A Self-Adapting Algorithm for Context Aware Systems

Tudor Cioara, Ionut Anghel, Ioan Salomie, Mihaela Dinsoreanu,
Georgiana Copil and Daniel Moldovan
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
{Tudor.Cioara, Ionut.Anghel, Ioan.Salomie, Mihaela.Dinsoreanu}@cs.utcluj.ro

***Abstract*— This paper presents a self-adapting algorithm that can automatically detect the changes in a system execution context and decide how the system should react. The self-adapting algorithm is characterized by a closed feedback loop with four phases: monitoring, analyzing, planning and execution. The monitoring phase uses the RAP (Resources, Actions, Policies) context model to represent in a programmatic manner the raw data collected about the system's self and execution environment. In the analysis phase, the context entropy concept is used to evaluate the context situation for detecting the context changes and determining the degree of respecting a predefined set of policies. The planning phase uses a reinforcement learning based technique to explore all possible system's states and select the adaptation action that should be executed by the system as a response to the context changes. The execution phase modifies the system behavior by enforcing the adaptation actions selected in the planning phase.**

Keywords – autonomic computing, pervasive computing, self-adaptive, MAPE phases, reinforcement learning

I. INTRODUCTION AND RELATED WORK

The complex structure and execution environment of today's distributed systems makes their management an extremely difficult task, which usually requires human intervention. There is a high demand for reducing the distributed system management complexity while ensuring a good system tolerance and robustness. A promising solution is to develop self-adaptive systems which automatically change their observable behavior and structure according to the execution environment internal or external conditions [1]. A self-adaptive system features a closed feedback loop with four MAPE phases: Monitoring, Analysis, Planning and Execution. In the *monitoring phase*, the self-adaptive system continuously monitors itself and its execution environment in order to capture and represent the relevant information. The *analysis phase* deals with detecting significant changes in the system self and execution environment while the *planning phase* decides and selects the appropriate adaptation actions as response to the detected changes. The *execution phase* modifies the system behavior by enforcing the adaptation actions selected in the planning phase.

This paper proposes a self-adapting algorithm that can automatically detect and analyze the changes in a system execution context, decide how the system should react and execute such decisions. For the monitoring phase we have used our sensor based monitoring infrastructure detailed in [2]. The system execution context is represented in a programmatic manner using our RAP context model [3]. The analysis phase evaluates the system execution context changes and calculates the context entropy values. We have defined and used the context entropy concept to reflect the degree of fulfilling a predefined set of policies that guide the system execution. The policies are described in XML and are automatically converted into SWRL (Semantic Web Rule Language) using our policy representation and evaluation model, proposed in [4]. The planning phase has four main steps: (1) identify and select the policy p with the highest contribution to the context entropy, (2) determine the set of resources affected by p, (3) organize the affected resources into inter-independent resource groups (IIRGs) and (4) use reinforcement learning to generate the best action to be taken in order to bring the IIRG's resources as close as possible to policy p compliant state.

The research regarding the self-adaptivity problem is focused on tree major directions: (i) the development of models and tools for acquiring and formally representing the system execution context, (ii) the development of models and techniques for analyzing the system execution context related information and (iii) the development of models and algorithms that allow computational systems to decide, select and execute actions according to the context or situation at hand.

The most important problems related to *context information acquisition* are to identify the characteristics that define the system execution context [5], and to define models for capturing the characteristics specific information [6]. In the domain literature, several characteristics that may define the system execution context are considered [7], [8], [9]: spatiotemporal characteristics (time and location), ambiental characteristics, facility characteristics (the system devices and their capabilities), user-system interaction, system internal events, system life cycle, etc. Regarding *context representation*, generic models that aim at accurately describing the system execution context in a programmatic manner are proposed. In [10] the use of key-value models to represent the set of context characteristics and their associated values is proposed. Markup models [11] and object oriented models [12] are also used to structure and represent the context information. The use of ontologies to represent the context information is proposed in [13]. The context characteristics are represented as ontological concepts and instantiated with run-time sensor captured values.

In the *context analyzing* research direction, models and techniques that aim at determining and evaluating the context changes are proposed. These models are strongly correlated with the system execution context representation model. In [14] 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 proposed in [16], [17]. Context changing rules are described using natural language [15] or first order logic and evaluated using reasoning engines.

Models and algorithms that allow computational systems to *decide* on the adaptation actions that have to be executed according to the context or situation at hand are proposed. The objective is to associate a certain degree of intelligence to the computational systems for context adaptation [1]. In [18], the authors propose a context adaptive platform based on the closed loop control principle. [19] proposes adaptation models based on defining the system behavior in a certain situation using a set of context adapting rules. A self-adapting model that uses a system situation space to represent the system execution context is proposed in [20]. Using learning algorithms, the system may infer the action to be executed for a new situation by placing it in a situation space group.

II. THE SELF-ADAPTING ALGORITHM

To define and formalize the self-adapting property of a context aware system we have used three main situation calculus concepts: (1) the *situation* concept also referred as system situation or context situation, represents the state of a context aware system (system's self and execution environment) at a moment of time, (2) a *set of conditions* that guide and control the system execution and (3) the *action plans* that may be executed by the context aware system in order to enforce the set of conditions for a specific situation. Using these concepts, we define the self-adapting property of a context aware system as a function, which associates to each system context situation s that fails to fulfill the prescribed set of conditions, an action plan a, that should be executed in order to reinforce the set of conditions.

```
1.  input: RAP_model - the context model sets R,A,P
2.         S^A - the set of existing equvalence classes
3.         T_E - the entropy threashold
4.  output: a new adapted system state
5.  procedure selfAdaptingAlgorithm(RAP_model, S^A, T_E)
6.  begin_MAPE
7.   //Monitoring Phase
8.    envI = gatherEnviromentInformation();
9.    s = createContextSituation (RAP_model, envI)
10.  //Analisys Phase
11.   E_S = evaluateContextSituationEntropy(s, P)
12.  //Planning Deciding Phase
13.   action = Ø
14.   if (E_S > T_E) then
15.    s^a = determineEquivClass(P, s, S^A)
16.    if (s^a in S^A)
17.     action = a
18.    else
19.     a' = determineActionsSequence(P, E_S, T_E)
20.     action = a'
21.     s^a = createNewEquivClass(P, s, a')
22.     S^A = S^A + s^a
23.    end if
24.   end if
25.  //Execution Phase
26.   executeAction(actionSeq)
27. end_MAPE
```

Figure 1. The self-adapting algorithm

Fig. 1 presents the proposed self-adapting algorithm which is based on a control feedback loop with the following MAPE phases: (i) the Monitoring Phase – collects raw data about the system's self and execution environment and uses it to construct and represent the system context situation in a programmatic manner (lines 7-9), (ii) the Analysis Phase – analysis the context situation in order to detect significant abnormal changes (lines 10-11), (iii) the Planning Phase – decides and selects the appropriate adaptation action plan to be executed (lines 12-24), (iv) the Execution Phase – the proper execution of the selected action plan (lines 25-26).

The following sub-sections detail the first three phases of the self-adapting algorithm. The execution phase is trivial and is not further detailed.

A. The Monitoring Phase

The goal of the self-adapting algorithm monitoring phase is to collect the system's self and execution environment raw data and to construct / represent the system context situation in a programmatic manner. The system's self and execution environment raw data describes the context in which the system evolves. The system's self raw data represents the system internal state and is used to assure the system's self-awareness. This data is usually gathered using low level system calls, logging or life cycle events. The system's execution environment raw data is captured using sensor networks or intelligent devices [2] with the goal of assuring the system context awareness.

To represent the system's context situation in a programmatic manner we have used our RAP (Resources, Actions and Policies) context model defined in [3]. The RAP model defines two types of context information representations: (i) set based, used to determine the system execution environment changes and (ii) ontology based, used to infer new context 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 provide raw data about the system's self and execution environment, A is the set of actions which are executed to enforce system execution rules and P is a set of policies which define the rules that guide the system execution. 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 RAP context model instance represents a context situation and contains the values of the RAP context model sets elements in a specific moment of time. The RAP context model instance is ontologically represented by the core ontology together with the domain specific concepts sub trees and their instances.

B. The Analisys Phase

The analysis phase targets the evaluation of the system context situations in order to detect those situations in which the rules that guide the system execution are broken. The set of guiding rules or conditions are described as RAP policies using our policy representation / evaluation model proposed in [4]. For measuring the degree of fulfilling the set of policies of a system context situation, we have defined the concept of context situation entropy (E_S) and its associated threshold (T_E). The next sub-sections give a short overview of the policy representation / evaluation model and the context situation entropy concept.

1) The policy representation / evaluation model

The policy representation model uses three main XML elements to describe a policy: Reference, Subject and Target.

The XML Reference element represents a collection of system resources (see Fig. 2 lines 1-7 for an example). Each reference has a unique name that can be referred from all the other XML elements of the policy. The reference resources collection is defined by applying three types of restrictions: domain restriction (location as criteria), type restriction (resource class as criteria) and property restriction (resource properties other than location as criteria).

The XML Subject element uses a reference name to indentify the system resources collection on which the policy imposes restrictions (see Fig. 2, lines 8-16). The Subject element has two children: (i) EvaluationTrigger XML element that contains a set of possible events that may trigger the policy evaluation and (ii) EvaluationCondition XML element containing a set of conditions for which the policy is enforced.

The XML Target element represents a collection of resources for which the goals specified by the PolicyGoals XML element applies (Fig. 2 lines 17-21). The PolicyGoals element is used to determine the action plan that has to be executed to enforce the policy.

```
1  <Reference Name="DSRLTEMP">
2    <Restrictions>
3      <DomainRestriction Domain="DSRL"/>
4      <TypeRestriction Type="TemperatureSensor"/>
5      <PropertyRestriction Property="InfluenceZone"
                    Value="10" Operator="greaterThan" />
6    <Restrictions>
7  </Reference>
8  <Subject Name="DSRLTEMP">
9    <EvaluationTriggers>
10     <Trigger Event="TemperatureChangedEvent"/>
11   </EvaluationTriggers>
12   <EvaluationConditions>
13     <Condition Property="TemperatureValueProp"
                    Operator="equal" Value="22">
14     </Condition>
15   </EvaluationConditions>
16 </Subject>
17 <Target Name="DSRLTEMP">
18   <PolicyGoals>
19     <Goal Property="TemperatureValueProp" Value="22" />
20   </PolicyGoals>
21 <Target>
```

Figure 2. The XML representation of the policy "In the DSRL laboratory the temperature must be 22 degrees"

In order to be evaluated, the policies are automatically converted into SWRL (Semantic Web Rule Language) rules and injected into the RAP specific context model instance ontology. The SWRL rules are used to reason about RAP context model instance ontology individuals in terms of ontology classes and properties. Rules are written in the form of an implication between an *antecedent* (body) and a *consequent* (head).

The process of obtaining the *SWRL rule antecedent* from the XML policy description consists from two phases: (i) the generation of the SWRL atoms used to identify the specific context model instance ontology individuals involved in the reasoning process and (ii) the transformation of the XML policy evaluation condition into SWRL rules.

The *SWRL rule consequent* contains a SWRL atom that sets the EvaluationResultProp attribute to true for all RAP context model Policy class individuals, which correspond to the XML broken policies. The Policy EvaluationResultProp class attribute value stores a Boolean information regarding the degree of fulfilling the policy.

2) The system context situation entropy

The system context situation entropy measures the level of system's self and execution environment disorder by evaluating the degree of fulfilling the policies. If the evaluated system context situation entropy is below a predefined threshold T_E, then all the policies are fulfilled and adaptation is not required. Otherwise the system must execute adaptation actions to control its self and execution environment and to keep the system entropy below T_E.

The entropy for a certain system context situation is:

$$E_S = \sum pw_i \sum rw_{ij} * v_{ij} \quad (1)$$

where:

- pw_i is the weight of the policy i, and represents the importance of the policy for the system execution
- rw_{ij} is the weight of the system resource i in the policy j. The resource weight reflects the system resource importance for the policy. If a policy imposes no restrictions to a system resource then the resource weight of the resource for that policy is zero.
- v_{ij} is the deviation between the system resource j recorded value and the policy i accepted value.

We also define the entropy contribution (E_i) of a policy:

$$E_i = pw_i \sum ri_{ij} * v_{ij} \quad (2)$$

Taking into account the system toleration to changes, we have defined two types of entropy thresholds: *restrictive threshold* and *relaxed threshold*. In the first case, we define the threshold $T_E = 0$ as lowest entropy value. This means that, whenever a policy-imposed restriction is broken, the self-adapting algorithm is triggered. In the second case, for each policy we define an accepted entropy contribution value E_i and compute current threshold T_E as in (3). For the relaxed threshold, broken policies are tolerated if their entropy contribution is lower than the accepted contribution value.

$$T_E = \sum pw_i * v_{ij} * (100 + Ei \% 100) \quad (3)$$

C. The Planning Phase

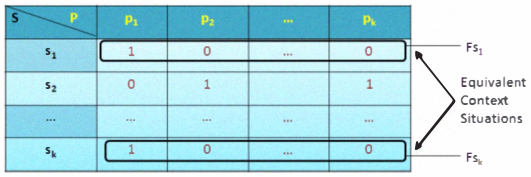
This phase deals with deciding and planning the actions that need to be taken in order to enforce a broken policy. The phase starts with identifying previously encountered similar context situations in which the same policies were broken. If a similar situation is found, the same action plan is selected and executed, otherwise a new action plan must be constructed.

1) Equivalent system context situations

To identify and classify similar system context situations we have defined the equivalence context situation class concept and an equivalence relation. Two context situations are equivalent if both fulfill and break the same policies. To

clasify the system context situations in equivalence classes we use the information system theory. An information system is defined as an association (W, Atr) where: W is a non-empty finite set of objects and Atr is a non-empty finite set of attributes. For this association a function a_f that assigns to every object in W, a value for each attribute in Atr, can be defined. For a context aware system, the non-empty set of objects W is mapped onto the set of context situations S while Atr is mapped to the set of policies P. We define a function, Fs_k, that assigns to every system context situation s_k, a list of values returned by evaluating all the defined policies in that context situation. Two system context situations s_1 and s_2 belong to the same equivalence class if and only if Fs_1 and Fs_2 generate the same lists of policy evaluation values (Table 1).

TABLE I. DETERMINING THE EQUIVALENCE CLASSES FOR THE SYSTEM CONTEXT SITUATIONS



As a result, *the equivalence class* is defined as the set of system context situations in which the same adaptation action plan a is executed (we use the notation S^a, where a is the adaptation action plan).

2) Adaptation actions planning

The action selection phase of the self-adapting algorithm considers the broken policies one by one, ordered by their contribution to the overall entropy (see relation 3). The action selection phase can be divided in four main steps (Fig. 3): (1) select the policy with the highest contribution to entropy (line 9), (2) for the selected policy, determine the set of affected system resources and organize them into inter-independent resource groups (IIRGs) (line 10) and (3) use reinforcement learning (if necessary) to generate the best action sequence to be taken in order to bring the IIRG's resources as close as possible to a policy compliant state (lines 11-15).

```
1   input: P - the set of context policies
2          Es – the entropy value for the curent context situation
3          TE - the entropy threashold
4   output: actionsToBeTaken - the sequence of actions to be
           executed
5   procedure determineActionsSequence(P, Es, TE)
6   begin
7    ActionsList actionsToBeTaken;
8    while Es > TE do
9     policy = determineLargestContributionToEntropyPolicy ()
10    IIRGs = IIRGExtraction(policy)
11    while ( faultyResourcesNotTaken(policy) ) do
12     faultyResource = selectFaultyResource(policy)
13     IIRG = selectInterIndependentResourcesGroup(IIRGs)
14     seqActions = reinforcementLearning(faultyResource, Es,
                                          rootNode, rootNode)
15     add(actionsToBeTaken, seqActions)
16    endwhile
17   endwhile
18   return actionsToBeTaken
19  end
```

Figure 3. The actions selection phase procedure pseudo-code

The *inter-independent resource group (IIRG)* concept is defined and used into the reinforcement learning algorithm to avoid the situation when healing a context policy may cause breaking another one because their associated resources are depended. For example, a temperature sensor and a humidity sensor are dependent taking into consideration a room temperature policy and a human thermal comfort factor policy. In this situation, an action executed to heal a specific policy (e.g. room temperature policy), changes the policy associated resources values and triggers a variation of its depended resources values with the result of breaking another context policy (e.g. human thermal comfort factor policy). If all IIRG resources comply with the policy rules, no action is executed. More details about the IIRG concept and our IIRG extraction algorithm can be found in [21].

3) Determining the Best Action Sequence

The best actions sequence to be taken for a certain IIRG is found by merging the best sequences of actions determined for each IIRG resources. For an IIRG resource, the sequence of actions is computed through a reinforcement learning algorithm, by using a policy iteration technique (see Fig. 4).

The learning process considers all possible system context situations and builds a decision tree by simulating the execution of all available actions for each system context situation. A tree node represents a system context situation. A tree path between two nodes A and B defines a sequence of actions which, executed on the system context situation represented by the node A, generates a new system context situation represented by the node B. The reinforcement learning process uses a reward / penalty approach to find the best sequence of actions that the system may execute in a given context situation. In our case, the minimum entropy path in the reinforcement learning decision tree represents the best sequence of actions.

```
1   input: faultyResource - faulty resource
2          rootNode - reinforcement learning state tree root node
3          currentNode - reinforcement learning state tree current node
4   output: Decision tree minimum path wich represents
           the sequence of actions
5   procedure reinforcementLearning(faultyResource, currentEntropy ,
                                    rootNode, currentNode)
6   begin
7    Actions = getActions(resource)
8    for each ( action ∈ Actions) do
9     newEntropy = simulateActionExecution(faultyResource, action)
10    newNode = addChild(currentNode, newEntropy, resource)
11    if ( allCycles() = FALSE ) then
12     if (newEntropy < TE ) then
13      return pathFromRootActionsList(currentNode)
14     else
15      if (newEntropy < currentEntropy) then
16       reinforcementLearning(faultyResource, newEntropy, root, newNode)
17      end if
18     end if
19    else
20     return minimumPathActionsList()
21    end if
22   end for
23  end
```

Figure 4. The reinforcement learning algorithm

The learning process may generate different type of results discussed below.

Case 1: The algorithm finds only a possible system context situation that has its entropy lower than the predefined threshold. In this case the sequence of actions that lead to this context situation is selected and the search is stopped.

Case 2: The current system context situation entropy is higher than the threshold, but smaller than the minimum entropy determined so far. We replace the minimum entropy

with the new entropy and continue with the search process.

Case 3: The current entropy is higher than both the threshold and the minimum entropy; the reinforcement learning algorithm continues the search process. If all exercised paths of the decision tree are cycles, the algorithm stops and chooses the path leading to a state with the minimum entropy.

III. CASE STUDY

As case study, we develop an intelligent pervasive system used to manage our DSRL (Distributed Systems Research Lab) smart laboratory. The system uses a sensor network to collect the information regarding students' location or orientation and ambiental information like light level, temperature or humidity. Also in the laboratory, the students are identified using a camera with face recognition software. The system uses a set of actuators to control the laboratory environmental characteristics and to enforce the actions executed as a response to policy breaking. The system infrastructure components (sensors and actuators) and their possible values / default actions are shown in Table 2.

TABLE II. THE SYSTEM INFRASTRUCTURE COMPONENTS

Infrastructure Component	Type	Possible values/ Available actions
Temperature	Sensor	Z °C
Humidity	Sensor	[0-100]%
Light	Sensor	{ON, OFF}
Computer State	Sensor	{ON, OFF}
Alarm State	Sensor	{ON, OFF}
Face Recognition	Camera	{Professor, Student, Unknown}
Room State	Sensor	{Empty, Not Empty}
Air Conditioning Unit	Actuator	{Decrease by 5°C, Decrease by 2 °C}
Heater	Actuator	{ Increase by 5 °C }
Humidity Controller	Actuator	{Increase by 3 %,Decrease by 3 %}
Light Controller	Actuator	{Turn ON, Turn OFF}
Alarm Controller	Actuator	{Turn ON, Turn OFF}
Computer Controller	Actuator	{Turn ON, Turn OFF}

In the smart laboratory, a set of policies is defined to govern the environmental characteristics and the access to the laboratory resources. The set of policies used to drive the intelligent system management process execution are presented in Table 3.

TABLE III. THE SMART LABORATORY POLICIES

Policy	Accepted values
Temperature Humidity	$Temperature \in (18°C, 23°C)$ $Humidity \in (20\,\%, 30\,\%)$
Actor Recognition	$\boldsymbol{Alarm} = \begin{cases} OFF, Face\ Recognition = Professor \mid Student \\ ON, Face\ Recognition = Unknown \end{cases}$ $\boldsymbol{Computer} = \begin{cases} ON, Face\ Recognition = Professor \\ OFF, Face\ Recognition = Unknown \mid Student \end{cases}$
Light	$Light = \begin{cases} OFF, Room\ empty \\ ON, Room\ not\ empty \end{cases}$

For simulation purposes, we have provided two sensor values manipulation mechanisms: a Context Disturbing Mechanism (CDM) and a Direct Manipulation Mechanism (DMM). CDM assigns values to sensors using a predefined pattern (or random values) in order to generate complex context situations in which adaptation is needed. DMM uses an Extensible 3D (X3D) implementation, to represent the smart laboratory infrastructure as X3D objects (Fig. 5). A click event performed on an object generates a change in the object state. The X3D approach allows us to accurately construct real life scenarios and closely control their evolution.

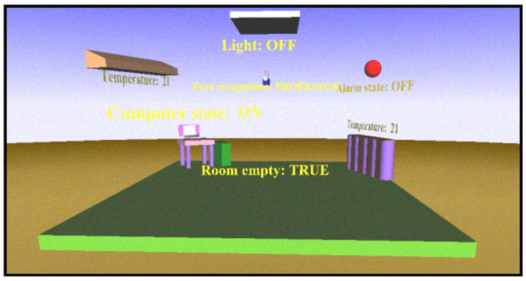


Figure 5. The X3D representation of the DSRL managing system

For testing purposes, the following scenario is considered: a Professor enters the laboratory, the Computer is "OFF" and the Alarm is "ON". The system constructs an instance of the current context situation and uses it to evaluate the SWRL representation of all defined policies. As a result of the evaluation process, the "Actor Recognition" policy is broken (see Table 3) and the managing system needs to take adaptation decisions. In the decision process, the self-adapting algorithm uses the reinforcement learning based approach presented in section C.3 and constructs a decision tree by simulating the execution of all actuators available actions. In our test case scenario, the "Actor Recognition" policy affects the following system infrastructure components: the Alarm and the Computer State sensors. For each resource, we simulate the following actions execution: (i) turn the computer on / off, executed by the Computer controller and (ii) turn the alarm on / off, executed by the Alarm Controller. The process continues until the best action sequence that generates a context situation with minimum entropy is found (Fig. 6). For the above presented scenario, we assume the following: (i) the context entropy threshold is T_E=0, (ii) the weights for policies and resources are equal to 1 and (iii) all actions have an immediate effect.

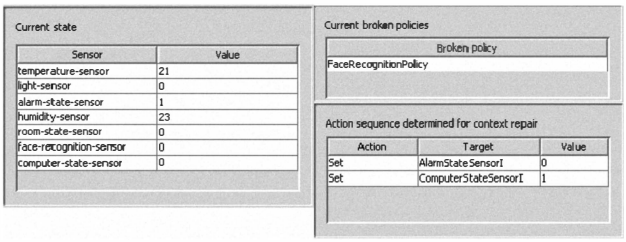


Figure 6. The Actor Recognition Policy broken

To assess the self-adapting algorithm's performance, the CDM mechanism is used in two different tests. In the first test, a pattern of four context situations that need adaptation is generated: (i) the professor is in the room while the Computer is "OFF" and the Light is "OFF", (ii) the student is in the room while the alarm is "ON", (iii) the Temperature and Humidity are out of their admissible ranges and (iv) an

unknown person is in the room and the Alarm is "OFF". Fig. 7 graph shows the time needed by the self-adapting algorithm to analyze each of the four context situation and to identify the corresponding actions plans: 20, 4, 61 and respectively 22 seconds. When a similar situation was previously encountered, the time decreases to 0 seconds because the self-adapting algorithm directly selects the same adaptation actions.

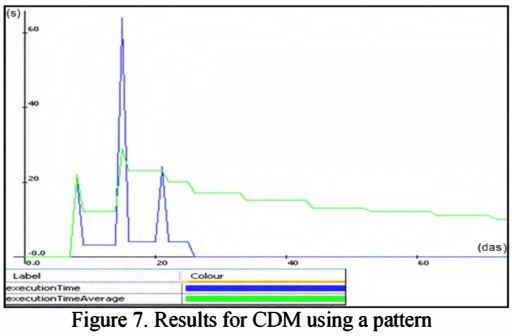


Figure 7. Results for CDM using a pattern

For the second test case, the self-adapting algorithm was run for about 3 hours (Fig. 8). During this time the CDM mechanism gave random values to all sensors as following: (i) for the Temperature sensor [15…25], (ii) for Humidity sensor [15…35], (iii) for the Light, Room State, Computer State and Alarm sensors 0 or 1 ("OFF", "ON") and (iv) for the Face Recognition camera 0, 1, 2 ("Professor", "Student", "Unknown"). Each variation in the plot represents a situation in which an unknown context situation has been encountered and a search for the best sequence of actions was performed. In the first 1000 seconds, almost all running times of the action selection algorithm are greater than 10 seconds. After that, the self-healing mechanism begins to learn, achieving the performance of having only four running times greater than 10 seconds in the [5000, 7000] time interval.

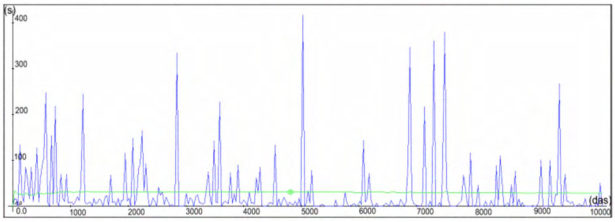


Figure 8. Results for CDM with random values

Also, an overall reduction in the number and height of the peaks is visible because at each step the algorithm checks if it doesn't already know the best sequence of actions for the current situation. Considering that the number of possible sensor combinations for the chosen ranges is 22.481.940, the self-adapting algorithm shows promising results.

IV. CONCLUSIONS

This paper presents a self-adapting algorithm that can automatically detect and analyze the changes in a system execution context and decide how the system should react, and execute adaptation actions. The test case results are promising, showing that the self-adapting algorithm is capable to take adaptation decisions in various types of context situations. Also the adaptation decision time tends to decrease in time as a result of the learning process.

REFERENCES

[1] M. Salehie, L. Tahvildari, "Self-adaptive software: Landscape and research challenges" ACM Tran. on Aut. and Adaptive Systems, ISSN:1556-4665, 2009.

[2] I. Anghel, T. Cioara, I. Salomie, M. Dinsoreanu, "A Self-configuring Middleware for Managing Context Awareness", Int. Conf. on Wireless Inf. Net. and Systems, pp. 131-139, ISBN: 978-989-674-008-5, 2009.

[3] T. Cioara, I. Anghel, I. Salomie "A Generic Context Model Enhanced with Self-configuring Features", Journal of Digital Information Management (JDIM), Volume 7/ 3, pp.159-165, ISSN 0972-7272, 2009.

[4] T. Cioara, I. Anghel, I. Salomie "A Policy-based Context Aware Self-Management Model", 11th Int. Symposium on Symbolic and Numeric Alg. for Scientific Comp., 333-341, ISBN: 978-0-7695-3964-5, 2009.

[5] K.Wang, "Context awareness and adaptation in mobile learning", Proc. of the 2nd IEEE Int. Work. on Wireless and Mobile Tech. in Education, pp. 154-158, ISBN:0-7695-1989-X , 2004.

[6] Z. Yu, "iMuseum: A scalable context-aware intelligent museum system, Computer Communications", Vol. 31/18, pp. 4376-4382 , 2008.

[7] C. Burghardt, C. Reisse,"Implementing scenarios in a Smart Learning Environment", 6th Annual IEEE Int. Conf. on Pervasive Computing and Communications, ISBN: 0-7695-3113-X, 2008.

[8] L. Pareschi, D. Riboni, "Composition and Generalization of Context Datafor Privacy Preservation", 6th Annual IEEE Int. Conf. on Perv. Comp. and Comm. ISBN: 0-7695-3113-X, 2008.

[9] M. Grossniklauss, Context Aware Data Management, 1st ed, VDM Verlag, ISBN 978-3-8364-2938-2, 2007.

[10] 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, Denmark, pp. 99 – 110, 2006.

[11] D. Raz, A. T. Juhola, "Fast and Efficient Context-Aware Services", Wiley Series on Comm. Networking & Distributed Systems, ISBN-13: 978-0470016688, pp. 5-25, 2006.

[12] T. Hofer, W. Schwinger, M. Pichler, "Context-awareness on mobile devices – the hydrogen approach", In Proc. of the 36th Annual Hawaii International Conference on System Sciences, USA, pp. 292, 2003.

[13] I. Cafezeiro, E. Hermann, "Ontology and Context", 6th Annual IEEE Int. Conf. on Perv. Comp. and Comm. ISBN: 0-7695-3113-X, 2008.

[14] Q. Huaifeng, "Integrating Context Aware with Sensornet", Proc. of 1st Int Conf. on Semantics, Knowledge, Grid, ISBN:0-7695-2534-2, 2006.

[15] A. Bernstein, E. Kaufmann, "Querying the Semantic Web with Ginseng:A Guided Input Natural Language Search Engine", Proc. of 15th Work. on Information Technology and Systems, 2005.

[16] E. Sirin, B. Parsia, "Pellet: A practical OWL-DL reasoner", Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 5, No. 2., pp. 51-53, 2007.

[17] M. Amoui, M. Salehie, "Adaptive Action Selection in Autonomic Software Using Reinforcement Learning", Proc. of the 4th Int. Conf. on Aut. and Autonomous Sys., pp. 175-181, ISBN 0-7695-3093-1, 2008.

[18] M. Cremene, M. Riveill "Autonomic adaptation based on service-context adequacy determination" Electronic Notes in Theoretical Computer Science, 2007.

[19] M. Parashar, and S. Hariri, "Autonomic Computing: An Overview", LNCS Springer Verlag, Vol. 3566, pp. 247 – 259, 2005.

[20] N. O'Connor, R. Cunningham, "Self-Adapting Context Definition", 1st Int. Conf. on Self-Adaptive and Self-Organizing Systems, 2007.

[21] T. Cioara, I. Anghel, I. Salomie, "A Reinforcement Learning based Self-healing Algorithm for Managing Context Adaptation", 8th Annual IEEE Int. Conf. on Perv. Comp. and Communications, 2010.