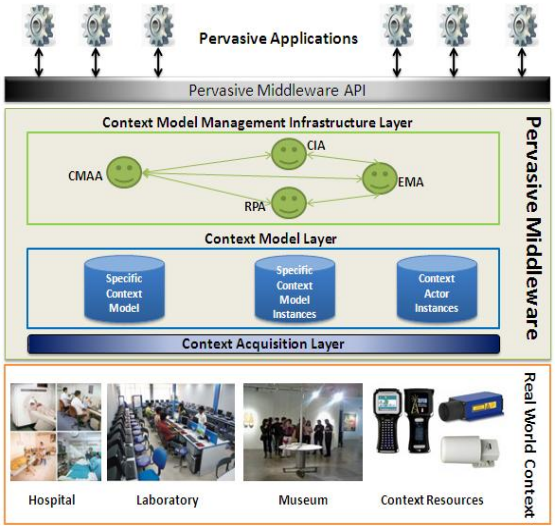

Fig. 1. The middleware conceptual architecture

2.1 The Context Acquisition Layer

The context acquisition layer collects information from various context sources (sensor, intelligent devices, etc.) and makes it available to the context model layer (see Fig. 2.a) through a *Context Acquisition API*. To make sensor information visible to the upper layers in an independent way, we have used the web services technology. Each sensor has an attached web service for exposing its values. The structure of the Context Acquisition API is presented in Fig. 2.b. The communication between a sensor attached web service and the Context Acquisition API is managed by the *WSClient* class. It provides methods that: (i) builds a SOAP request, (ii) sends the request to the web service and (iii) waits for the sensor value response.

From middleware perspective, the context acquisition layer defines both push and pull mechanisms for sensor information retrieval. The push mechanism uses event listeners for gathering context data from sensors while the pull mechanism uses a query based approach that allows the context data to be provided on demand. The pull information retrieval mechanism is implemented in the *SensorTools* class by defining a method that queries a specific web service to obtain the sensor value. For the push mechanism, the Observer design pattern is used. A *SensorWSReader* instance must be created first by specifying the URL of the web service and the time interval at which the sensor data will be updated. The *SensorWSReader* instance also contains a list of listeners that are notified when a sensor value has changed. The listeners are created by the middleware upper layers by extending the *AbstractSensorListener* class. To verify the sensor value, separate threads that continuously send requests to the web service are created using the *WSReaderThread*.

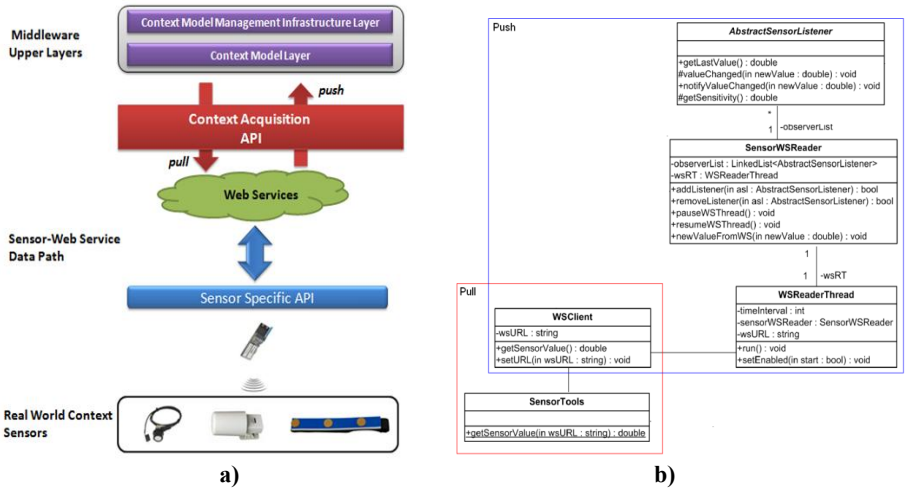

Fig. 2. (*a*) The context data retrieval flow and (*b*) the Context Acquisition API class diagram

2.2 The Context Model Layer

To represent the real world context in a programmatic manner, the RAP context model is used. The RAP model represents the context information as a triple: C = <R, A, P>, where R is the set of context resources that captures 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: R_E - the set of environment context resources and R_A - the set of actor context resources.

The accurate representation of the real world contexts is achieved by defining the artifacts of (see Fig. 3a): *specific context model* C_S, *specific context model instance* C_{SI} and *context – actor instance* CI_a^t.

The specific context model $C_S = <R_S, A_S, P_S>$ maps the context model onto real contexts and populates the context model sets with context specific actors, resources and policies. A specific context model instance $C_{SI}^t = < R_{SI}^t, A_{SI}^t, P_{SI}^t >$ 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. A context – actor instance represents the projection of the specific context model instance onto a certain actor.

The RAP model also offers an ontological representation of the context model artifacts, which allows for learning and reasoning processes in order to obtain context knowledge (Fig. 3b). The relationships between the R, A and P context model elements are represented in a general purpose context ontology core. The specific context model concepts are represented as sub trees of the core ontology. A context situation or a context instance is represented by the core ontology together with the specific context model sub trees and their individuals in a specific moment of time.

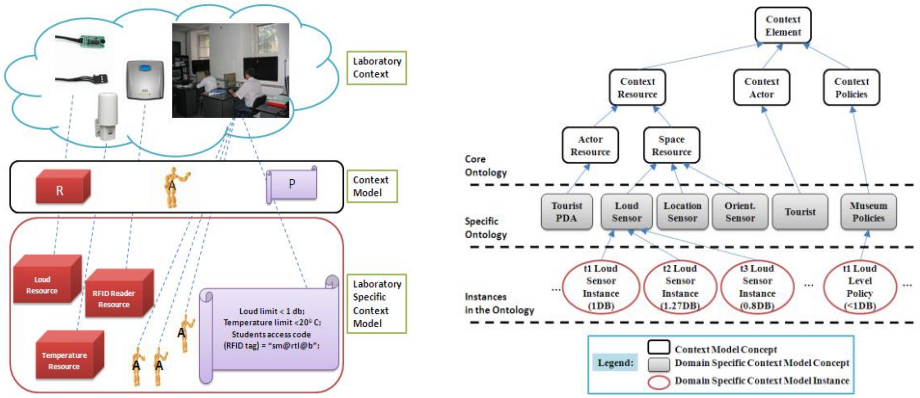

Fig. 3. The RAP context model context representation: (*a*) set-based and (*b*) ontology-based

The set-based and ontology-based context representations are equivalent and need to be kept synchronized. The set-based context representation is used to evaluate the conditions under which the context management agents should execute self-* processes in order to enforce the autonomic properties at the middleware level. The ontology-based model uses reasoning and learning processes for generating new context knowledge.

2.3 The Context Model Management Infrastructure Layer

The context model management infrastructure layer is based on four cooperative BDI type agents: *Context Model Administering Agents, Context Interpreting Agents, Request Processing Agents* and *Execution and Monitoring Agents*.

The *Context Model Administering Agent* (CMA Agent) is the manager of the specific context model. Its main goal is to synchronize RAP context model artifacts with the real context. This agent is also responsible for the negotiating processes that take place when an actor or resource is joining the context. The *Context Interpreting Agent* (CI agent) semantically evaluates the information of a context instance and identifies the context instance "meaning". The *Request Processing Agent* (RP agent) processes the actor requests. This agent identifies and generates the action plans that must be executed for serving an incoming request. The RP agent uses the specific context model instance to identify / generate the adequate plan to be executed by the Execution and Monitoring Agent. The *Execution and Monitoring Agent* (EM agent) executes the action plans received from the RP agent using the available services. After mapping the action plans onto services, a plan orchestration is obtained and executed using transactional principles.

The context management infrastructure agents are implemented using the Java Agent Development Framework platform [23]. When the middleware is deployed, the CMA agent is the first running agent. It instantiates the CI, RP and EM agents and sends them the context representation.

3 Enhancing the Middleware with Self-Configuring Capabilities

The middleware context acquisition and representation processes need to be reliable and fault tolerant because the context resources can fail or new resources may be identified at run-time. Consequently, the context representation constructed by de middleware needs to accurately reflect the real context. To provide an efficient, fault tolerant and robust context management, the middleware is enhanced with self-configuring properties.

The self-configuring property is enforced by monitoring the real world context to detect context variations or conditions for which the context artifacts must be updated. We have identified three causes that might generate context variation: (1) adding or removing context elements (resources, actors or policies) to / from the context, (2) actors' mobility within the context and (3) changes of the resources property values (mainly due to changing the sensors' captured values). In the following sections we discuss each of the context variation sources targeting to determine: (i) the context variation degree and (ii) the triggering condition of the self-configuring process.

3.1 Context Variation Generated by Adding or Removing Context Elements

During the context data acquisition process, the sources of context data 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 the CMA agent which updates the RAP specific context model, according to the new real context. Next, we evaluate the context variation degree generated by: (1) context resources ΔR, (2) context policies ΔP and (3) context actors ΔA against the values of the associated defined thresholds T_R, T_P, and T_A.

The context resources set variation ΔR is generated by adding or removing a context resource r (sensor or actuator) to / from the real context. ΔR is calculated using the set difference operation applied for two consecutive moments of time: t and t+1, where t+1 represent the moment when the resource r became available. The same reasoning pattern is applied when the resource r fails or becomes unavailable:

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

In formula (1) $R_E^{t+1} \setminus R_E^{t}$ contains the set of context resources that become available at t+1 while $R_E^{t} \setminus R_E^{t+1}$ contains the set of context resources that become unavailable at t+1. If $Card(\Delta R) \geq T_R$, the RAP specific context model is updated by adding or removing the context resources contained in ΔR.

The variation of the policy set ΔP is generated by adding, removing or updating a context policy. Using the same assumptions and conclusions as for context resources, the policy set variation is calculated as:

$$\Delta P = \{P^{t+1} \setminus P^{t}\} \cup \{P^{t} \setminus P^{t+1}\} \quad (2)$$

The variation of the actors set ΔA is generated by the actors that enter or leave the context. Each context actor has an attached context resources set during its context interactions. An actor features a large number of actor-context interaction patterns, but only two of these patterns may determine the actor set variation: (i) the actor enters the context and (ii) the actor leaves the context. The actor's context variation is:

$$\Delta A = \{A^{t+1} \setminus A^{t}\} \cup \{A^{t} \setminus A^{t+1}\} \cup \{R_A{}^{t} \setminus R_A{}^{t+1}\} \cup \{R_A{}^{t+1} \setminus R_A{}^{t}\} \quad (3)$$

Overall, the RAP model context variation ΔRAP is given by the union of all context elements' variations, as shown below:

$$\begin{aligned} \Delta RAP &= \Delta R \cup \Delta A \cup \Delta P \\ Card(\Delta RAP) &= Card(\Delta R) + Card(\Delta A) + Card(\Delta P) \end{aligned} \quad (4)$$

The CMA agent starts the execution of the self-configuring process and updates the context model when $Card(\Delta RAP) \geq T_{Self\text{-}Configuring}$ where the self-configuring threshold is defined as:

$$T_{Self\text{-}Configuring} = \min(T_R, T_A, T_P) \quad (5)$$

3.2 Context Variation Generated by Actors Mobility

Due to their mobility, model actors are changing their context location and implicitly the set of context resources with which they may interact. The CMA agent identifies this variation and generates a new context-actor instance and updates the specific context model instance. To evaluate the context variation generated by actors' mobility we use the *isotropic context space* concept, as defined in [9]. A context space is isotropic if and only if the set of context resources remains 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, called *Context Granules (CG)*. For a given moment of time, an actor can be physically located in a single CG. As a result, the space isotropy variation ΔIZ is non-zero only when an actor moves between two CGs. The isotropy variation for a context actor is calculated as:

$$\Delta IZ_a = \{R_{CG}{}^{t+1} \setminus R_{CG}{}^{t}\} \cup \{R_{CG}{}^{t} \setminus R_{CG}{}^{t+1}\} \quad (6)$$

The CMA agent continuously monitors the actors' movement in the real context and periodically evaluates the space isotropy variation. If for an actor, the space isotropy variation is non-empty, then the self-configuring process executed by the CMA agent updates the context-actor instance. It actually represents the specific context model instance projection onto a certain actor:

$$CI_a^{t+1} = <R_a^{t+1}, a, P^{t+1}> \mid R_a^{t+1} = R_{CG}^{t+1} \quad (7)$$

The context variation ΔCAM, generated by all actors' mobility in a context is:

$$\Delta CAM = \cup_{a \in A} \Delta IZ_a \quad (8)$$

3.3 Context Variation Generated by Changes of Resources Property Values

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

$$K_{val}(R) = \{(k_1,val_1),\ldots, (k_n,val_n)\} \mid k_1,\ldots,k_n \in K(R) \quad (9)$$

If the values captured by the Hot&Humidity sensor in a moment of time is 5 degree Celsius for temperature and 60%, for humidity, then K_{val}(Hot&HumiditySensor) = {(Temperature, 5), (Humidity, 60%)}. CMA agent calculates the context variation generated by changes of resource properties' values (ΔRPV) as presented in 10. As a result, a new specific context model instance is created when Card(ΔRPV) ≥ 0.

$$\Delta RPV = K_{val}(R^{t+1}) - K_{val}(R^t)=\{(k_1,val_1^{t+1} - val_1^t),\ldots,(k_n,val_n^{t+1}-val_n^t)\} \quad (10)$$

3.4 The Self-Configuring Algorithm

The CMA agent executes the self-configuring algorithm in order to keep the context model artifacts synchronized with the real context (see Fig. 4). The CMA agent periodically evaluates the context changes. When a significant context variation is determined, the context model ontology artifacts are updated using the *updateOntologyModel (owlModel, C_S^{t+1}, CI_a^{t+1}, C_{SI}^{t+1})* method.

```
Algorithm CMA_Self_Configuring
input: new real world context elements - R^{t+1}, A^{t+1}, P^{t+1}
       context elements variation thresholds - T_R, T_A, T_P
output: new context artifacts - C_S^{t+1}, CI_a^{t+1}, C_SI^{t+1}
resources: current context artifacts set - C_S^t, CI_a^t, C_SI^t
           context ontology - owlModel
begin
  ΔR = {R_E^{t+1} \ R_E^t} ∪ {R_E^t \ R_E^{t+1}}
  ΔA = (A^{t+1} \ A^t} ∪ {A^t \ A^{t+1}} ∪ {R_A^{t+1} \ R_A^t} ∪ {R_A^t \ R_A^{t+1}}
  ΔP = {P^{t+1} \ P^t} ∪{P^t \ P^{t+1}}
  ΔRAP = ΔR ∪ ΔA ∪ ΔP
  ΔRPV = K_val(R^{t+1}) - K_val(R^t)
  ΔCAM = ∪_{a ∈ A} ΔIZ_a
  T_Self-Conf = min (T_R, T_A, T_P)
  if  (Card (ΔRAP) ≥ T_Self-Conf )
    begin //CMA agent creates a new specific context model
      if  (Card (ΔR) ≥ T_R )
        if (R^t ∩ ΔR = ∅) C_S^{t+1} = C_S^t + ΔR = (R^t, A^t, P^t) + ΔR = (R^t ∪ ΔR, A^t, P^t)
                    else C_S^{t+1} = C_S^t - ΔR = (R^t, A^t, P^t) - ΔR = (R^t \ ΔR, A^t, P^t)
      if (Card (ΔA) ≥ T_A )
        if (A^t ∩ ΔA = ∅)  C_S^{t+1} = C_S^t + ΔA = (R^t, A^t, P^t) + ΔA = (R^t, A^t ∪ ΔA, P^t)
                    else C_S^{t+1} = C_S^t - ΔA = (R^t, A^t, P^t) - ΔA = (R^t, A^t \ Δ_A, P^t)
      if (Card (ΔP) ≥ T_P )
        if (P^t ∩ ΔP = ∅)  C_S^{t+1} = C_S^t + ΔP = (R^t, A^t, P^t) + ΔP = (R^t, A^t, P^t ∪ ΔP)
                    else C_S^{t+1} = C_S^t - ΔP = (R^t, A^t, P^t) - ΔP = (R^t, A^t, P^t \ ΔP)
    end
  else
    begin
      if (Card (ΔCAM) ≥ T_Self-Conf ) // CMA creates a new context-actor instance
        foreach a ∈ A if (ΔIZ_a ≠ ∅) CI_a^{t+1} = <R_a^{t+1}, a, P^{t+1}>
      else // CMA agent creates a new specific context instance
        if (Card (ΔRPV) ≥ T_Self-Conf ) C_SI^{t+1} = <R_SI^{t+1}, A_SI^{t+1}, P_SI^{t+1}>
    end
  updateOntologyModel (owlModel, C_S^{t+1}, CI_a^{t+1}, C_SI^{t+1})
end
```

Fig. 4. The CMA agent self-configuring algorithm

4 Case Study and Results

For the case study we have considered a real context represented by our Distributed System Research Laboratory (DSRL). In the laboratory the students are marked and identified by using RFID tags and readers. The students interact with the smart laboratory by means of wireless capable PDAs on which different laboratory provided services are executed (for example: submit homework services, lesson hints services, print services, information retrieval services, etc.). A sensor network captures information regarding students' location and ambiental information such as temperature or humidity. In the laboratory, a set of policies like "*the temperature should be 22 degrees Celsius*" or "*the loud upper limit is 80 dB*" should be respected.

The DSRL infrastructure contains a set of sensors through which context data is collected: two Hot&Humidity sensors that capture the air humidity and the temperature, four Orient sensors placed in the upper 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 Fig. 5). The sensors are connected through a Wi-microSystem wireless network from Infusion Systems [24]. The middleware is deployed on an IBM Blade-based technology physical server. The IBM Blade technology was chosen because its 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 analogue to digital encoder as its main part. The WimicroDig is a configurable hardware device that encodes up to 8 analogue sensor signals to MIDI messages which are real-time wirelessly transmitted, through Bluetooth waves, to the server for analysis and / or control purposes. The Bluetooth receiver located on the server is mapped as a Virtual Serial Port (VSP).

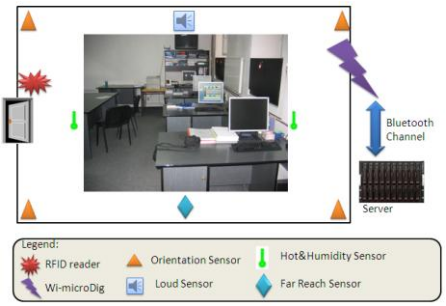

Fig. 5. The DSRL infrastructure

In order to read/write to/from the VSP we used two sensor manufacture applications: (i) BlueMIDI which converts the Bluetooth waves received on the VSP into MIDI messages and (ii) MIDI Yoke which creates pairs of input/output MIDI ports and associates the output MIDI port with the VSP. The MIDI message information is extracted using the Microsoft Windows API multimedia operations and published through web services (see Fig. 6).

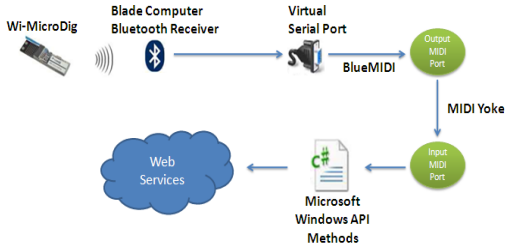

Fig. 6. The context information data path form sensors to their attached web services

The CMA agent 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 3.4. When the middleware is deployed and starts its execution (t=0) there are no context model artifacts constructed yet, i.e. the R, A and P sets of the RAP context model are empty. After one second (t=1), when two students John and Mary enter the lab, the CMA agent receives the updated context information from the Context Acquisition Layer and calculates the context elements variation ΔR, ΔP and ΔA as presented in Fig. 7a. By default the self-configuring thresholds are set to the value 1: $T_{Self\text{-}Conf} = T_R = T_A = T_P = 1$. As a result of evaluating the context variation at t=1, the CMA agent executes the self –

configuring algorithm which adds new concepts and updates the context model artifacts ontology. The new added concepts (see Fig. 7a) originate from the context elements set variations ΔR, ΔP and ΔA. To test the middleware self-configuring capabilities we have considered that after 60 seconds the following context changes have occurred: (i) student John leaves the laboratory, (ii) Orientation Sensor1 and OrientationSensor4 are disabled and (iii) LoudSensor is disabled. The CMA agent calculates the variation in the new context at t = 61 (Fig. 7b), executes the self-configuring algorithm and updates accordingly the context ontology.

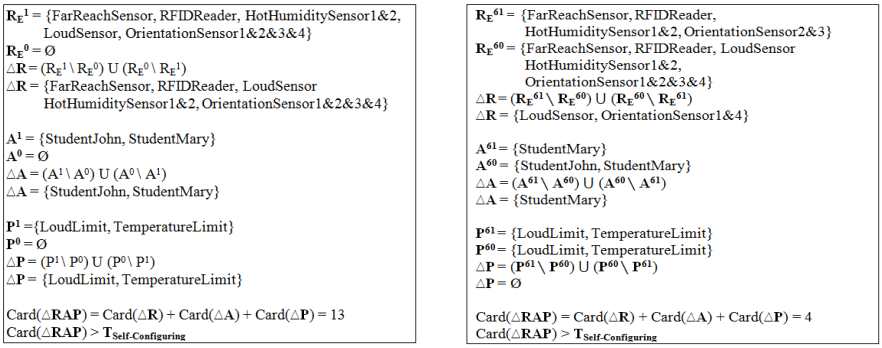

Fig. 7. DSRL context variation at: (*a*) t=1 and (*b*) t=61

To test the scalability of our self-configuring algorithm we have implemented an application that can simulate the behavior of a large number of sensors that randomly generate context data at fixed time periods. The results show that the self-configuring algorithm implemented by CMA agent can generate, synchronize and update the context model artifacts that change their values simultaneously in a reasonable time for up to 20 sensors (Fig. 8). However, it is possible that sensor values change much faster than the CMA agent is capable of synchronizing the contexts representation, thus requiring a higher ticker interval value.

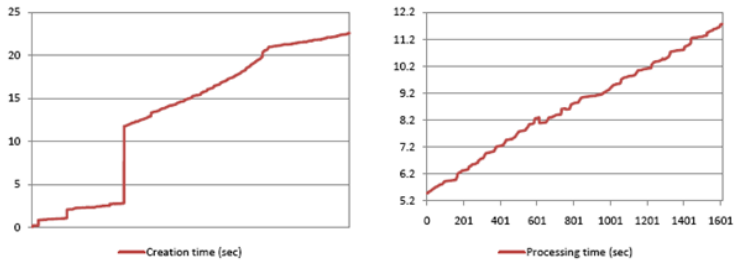

Fig. 8. The self-configuring algorithm scalability results

To assess the overhead of the proposed self-configuring algorithm, a simulation editor was developed in which complex test cases can be described by generating sets of (simulation time, sensor value) associations. We evaluated the memory and

processor loading when executing the self-configuring algorithm to update the specific context model instance due to sensor values changes. Using the simulator, we tested our middleware with 100 sensors changing their values every 100ms for the first test case and every 2000ms for the second test case. Even if the sensor values change rate is much higher in the first test case than in the second test case, the memory and processor loading did not show major differences (see Fig. 9).

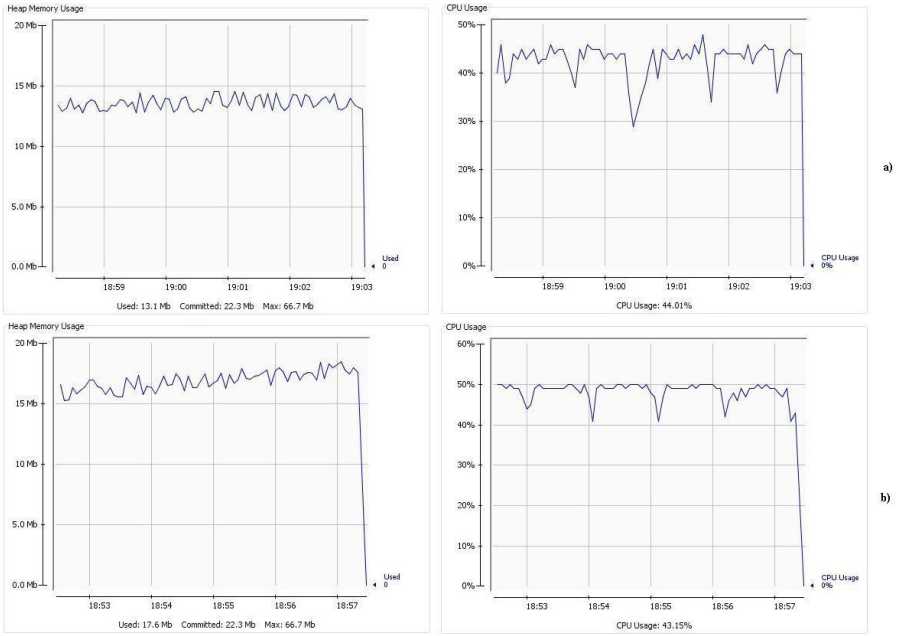

Fig. 9. The self-configuring algorithm CPU and memory overloading with 100 sensors at (*a*) t2=2000 ms and (*b*) t1=100 ms

5 Conclusions

This paper addresses the problem of managing the context information acquisition and representation processes in a reliable and fault tolerant manner by using a self-configuring middleware. The middleware defines an agent based context management infrastructure to gather context data from sensors and generate a RAP model context representation at run-time. The self-configuring property is enforced at the middleware level by monitoring the context in order to detect context variations or conditions for which the context model artifacts must be created / updated. The evaluation results are promising showing that the self-configuring algorithm can manage in a reasonable time up 20 sensors which change their values simultaneously at a high sampling rate. Also we have proved that the memory and processor overload induced by executing the self-configuring algorithm is negligible.

References

1. Wang K.: Context awareness and adaptation in mobile learning. In: Proc. of the 2nd IEEE Int. Wshop. on Wireless and Mobile Tech. in Education, pp. 154-158, ISBN:0-7695-1989-X (2004)
2. Yu Z., Zhou X., Park J.H.: iMuseum: A scalable context-aware intelligent museum system. Computer Communications, Volume 31, Issue 18, pp. 4376-4382 (2008)
3. Pareschi L.: Composition and Generalization of Context Datafor Privacy Preservation. 6th IEEE Int. Conf. on Perv. Comp. and Comm., ISBN: 0-7695-3113-X, 429-433 (2008)
4. Grossniklauss M.: Context Aware Data Management, 1st ed. VDM Verlag, ISBN 978-3-8364-2938-2 (2007)
5. Anderson K., Hansen F.: Templates and queries in contextual hypermedia. In: Proc. of the 17th Conf. on Hypertext and hypermedia, ISBN:1-59593-417-0, pp. 99 – 110 (2006)
6. Raz D., Juhola A. T.: Fast and Efficient Context-Aware Services. Wiley Series on Comm. Networking & Distributed Systems, ISBN-13: 978-0470016688, pp. 5-25 (2006)
7. Hofer T.: Context-awareness on mobile devices – the hydrogen approach. In: Proc. of the 36th Hawaii Int. Conf. on System Sciences, USA, ISBN:0-7695-1874-5, pp. 292 (2003)
8. Cafezeiro I., Hermann E.: Ontology and Context. In: Proc. Of the 6th Annual IEEE Int. Conf. on Pervasive Comp. and Comm., ISBN: 978-0-7695-3113-7, pp. 417-422 (2008)
9. Salomie I., Cioara T., Anghel I.: RAP-A Basic Context Awareness Model. In: Proc. Of The 4th IEEE Int. Conf. on Intelligent Comp Comm. and Proc., ISBN: 978-1-4244-2673-7, pp. 315-318, Cluj-Napoca, Romania (2008)
10. Bellavista P.: Mobile Computing Middleware for Location and Context-Aware Internet Data Services. ACM Trans. on Internet Tech., ISSN:1533-5399, pp. 356 - 380 (2006)
11. Fournier D., Mokhtar S.B.: Towards Ad hoc Contextual Services for Pervasive Computing. In: IEEE Middleware for S.O.C., pp 36 - 41, ISBN:1-59593-425-1 (2006)
12. Spanoudakis G., Mahbub K.: A Platform for Context Aware Runtime Web Service Discovery. In: IEEE Int. Conf. on Web Services, pp 233-240, USA (2007)
13. Calinescu R.: Model-Driven Autonomic Architecture. In: Proc. of the Fourth International Conference on Autonomic Computing, ISBN: 0-7695-2779-5, pp. 9 (2007)
14. Patouni E., Alonistioti N.: A Framework for the Deployment of Self-Managing and Self-Configuring Components in Autonomic Environments. In: Proc. of the Int. Symp. on a World of Wireless, Mobile and Multimedia, ISBN:0-7695-2593-8, pp. 484-489 (2006)
15. Bahati R.: Using Policies to Drive Autonomic Management. In: Proc. of the Int. Symp. on a World of Wireless, Mob. and Multimedia, ISBN:0-7695-2593-8, pp. 475-479 (2006)
16. Cremene M, Riveill M.: Autonomic Adaptation based on Service-Context Adequacy Determination. In: Electronic Notes in Theoretical Comp. Sc., ISSN:1571-0661, pp. 35-50, Elsevier (2007)
17. Huaifeng Q.: Integrating Context Aware with Sensornet. In: Proc. of 1st Int Conf. on Semantics, Knowledge, Grid, ISBN:0-7695-2534-2, Beijing, China (2006)
18. Bernstein A.: Querying the Semantic Web with Ginseng:A Guided Input Natural Language Search Engine. 15th Workshop. on Inf. Tech. and Syst., pp. 112-126 (2005)
19. Sirin E., Parsia B.: Pellet: A practical OWL-DL reasoner. In: Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 5, No. 2, pp. 51-53, Elsevier (2007)
20. Amoui M., Salehie M.: Adaptive Action Selection in Autonomic Software Using Reinforcement Learning. In: Proc. of the 4th Int. Conf. on Aut. and Autonomous Sys., pp. 175-181, ISBN 0-7695-3093-1 (2008)
21. Thangarajah J., Padgham L.: Representation and reasoning for goals in BDI agents. In: Proc. of the 25th Australasian Conf. on Comp. Sci., pp. 259-265, ISSN:1445-1336 (2002)
22. Distributed Systems Research Laboratory, dsrl.coned.utcluj.ro
23. Jade-Java Agent DEvelopment Framework, http://jade.tilab.com
24. Infusion Systems Ltd., http://www.infusionsystems.com