A Generic Context Model Enhanced with Self-Configuring Features

Tudor Cioara, Ionut Anghel, Ioan Salomie, Mihaela Dinsoreanu

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

Abstract. *This paper addresses two fundamental research problems in the domain of context aware autonomic systems: the development of a generic context model that can be used to represent general purpose contexts in a system interpretable way and the autonomic context model management. The proposed context model uses two equivalent and synchronized ways of representing the context: a set based representation and an ontology based representation. The set based representation is used to evaluate the conditions under which the self-* processes should be executed in order to enforce the autonomic properties. The ontology representation is used by context aware applications for reasoning and learning purposes. The increasing complexity of execution environments of the context aware systems and the difficulties of their management have headed us towards the necessity of defining and integrating the self-configuring autonomic paradigm in the context model management process. The self–configuring property is enforced by monitoring the real world context in order to detect context variations or conditions for which the context artifacts must be updated.*

Categories and Subject Descriptors
H.1.1 **[Models and Principles]:** Systems and Information Theory - *general systems theory, information theory*.

General Terms
Algorithms; Management; Design; Reliability; Languages; Theory.

Keywords: Self-Configuring; Pervasive Middleware; Context Awareness; Autonomic Computing;

1. Introduction and related work

An important challenge in developing context aware systems is the dynamic nature of their execution environment which makes the processes 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.

The *objective of this paper* is to define a self-configuring context model that accurately captures and represents general purpose real contexts, targeting the development of context aware autonomic systems. The researches efforts in the context aware autonomic systems domain are concentrated on two directions: (i) the development of generic context models that can be used to represent the system environment and (ii) the development and integration of context aware systems self-* enhanced components.

In the *real world context modeling* research direction many approaches have been proposed. In [Rarau, 2006] the concept of multi-faceted entity is defined for modeling the set of context properties. A facet is represented as the effective values of context properties, at a particular moment, to which the context sensitive application has access. The main drawback of this approach is the lack of semantic information encapsulated in the facet concept. As a result, inferring new context related knowledge is difficult. An original approach to the context modeling problem is the use of parametric state machines to represent a context aware system [Chen, 2006]. The context is modeled using context functions that modify the context aware system's state. The complexity of a real system's associated parametric state machine, in terms of number of states and transitions, is the main disadvantage of this approach. The use of ontologies is a another direction for context representation.

The context properties are represented as ontological concepts during design time and instantiated with actual values, captured by sensors, during execution [Lee, 2007][Feruzan Ay, 2007]. In this way, context properties relations are easily modeled. The main disadvantage is the high degree of inflexibility determined by the human factor intervention in context representation. In [O'Connor, 2007] the authors propose the construction of a system situation space where system execution context is represented as group in this space. A function can be defined taking values in the context set of situations, with values in the system's action set. 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.

Regarding the *context aware self-* systems***,** most of the researches reported in the literature are focused on the self-adaptation problem. Research efforts are made to create new models and algorithms that allow computational systems to execute specific actions according to the context or situation at hand. The objective is to associate a certain degree of intelligence to the computational systems for context adaptation. In [Cremene, 2007], 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. [Desmet, 2007], [Santos, 2007] and [Seyler, 2007] propose context adaptation models based on defining the system behavior in a certain situation using a set of context adapting rules. A rule consists of a context condition and an associated action. The main disadvantage of these approaches is given by the fact that new rules can not be learned or inferred during run time. In [Bellavista, 2006], a model for capturing and updating the context information based on information type are proposed. The authors classify the context information in three types: user context information, physical context information and computational context information. Another approach [Fournier, 2006] is to define reusable components for updating the context specific data. These components provide stable communication channels for capturing and controlling context specific data. [Spanoudakis, 2007] proposes 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.

To conclude, we can state that none of the research approaches provide a unitary and complete solution for the development of pervasive autonomic systems. In this paper we try to overcome this deficiency by: (i) defining a generic context model that can be used to accurately represent general purpose real contexts, (ii) enhancing the context model with new concepts that allow specifying the self-* autonomic properties, (iii) defining the context model self-configuring property and (iv) proposing a self**-**configuring algorithm used to monitor and evaluate the environment changes in order to keep updated the context model artifacts.

The rest of the paper is organized as follows: in Section 2, the context model and its main elements are presented; in Section 3 we detail the context model management processes; Section 4 introduces the grounding concepts used to enhance the context model with autonomic features; Sections 5 defines the context model self-configuring property and presents a generic self-configuring algorithm*;* Sections 6 shows how the self-configuring context model is used to represent and manage context representation of our intelligent laboratory environment [DSRL], while Section 7 concludes the paper and shows the future work.

2. The context model

Let's consider a pervasive system used to guide the tourists into a museum. The museum is an intelligent space where the visitors are identified by RFID readers, while their location and orientation is determined using a sensor network. The tourists can interact with the pervasive system if they have a wireless capable PDA 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, the loud limits, etc.

By studying and analyzing similar real world relevant scenarios we define the *context model* as a triple:

$$C =< R, A, P > \quad (1)$$

where: R is a set of context resources, A is a set of actors which interact with context resources and P is a set of real context related policies. In our model the *context abstraction* is represented by the set of all context properties in terms of relevant information provided by context resources.

In order to provide an accurate representation of the real world context, the following context representation artifacts are defined: *specific context model*, *specific context model instance* and *context – actor instance*.

The context model is mapped onto different real contexts by populating the sets with real context specific elements. The mapping result is a *specific context model* that is defined as follows:

$$C_S =< R_S, A_S, P_S > \quad (2)$$

Using the above presented scenario, we have identified the following context specific elements (see Figure 1) that populate the context model sets: (i) the context resource set contains the tourist attached resources such as PDA or RFID tags and the intelligent museum resources such as location sensors; (ii) the set of context actors contains the tourists and the executable context aware applications; (iii) the set of context policies contains the constraints used to drive the tourist - museum interaction such as the minimum distance to the artifacts or loud limits.

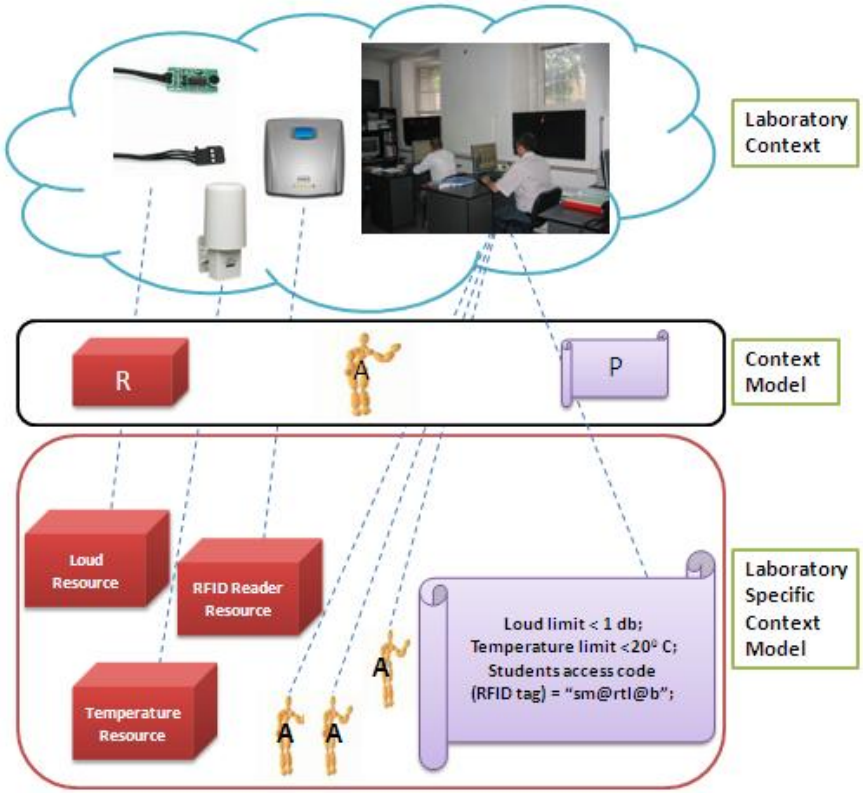

Figure 1. Context model elements

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. The specific context model represents the context situation to which a context-aware application must adapt.

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. A context–actor instance represents the projection of the specific context model instance onto a certain actor.

The relationships between the context model sets can be modeled in a general purpose context ontology core (see Figure 2). The domain specific concepts are represented as sub trees of the core ontology by using is-a type relations. A system context situation is represented by the core ontology together with the domain specific concepts sub trees and their instances in a specific moment of time. The two ways of representing the context (set based and ontology based) are equivalent and need to be kept synchronized. The set based context model 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 (self-configuring, self-healing, self-optimizing and self-protection). The ontology based model will be used by the context aware applications for reasoning and learning. The following sections detail the context model main concepts

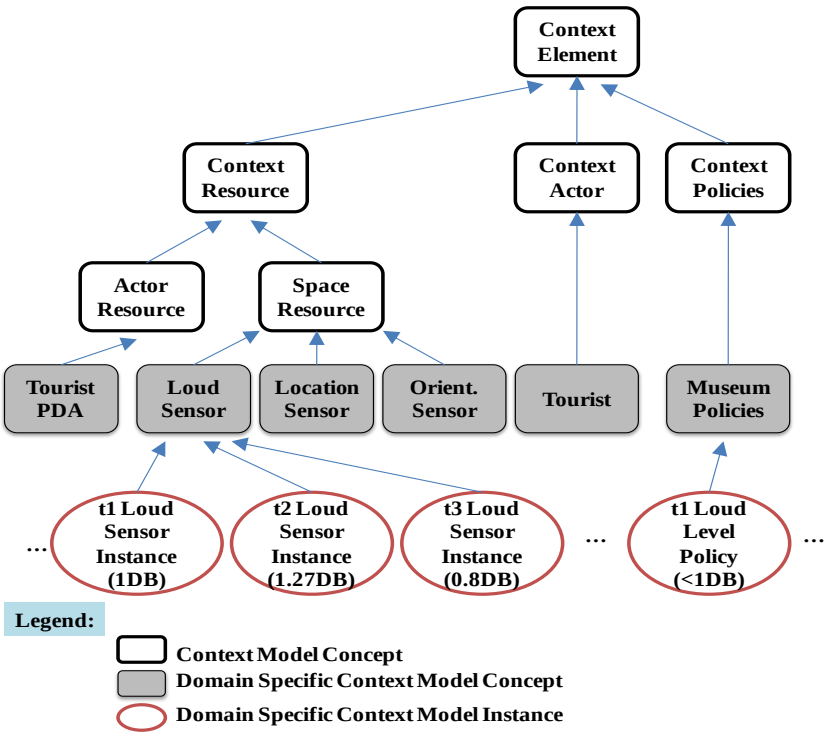

Figure 2. The context model ontology representation

.

2.1. Resources

A context resource is a physical or virtual entity which generates and / or processes context information. In a real context, we have identified passive and active resources. The *passive context resources* such as sensors, aim at capturing and storing context specific data while the *active context resources* such as actuators, can interact directly with the context and modify the context state.

The set of context resources R can be separated in two disjunctive subsets: (i) the set of context resources attached to the physical space / environment R_S in which actor-context interactions occur and (ii) the set of context resources attached to the actors R_A that provide information related to actor-context interactions:

$$R = R_A \cup R_S \tag{3}$$

A context resource has a unique identity, can be annotated with semantic information and is characterized by its *properties*, *services* and *influence zone*. Resource Properties, $K(r)$, specify the set of relevant context information provided by the resource. For example, $K(PDA)$= {Bluetooh, Wireless} . Resource Services, $S(r)$, specifies the resource functionality as a set of services (for example a service that locates / updates an object). The resource services are exposed by publishing them in a public registry (UDDI). The actors interact with a context resource through its attached services. Resource Influence Zone, $Z(r)$, represents the 3D physical space in which a resource captures / provides context information or in which the resource presence can be sensed (in other words, it becomes visible for an actor or for another resource). The influence zone of a context resource attached to an actor is the zero volume space: $\forall r \in R_A \Rightarrow Z(r) = 0_V$. The influence zone for a context resource that is attached to the physical space is a non-zero volume space: $\forall r \in R_S \Rightarrow Z(r) \neq 0_V$. A physical space resource is considered an immobile resource so the influence zone is specified by using the resource position in the real space and the resource range.

2.2. Actors

An actor represents a physical or virtual entity that interacts directly with the context or uses the context resources to fulfill its needs. The actor is a context information generator, has a unique identity and can be annotated with semantic information. An actor is characterized by its: (i) specific resources, (ii) context related requirements and (iii) actor-context contract. Actor

Resources, R_a, specifies the set of actor associated resources such as position elements, RFID tags and / or augmented reality elements. Context Related Requirements, $Req(a)$, specifies actor context related preferences. Context Contract, $Ctr(a, C_S)$, stipulates the actor's rights and responsibilities within a specific context.

2.3. Policies

A policy represents a set of rules that must be followed by the actors or resources located in the context influence zone. The process of evaluating the policy complying degree is calculated by considering the complying degree of all policy rules. If a rule is broken an exception is thrown determining the elimination from the context of the actor or the resource that committed the fault. According to the context resources classification passive context resources and active context resources we have defined two types of context policies: metric constraints policies and action policies. The metric constraints policies are defined for the set of passive context resources in order to impose some restrictions to the captured context specific data. The context aware application needs to automatically determine what actions or plans of actions should be executed in order to enforce and maintain these constraints. The action policies are defined for the context elements that can directly modify the context state (active resources or actors) and specify the actions that should be performed to satisfy the policy constraints.

3. The context model management

The context model management infrastructure layer is based on four types of intelligent, cooperative BDI type agents [Rao, 1995]: *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 specific artifacts with the system execution environment. This agent is also responsible for negotiating 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" for the pervasive application. The Request Processing Agent (RPA) processes the actor requests. This agent identifies and generates the action plans that must be executed for serving an incoming request. The RPA agent uses the specific context model instance 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 the RPA agent and executes every plan action using the available services. After mapping action plans onto services, a plan orchestration (smart workflow) which can be executed using transactional principles is obtained.

The context management infrastructure agents are implemented using the Java Agent DEvelopment Framework platform [JADE]. Figure 3 shows how the four context management infrastructure agents communicate and coordinate their actions in order to manage the context representation process.

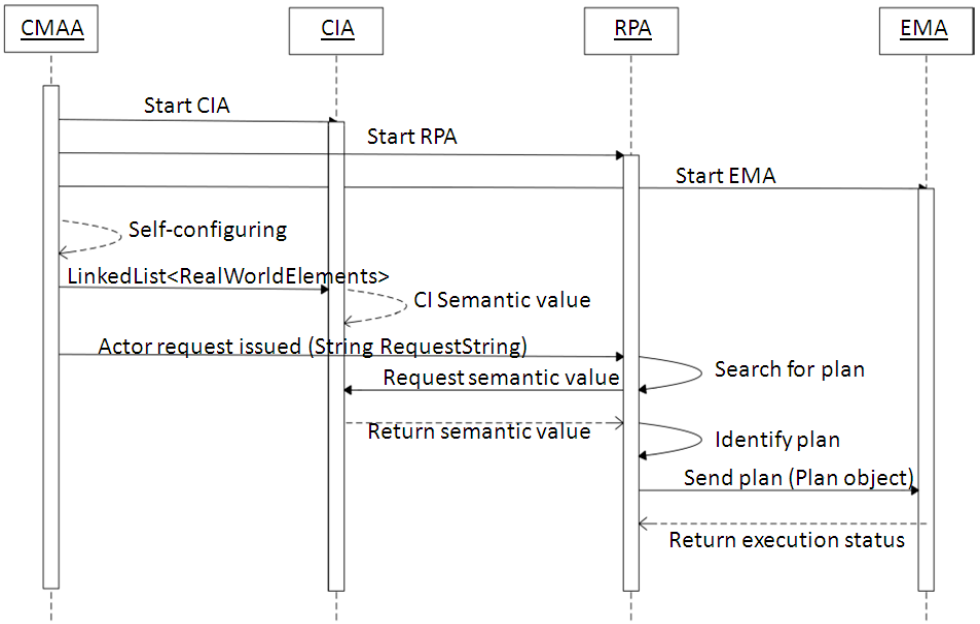

Figure 3. The cooperative agents intercommunication

When the middleware is deployed, CMAA is the first running agent. It instantiates the CIA, RPA and EMA context management agents and sends them the real world context representation. Also CMAA agent keeps the context model synchronous with the real world context by executing the self-configuring algorithm presented in section 5.4. CIA agent uses reasoning and learning algorithms and the context model ontological representation provided by CMAA agent to infer high level context information. When a context actor issues a request, the RPA agent is activated. It uses the context information provided by CIA agent, to construct / find a plan that can be executed as a response to the actor's request. The plan is then forwarded for execution to the EMA agent.

4. Towards an autonomic context model

To enhance the basic context model with autonomic capabilities we have introduced three new concepts: *isotropic context space*, *context granule* and *context model entropy*. Each of these concepts is discussed below.

4.1. Isotropic context space

A context sub-space (part of the whole context space) is *isotropic* if and only if the set of sub-space attached resources R_S is invariant to the movements of all actors in the context sub-space. In other words, in an isotropic context sub-space, the R_S set is the same for all the actors that are physically located in sub-space influence zone. It should be noted that if $Card(R_S)=1$, the context sub-space is isotropic. From now on, the context sub-space will be also considered and referred as a context space. Given a non-isotropic context space, the *variation degree of the space isotropy* Δ_{IZ} is defined as the variation of the R_S set, while the actor moves in the context space.

4.2. Context granule

Usually, a context space is non-isotropic but it can be split in a set of disjunctive isotropic context space volumes. We define the *Context Granule* (GC) as the maximum volume of a context space where the space isotropy degree variation is zero: $\Delta_{IZGC\text{-}GC}=\varnothing$. In a given moment of time, an actor can be physically located in a single context granule. As a result, Δ_{IZ} is non-zero only when an actor moves between context granules. Considering two context granules GC1 and GC2, the space isotropy degree variation is determined as:

$$\Delta_{IZ_{GC1\text{-}GC2}} = \{R_{GC1} \setminus R_{GC2}\} \cup \{R_{GC2} \setminus R_{GC1}\} \quad (4)$$

If $\Delta_{IZ_{GC1\text{-}GC2}}=\varnothing \Rightarrow R_{GC1}=R_{GC2}$, the actor remains in the same context granule.

4.3. Specific context model entropy

$E(C_S)$ is defined as the specific context model entropy (the level of disorder) reflecting the degree of fulfilling the context policies ($E(C_S)=0$, all context policies are respected).

If Rf, is a function over the policy rules that evaluates whether a certain rule is broken or not and Pf measures the policy fulfilling degree then the entropy is defined as:

$$E : C_S \to \mathbb{Z},\ E(C_S) = \sum_{i>0} \sum_{j>0} Pf(Rf(rule_{ij})) \quad (5)$$

The entropy $E(C_S)$ is used to globally determine the autonomic capabilities of the specific context model:

$$E(C_S) = 0 \Rightarrow C_S \text{ is in an autonomic state} \quad (6)$$

$$E(C_S) > 0 \Rightarrow C_S \text{ is in a non-autonomic state} \quad (7)$$

$$E^t(C_S) * E^{t+1}(C_S) = 0 \quad (8)$$

A specific context model features autonomic behavior if the autonomy invariant (relation 8) is always true.

5. The self-configuring algorithm

In order to provide an efficient context information management, we enhanced the context model with self-configuring properties. The self – configuring property is enforced by monitoring the real world context in order to detect context variations or conditions for which the context artifacts must be updated. We have identified three causes that generate context variation: (1) adding or removing context elements (resources, actors, policies) to / from the real world context, (2) the actors' mobility within the real world 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 these context variation causes targeting to determine (i) the context variation degree and (ii) the starting condition of the self-configuring process.

5.1. Context variation generated by 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 the CMMA agent which creates a new specific context model adapted to the new real world context.

Next, we evaluate the context variation degree generated by context resources (Δ_R), context policies (Δ_P) and context actors (Δ_A) using a set of associated thresholds T_R, T_P and T_A respectively.

The context resources set variation is generated by adding or removing a context resource r (sensor or actuator) to / from the pervasive application execution environment. The context resource set variation is calculated using the set difference operation applied in two consecutive moments of time t and $t+1$ ($t+1$ represents the moment when the resource r becomes available). The same reasoning can be applied when the resource r fails or becomes unavailable:

$$\Delta_R = \{R_S^{t+1} \setminus R_S^t\} \cup \{R_S^t \setminus R_S^{t+1}\} \quad (8)$$

In relation 8 $R_S^{t+1} \setminus R_S^t$ contains the set of context resources that become available and $R_S^t \setminus R_S^{t+1}$ contains the set of context resources that become unavailable. If $Card(\Delta_R) \geq T_R$ a new specific context model is generated by adding or removing the context resources contained in Δ_R.

The policy set variation is determined by adding, removing or updating an execution environment policy. The update operation is always achieved by removing the old context policy followed by adding a new one. Using the same assumptions and conclusions as for context resources, the policy set variation is defined below:

$$\Delta_P = \{P^{t+1} \setminus P^t\} \cup \{P^t \setminus P^{t+1}\} \quad (9)$$

The actors set variation is generated by the actors that enter or leave the execution context. Each context actor has an attached context resources set during its context interactions. In a given context, an actor is characterized by a large number of actor-context interaction patterns, but only two of these patterns determine a variation of the actor context resources set R_A: (i) the actor enters the context and (ii) the actor leaves the context. The actors related context variation is:

$$\Delta_A = \{A^{t+1} \setminus A^t\} \cup \{A^t \setminus A^{t+1}\} \cup \{R_A^{t+1} \setminus R_A^t\} \cup \{R_A^t \setminus R_A^{t+1}\} \quad (10)$$

Overall, the real world context variation Δ_{ENV} is given by the union of all context elements' variation as shown below:

$$\Delta_{ENV} = \Delta_R \cup \Delta_A \cup \Delta_P \quad (11)$$

$$Card(\Delta_{ENV}) = Card(\Delta_R) + Card(\Delta_A) + Card(\Delta_P) \quad (12)$$

The self-configuring threshold is defined as:

$$T_{Self-Configuring} = \min(T_R, T_A, T_P) \quad (13)$$

The CMMA agent should start the execution of the self-configuring process and generate a new specific context model when $Card(\Delta_{ENV}) \geq T_{Self-Configuring}$.

5.2. Context variation generated by actor's mobility

Due to their mobility, the actors are changing their environment location and implicitly the set of resources with which they interact. The CMMA agent 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. The CMMA agent continuously monitors the actors' movement in the real world context and periodically evaluates the space isotropy variation. If for an actor, the space isotropy variation is a non empty set ($\Delta_{IZ} \neq \varnothing$) then the self-configuring process executed by the CMMA agent generates a new context – actor instance. It actually represents the specific context model instance projection onto a certain actor:

$$CI_a^{t+1} =< R_a^{t+1}, a, P^{t+1} > \ , \ R_a^{t+1} = R_{GC}^{t+1} \tag{14}$$

The context variation generated by all actors' mobility in a context space is given by the union of space isotropy degree variation in a certain moment of time for each actor:

$$\Delta_{CAM} = \bigcup_{a \in A} \Delta_{IZa} \tag{15}$$

5.3. Context variation generated by changes of resources property values

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:

$$K_{val}(R) = \{(k_1, val_1), ..., (k_n, val_n)\}, k_1, ..., k_n \in K \tag{16}$$

If the values captured by the Hot&Humidity sensor in a moment of time are for temperature 5 degree Celsius and for humidity 60%, then $K_{val}(\text{Hot\&HumiditySensor}) = \{(\text{Temperature}, 5), (\text{Humidity}, 60\%)\}$. CMAA agent calculates the context variation generated by changes of resource properties' values Δ_{RPV} as presented in relation 17.

$$\Delta_{RPV} = K_{val}(R^{t+1}) - K_{val}(R^t) = \{(k_1, val_1^{t+1} - val_1^t), ..., (k_n, val_n^{t+1} - val_n^t)\}, k_1, ..., k_n \in K \tag{17}$$

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 $Card(\Delta_{RPV}) \geq 0$.

5.4. The self-configuring algorithm

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

Algorithm CMAA_Self_Configuring
input: (1) new real world context elements: $\mathbf{R^n, A^n, P^n}$
(2) thresholds for context elements variation: $\mathbf{T_R, T_A, T_P}$
output: new context artifacts: $\mathbf{< C_S^n , CI_a^n , C_{SI}^n >}$
resources: current context artifacts set representation: $< \mathbf{C_S, CI_a , C_{SI}} >$
current context artifacts ontology representation: **owlModel**
begin
// CMAA evaluates the context variation
$\Delta_R = \{R_S^n \setminus R_S\} \cup \{R_S \setminus R_S^n\}$
$\Delta_A = \{A^n \setminus A_S\} \cup \{A_S \setminus A^n\} \cup \{R_A^n \setminus R_A\} \cup \{R_A \setminus R_A^n\}$
$\Delta_P = \{P^n \setminus P_S\} \cup \{P^n \setminus P_S\}$
$\Delta_{CAM} = \cup_{a \in A} \Delta_{IZa}$
$\Delta_{RPV} = K_{val}(R^n) - K_{val}(R)$

$T_{Self\text{-}Conf} = \min(T_R, T_A, T_P)$
if (Card (Δ_{ENV}) ≥ $T_{Self\text{-}Conf}$)
//CMAA tries to create a new specific context model
if (Card (Δ_R) ≥ 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 (Δ_A) ≥ 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 (Δ_P) ≥ 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
$C_S^n = C_S - \Delta_P = (R_S, A_S, P_S) - \Delta_P = (R_S, A_S, P_S \setminus \Delta_P)$
else
// CMAA tries to create a new context-actor instance
$T_{Self\text{-}Conf} = 0$
if (Card (Δ_{CAM}) > $T_{Self\text{-}Conf}$)
foreach a € A
if ($\Delta_{IZ\ a} \neq 0$) $CI_a^n = \langle R_a, a, P \rangle$
else
// CMAA tries to create a new specific context model instance
if (Card (Δ_{RPV}) > $T_{Self\text{-}Conf}$)
$C_{SI}^n = \langle R_a, a, P \rangle$

updateOntology (owlModel, Δ_R, Δ_A, Δ_P)
return < C_S^n , CI_a^n , C_{SI}^n >
end

Figure 4. The Self-Configuring algorithm

6. Case Study

For the case study we have used a intelligent closed environment represented by our Distributed System Research Laboratory. In the laboratory the 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 ambient information like the temperature or humidity.

The DSRL infrastructure contains a set of sensors through which the real context information is collected: two Hot&Humidity sensors that capture the air humidity and the temperature, four Orient 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 [Infusion]. The middleware is deployed on an IBM Blade-based technology Server Center. The IBM Blade technology was chosen because its maintenance software offers autonomic features like self-configuring of its hardware resources. The CMMA 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 5.4.

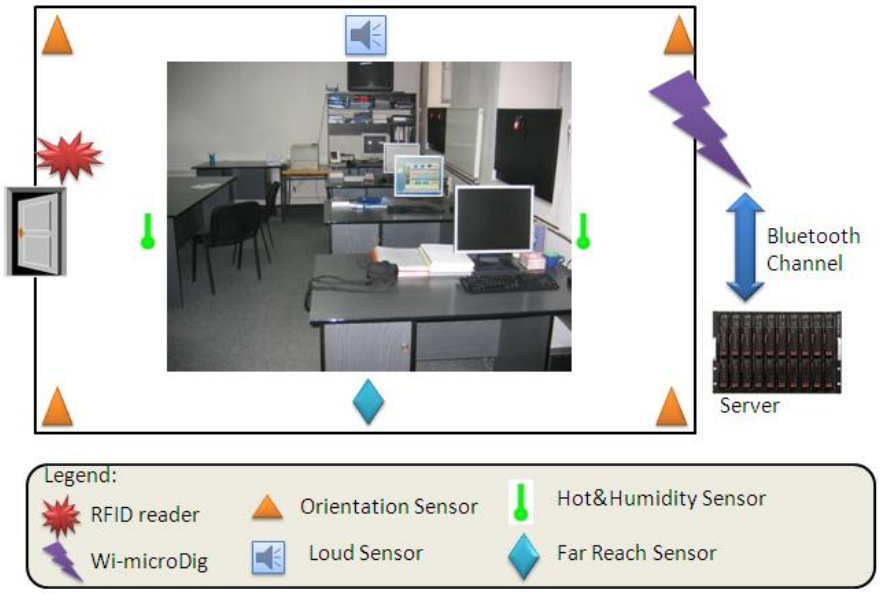

Figure 5. The DSRL infrastructure

When the middleware is deployed and starts execution ($t = 0$) there are no context model artifacts constructed (the R, A, P sets of the context model are empty). After one second ($t = 1$), when two students John and Mary enter the lab, the Context Model Administering Agent receives the updated context information from the Context Acquisition Layer and calculates the context elements variation Δ_R , Δ_P and Δ_A as shown 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 $t = 1$, the CMMA agent 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 Δ_R , Δ_P and Δ_A calculated in Figure 6.

R_S^1 = {FarReachSensor, RFIDReader, HotHumiditySensor1&2, LoudSensor, OrientationSensor1&2&3&4}
$R_S^0 = \emptyset$
$\Delta_R = (R_S^1 \setminus R_S^0) \cup (R_S^0 \setminus R_S^1)$
Δ_R = {FarReachSensor, RFIDReader, HotHumiditySensor1&2, LoudSensor, OrientationSensor1&2&3&4}

A^1 = {StudentJohn, StudentMary}
$A^0 = \emptyset$
$\Delta_A = (A^1 \setminus A^0) \cup (A^0 \setminus A^1)$
Δ_A = {StudentJohn, StudentMary}

P^1 ={LoudLimit, TemperatureLimit}
$P^0 = \emptyset$
$\Delta_P = (P^1 \setminus P^0) \cup (P^0 \setminus P^1)$
Δ_P = {LoudLimit, TemperatureLimit}

$Card(\Delta_{ENV}) = Card(\Delta_R) + Card(\Delta_A) + Card(\Delta_P) = 13$
$Card(\Delta_{ENV}) > T_{Self-Configuring}$

Figure 6. DSRL context variation at t=1

The CMMA agent dynamically updates / populates the context model artifacts ontology by invoking ontology management methods defined by the Protégé-OWL API. In order to test the middleware self-configuring capabilities we have considered that after 60 seconds the following context changes occurred: (i) student John leaves the laboratory, (ii) OrientationSensor1 and OrientationSensor4 are disabled and (iii) LoudSensor is disabled. The CMAA agent calculates the variation in the new context at $t = 61$(Figure 7) and executes the self-configuring algorithm.

R_S^{61} = {FarReachSensor, RFIDReader, HotHumiditySensor1&2, OrientationSensor2&3}
R_S^{60} = {FarReachSensor, RFIDReader, HotHumiditySensor1&2, OrientationSensor1&2&3&4, LoudSensor}
$\Delta_R = (R_S^{61} \setminus R_S^{60}) \cup (R_S^{60} \setminus R_S^{61})$
Δ_R = {LoudSensor, OrientationSensor1&4}

A^{61} = {StudentMary}
A^{60} = {StudentJohn, StudentMary}
$\Delta_A = (A^{61} \setminus A^{60}) \cup (A^{60} \setminus A^{61})$
Δ_A = {StudentMary}

P^{61} = {LoudLimit, TemperatureLimit}
P^{60} = {LoudLimit, TemperatureLimit}
$\Delta_P = (P^{61} \setminus P^{60}) \cup (P^{60} \setminus P^{61})$
$\Delta_P = \emptyset$

$Card(\Delta_{ENV}) = Card(\Delta_R) + Card(\Delta_A) + Card(\Delta_P) = 4$
$Card(\Delta_{ENV}) > T_{Self\text{-}Configuring}$

Figure 7. The DSRL context variation at t=61

7. Conclusions and future work

This paper addresses the problem of representing and managing the context information in a reliable and fault tolerant manner, targeting the development of context aware autonomic systems. In order to achieve our goal we have defined a self-configuring context model that accurately captures and represents general purpose real contexts, in a programmatic manner. The proposed context model uses two equivalent and synchronized ways of representing the context: a set based representation and an ontology based representation. The set based representation is used to evaluate the conditions under which the self-* processes should be executed in order to enforce the autonomic properties. The self-configuring property is enforced by monitoring the execution context in order to detect context variations or conditions for which the context artifacts must be constructed or updated. The proposed context model was tested and validated using our Distributed Systems Research Laboratory as a smart space infrastructure. For future development, we intend to enhance the pervasive middleware with new self-* capabilities like self-healing and self-optimizing.

8. References

Anca Rarau, K. Pusztai. I.Salomie (2006). MultiFacet Item based Context-Aware Applications. *International Journal of Computing and Information Sciences*, 10-18

Irene Y.L. Chen, Stephen J.H. Yang, Jia Zhang (2006). Ubiquitous Provision of Context Aware Web Services, *Proc. of the IEEE International Conference on Services Computing*, 60-68.

Ki-Chul Lee, Jung-Hoon Kim, Jee-Hyong Lee (2007). Implementation of Ontology Based Context-Awareness Framework for Ubiquitous Environment, *Proc. of the Int. Conference on Multimedia and Ubiquitous Engineering*, 278-282.

Feruzan Ay, (2007). Context Modeling and Reasoning using Ontologies, University of Technology Berlin.

Neil O'Connor, Raymond Cunningham, Vinny Cahill (2007). Self-Adapting Context Definition, *Proc. of the First International Conference on Self-Adaptive and Self-Organizing Systems*, 336-339

Marcel Cremene, Michel Riveill (2007). Autonomic Adaptation based on Service-Context Adequacy Determination, *Electronic Notes in Theoretical Computer Science, Elsevier*, 35-50

Brecht Desmet, Jorge Vallejos (2007). Layered design approach for context-aware systems, *Proc. Of the 1st Int Workshop on Variability Modelling of Software-Intensive Syst.*, Ireland, 157-165.

Luiz Silva Santos, Fano Ramparany, Patricia Dockhorn (2007). A Service Architecture for Context Awareness and Reaction Provisioning, *Proc of the IEEE Congress on Services*, 25-32.

Frederick Seyler, Chantal Taconet, Guy Bernard (2007). Context Aware Orchestration Meta-Model, *Proc. of the 3rd Int Conference on Autonomic and Autonomous Systems*, 351-356.

Paolo Bellavista, Antonio Corradi, Rebecca Montanari (2006). Mobile Computing Middleware for Location and Context-Aware Internet Data Services. *ACM Transactions on Internet Technology*, Vol. 6, No. 4, 356-380.

Damien Fournier, Sonia Ben Mokhtar, Nikolaos Georgantas (2006). Towards Ad hoc Contextual Services for Pervasive Computing. *IEEE Middleware for Service Oriented Computing*, Melbourne, Australia, 36-41.

George Spanoudakis, Khaled Mahbub (2007). A Platform for Context Aware Runtime Web Service Discovery, *IEEE International Conference on Web Services,* 233-240.

DSRL - Distributed Systems Research Laboratory. Technical University of Cluj-Napoca, http://dsrl.coned.utcluj.ro/

Anand S. Rao, Michael P. Georgeff (1995). BDI Agents: from Theory to Practice, *Proceedings of the First International Conference on Multiagent Systems*, 312-319.

JADE - Java Agent DEvelopment Framework, http://jade.tilab.com/

Infusion Systems Ltd, http://www.infusionsystems.com

Author biographies

Tudor Cioara is an Assistant Professor and PhD Student at the Computer Science Department of the Technical University of Cluj-Napoca having as main research areas Context aware systems, Learning/Reasoning techniques and Pervasive systems. He was involved in four national funded research projects. He obtained the postgraduate studies degree in Distributed Computer Systems in 2008. Tudor is currently developing the PhD thesis "Towards Context Aware Pervasive Systems". He also co-authored a book on Distributed Computing and Systems.

Ionut Anghel is an Assistant Professor and PhD Student at the Computer Science Department of the Technical University of Cluj-Napoca. His main research areas are Autonomic systems, Mobile agents and Ontology learning. Ionut was involved in several national funded research projects co-authored a book on Distributed Computing and Systems. He obtained the master degree in Distributed Computer Systems in 2008 and is currently developing the PhD thesis "Autonomic Computing Middleware".

Professor Ioan Salomie is the head of the Distributed Systems Research Laboratory (DSRL). His main research interests include Distributed Systems, Context Awareness, Autonomic Systems and Intelligent Systems. He has an extensive international experience in both teaching and research being Invited Professor at the Electronic and Computer Engineering Department, University of Limerick, Ireland (2001-2004) and Loyola College in Maryland, USA (1996). He was also Research Fellow at the University of Nottingham, UK (1993), External Examiner for the University of Limerick, Ireland (2007-2011) and member of PCs in international conferences. Professor Salomie leaded several national research projects and one international EU project.

Dr. Mihaela Dinsoreanu received her PhD based on her work in the field of agent-oriented software engineering. Her main research interests include Knowledge Engineering, Intelligent Systems, Business Intelligence. She was involved in several national research projects and two international projects.