KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2009
Cluj-Napoca (Romania), July 2–4, 2009, pp. 1–8

A SELF-CONFIGURING MIDDLEWARE FOR DEVELOPING CONTEXT AWARE APPLICATIONS

TUDOR CIOARA[(1)], IONUT ANGHEL, IOAN SALOMIE, MIHAELA DINSOREANU, AND ANCA RARAU

Abstract. This paper introduces a self-configuring middleware that manages the context information acquisition and representation processes targeting the development of context aware applications. The context information is represented in a programmatic manner, readable for the applications build on top of the middleware, using the RAP context model. An agent based context model management infrastructure generates and administrates the context artifacts at run time. 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.

1. Introduction related work

The context aware computing refers to the ability of a software application to detect and respond to changes in their execution environment. An important challenge in developing context aware applications is the complexity and dynamic nature of their execution environment which makes the application management and adaptation processes extremely difficult tasks. Also, the resources variety may influence the execution of the context aware application thus implying complex context information acquisition and representation processes. 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. The context aware application cannot wait indefinitely for an answer from a temporary unavailable context resource and many times the payoff for not taking into consideration the new available context resources can be very high.

From our perspective the solution for these problems is to use the self-* autonomic paradigms in the development and integration of self-* enhanced components into context aware applications. The goal is to introduce some degree of

2000 *Mathematics Subject Classification.* 68Q85, 68M14, 68T05, 68W15.
Key words and phrases. Self-Configuring, Context Awareness, Autonomic Computing.

autonomy for the context acquisition and representation processes. In this paper we address the problems of monitoring, capturing and representing the context information by proposing and developing a pervasive self-configuring middleware that manages smart environments using an agent based context management infrastructure. The smart environment context information is represented in a programmatic manner using the RAP context model [10]. The context model management infrastructure is implemented by BDI (Believe, Desire, Intentions) agents [12] that generate and administrate the context model artifacts at run time. The middleware self-configuring feature is implemented by monitoring and evaluating environment changes in order to keep updated the context artifacts. The middleware was tested in our Distributed Systems Research Laboratory (DSRL) [4].

During the pervasive middleware development process we have identified the context representation and the context management as the 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 [5]. Key-Value models represent context information using a set of attributes and their associated values [1]. Markup models enable structuring context information into a hierarchy. Tags describe context attributes and their associated values [9]. Object models structure context into object classes and their implicit relationships [6]. 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 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 [8]. 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. We solve this problem by defining an agent based solution that uses the RAP model set based 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 [2], models for capturing and updating the context information based on the information type are proposed, while in [5] reusable components which provide stable communication channels for capturing and controlling context specific data are defined for updating the context specific data. In [11], Spanoudakis 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. The main disadvantage these approaches is the lack of efficiency in the context model management process which is rather static and difficult to adapt to context changes. We overcame this problems by defining and using the self-* autonomic paradigms for the context acquisition and representation processes. 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 [7]. Their objective is to associate a certain degree of intelligence to the computational systems for context adaptation. In [3], 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 the system knowledge about the context. This description is used by the system to take reconfiguring and adapting decisions.

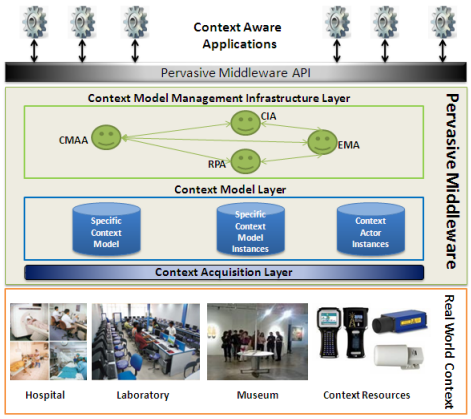
2. The pervasive middleware

The pervasive middleware conceptual architecture (see Figure 1a) defines three main layers: (i) the acquisition layer that captures the context information from real world contexts, (ii) the context model layer which represents the context information in a machine interpretable way and (iii) the context model management infrastructure layer that manages the context representation.

The context information acquisition layer design takes into consideration the following aspects: (i) the sensor information retrieval mechanism and (ii) the visibility of the sensor information to middleware upper layers. From the middleware perspective we have defined both push and pull types of sensor information retrieval mechanisms. The push mechanism uses event listeners to gather the context information from sensors while the pull mechanism uses a query based approach which allows the context information to be provided on demand.

The context representation layer uses the RAP context model to represent a real world context in a programmatic manner. This model defines two types of context information representations: (i) set based, used to determine the conditions under which the middleware executes the self-configuring algorithm and (ii) ontology based, used to infer new context related information by means of reasoning and learning algorithms. In the set based approach the context information is modeled 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. In the ontology based representation the relationships between the context model sets are modeled in a general purpose context ontology core. 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. In order to



a)

Algorithm CMAA_Self_Configuring
input: (1) new real world context elements: $\mathbf{R^n}$, $\mathbf{A^n}$, $\mathbf{P^n}$
(2) thresholds for context elements variation: $\mathbf{T_R}$, $\mathbf{T_A}$, $\mathbf{T_P}$
output: new context artifacts $\mathbf{C_S^n}$, $\mathbf{CI_a^n}$, $\mathbf{C_{SI}^n}$
resources: current context artifacts set $\mathbf{C_S}$, $\mathbf{CI_a}$, $\mathbf{C_{SI}}$
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 (ΔENV) $\geq T_{Self\text{-}Conf}$)
begin //CMAA tries to create a new specific context model
if (Card (ΔR) $\geq T_R$)
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 (ΔA) $\geq T_A$)
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 (ΔP) $\geq T_P$)
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 ($U_{a \in A} \Delta IZ_a$) $> T_{Self\text{-}Conf}$)
foreach $a \in A$ **if** ($\Delta IZ_a \neq 0$) $CI_a^n = \langle R_a, a, P \rangle$
else // CMAA tries to create a new specific context instance
if (Card (ΔRPV) $> T_{Self\text{-}Conf}$) $C_{SI}^n = \langle R_a, a, P \rangle$
end
updateOntology (ΔR, ΔA, ΔP)
end

b)

FIGURE 1. a) The pervasive middleware conceptual architecture and b) The middleware self-configuring algorithm

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 *specific context model* is obtained by mapping the context model onto different real contexts and populating the sets with real context specific elements. A *specific context model instance* contains the set of context resources with which the middleware interacts, together with their values in a specific moment of time t. The specific context model represents the context situation to which a pervasive application built onto the middleware must adapt. The *context-actor instance* contains the set of context resources with which the actor can interact, together with their values in a specific moment of time.

The context model management infrastructure layer is based on four types of intelligent, cooperative BDI type agents: Context Model Administering Agent,

Context Interpreting Agent, Request Processing Agent and Execution and Monitoring Agent. 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. The *Context Interpreting Agent (CIA)* semantically evaluates the information of a context instance and tries to find the context instance "meaning" for the context aware application. The *Request Processing Agent (RPA)* processes the actor requests. It identifies and generates the action plans that must be executed for serving an incoming request. The *Execution and Monitoring Agent (EMA)* processes the plans received from the RPA agent and executes every plan action using the available services.

3. The middleware self-configuring algorithm

At middleware level the self-configuring feature is implemented by monitoring the real world context in order to detect the context variations for which the context artifacts need to be updated and synchronized. We have identified three causes that generate context variation: (1) adding or removing context sources (resources, actors, policies) to / from the real world context, (2) actors' mobility within the real world context and (3) changes of the resources property values.

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 CMAA 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 ΔR in relationship with its associated threshold T_R . The same reasoning is used to determine the variation related to the context policies ΔP and the context actors ΔA with their thresholds T_P and T_A. The context resources set variation ΔR is generated by adding or removing a context resource r 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 $t+1$, where $t+1$ represents the moment when the resource r became available:

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

If Card(ΔR) $\geq T_R$ a new specific context model is generated by adding or removing the context resources contained in ΔR. The overall real world context variation ΔENV is given by the union of all context elements' variation:

$$\Delta ENV = \Delta R \cup \Delta A \cup \Delta P \quad (2)$$

The self-configuring threshold is defined as: $T_{Self-Configuring} = \min(T_R, T_A, T_P)$. The CMMA agent should start the execution of the self-configuring process and generate a new specific context model when Card(ΔENV) $\geq T_{Self-Configuring}$.

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. 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 [10]. 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. The space isotropy variation ΔIZ is non-zero only when an actor a moves between two context granules. If for an actor $\Delta IZ_a \neq \emptyset$, then the self-configuring process executed by CMMA generates a new context-actor instance.

Context variation generated by changes of resources property values. A context resource is a physical or virtual entity which generates and / or processes context information. In order to evaluate the context variation generated by the changes in the resource property values, we define a function K_{val} that associates a resource property to its value. CMAA calculates the context variation generated by changes of resource properties' values ΔRPV as presented in (3) and creates a new specific context model instance when Card(ΔRPV) ≥ 0.

$$\Delta RPV = K_{val}(R^{t+1}) - K_{val}(R^t) = \{(k_1, val_1^{t+1} - val_1^t), \ldots, (k_n, val_n^{t+1} - val_n^t)\} \quad (3)$$

The self-configuring algorithm is executed by CMAA in order to keep the context model artifacts synchronized with the real context (see Figure 1b). CMAA features a ticker based behavior and periodically evaluating the context changes. When a significant context variation is determined, the context model artifacts are updated.

4. Results

For the case study we have used the smart 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, etc.). A sensor network captures information regarding students' location or orientation and also ambient information like the temperature or humidity. The middleware is deployed on an IBM Blade-based technology Server Center.

In order to test 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. To make the simulated sensor information visible, in an independent manner, to the upper layers, we have used the web services technology. The results show that the self-configuring algorithm implemented by CMAA can synchronize and update the context model artifacts in a reasonable time for up to 200 sensors that change their values simultaneously (see Figure 2). 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. To test the self-configuring algorithm capacity

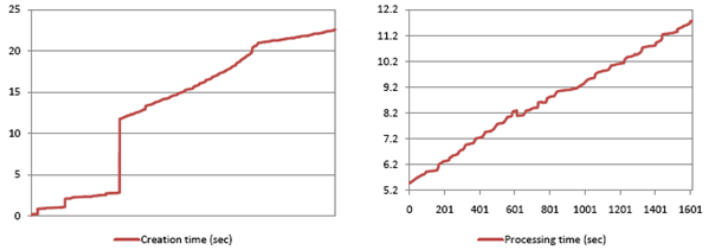


FIGURE 2. The self-configuring algorithm scalability results

to detect the temporary unavailable and the new available context resource we simulate the sensors using a web service that returns random numbers and a small program that periodically inserts new entries in a smart environment description file. The creation time was measured from the moment in which CMAA observes the new sensor in the smart environment description file until all the changes were processed, (the new context artifacts were generated). The results show that the self-configuring algorithm can successfully administrate up to 10 sensors that simultaneously become available in the DSRL smart environment (see Figure 2).

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 CMAA executes 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

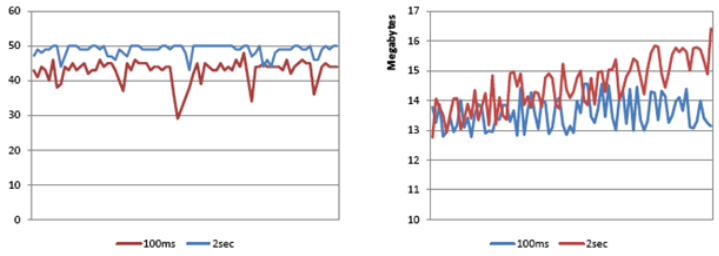


FIGURE 3. The CMAA self-configuring algorithm CPU and memory overloading with 100 sensors at t1=100 ms and t2=2000 ms

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 3).

5. Conclusions

In this paper we have provided a solution to the problem of managing the smart 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 an agent based context management infrastructure, 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. The results show that the proposed solution is viable from the perspective of scalability and processor / memory consumption. For future developments we intend to define a generic formalism for all self-* paradigms in order to enhance the proposed middleware with complete autonomic capabilities targeting run-time context self-adaptation.

References

[1] K. M. Anderson, F. A. Hansen, and N. O. Bouvin. Templates and queries in contextual hypermedia. In *Proc. of the 17th conf. on Hypertext and hypermedia*, page 99110, 2006.
[2] P. Bellavista, A. Corradi, and R. Montanari. Mobile computing middleware for location and context-aware internet data services. In *ACM Trans. on Internet Tech.*, volume 6, 2006.
[3] M. Cremene, M. Riveill, and C. Martel. Autonomic adaptation based on service-context adequacy determination. *Electronic Notes in Theoretical Computer Science*, 2007.
[4] Technical University of Cluj-Napoca Distributed Systems Research Laboratory. http://dsrl.coned.utcluj.ro.
[5] D. Fournier and S. Ben Mokhtar. Towards ad hoc contextual services for pervasive computing. In *Proc. of the 1st workshop on Middleware for SOC*, pages 36–41, 2006.
[6] T. Hofer, W. Schwinger, and M. Pichler. Context-awareness on mobile devices - the hydrogen approach. In *Proc. of the 36th Annual Hawaii Int. Conf. on System Sciences*, 2003.
[7] C. Klein, R. Schmid, C. Leuxner, W. Sitou, and B. Spanfelner. A survey of context adaptation in autonomic computing. In *Proc. of the Fourth International Conference on Autonomic and Autonomous Systems*, pages 106 – 111, 2008.
[8] K. C. Lee and J. H. Kim. Implementation of ontology based context-awareness framework for ubiquitous environment. In *Conf. on Mult. and Ubiq. Engineering*, page 278 282, 2007.
[9] D. Raz, A. Tapani Juhola, J. Serrat-Fernandez, and A. Galis. *Fast and Efficient Context-Aware Services*. Wiley Series on Communications Networking & Distributed Systems, 2006.
[10] I. Salomie, T. Cioara, I. Anghel, and M. Dinsoreanu. Rap - a basic context awareness model. In *Proc. of the 4th IEEE Int. Conf. on Intel. Comp Communication and Processing*, 2008.
[11] G. Spanoudakis and K. Mahbub. A platform for context aware runtime web service discovery. In *Proceedings of the IEEE Int. Conference on Web Services*, pages 233–240, 2007.
[12] J. Thangarajah, L. Padgham, and J. Harland. Representation and reasoning for goals in bdi agents. In *Proc. of the 25th Australasian conf. on Comp. science*, pages 259–265, 2002.

[1] Technical University of Cluj-Napoca, 15 Daicoviciu str, Cluj-Napoca, Romania
E-mail address: Tudor.Cioara@cs.utcluj.ro, *Phone*: 0040264401443