A Policy Driven Self-healing Algorithm for Context Aware Systems

Ionut Anghel, Tudor Cioara, Ioan Salomie, Mihaela Dinsoreanu, Anca Rarau
Technical University of Cluj-Napoca, Computer Science Department
{Ionut.Anghel,Tudor.Cioara, Ioan.Salomie, Mihaela.Dinsoreanu, Anca.Rarau}@cs.utcluj.ro

Abstract

This paper proposes a generic self-healing algorithm that automatically detects, diagnoses and repairs the problems appeared during the context aware systems' adaptation processes. The set, situation calculus and information system theories are used to define and formalize the concepts of context situation entropy and equivalent context situations together with a set of self-healing principles. These concepts and principles are used to generically define the self-healing algorithm. The self-healing property is enforced by monitoring the systems execution environment for: (i) evaluating the degree of fulfilling the context policies of the current context situation (context situation entropy), (ii) obtaining the equivalent context situations and (iii) determining the actions that have to be executed in order to keep the system in consistent functional states. The self-healing algorithm is validated on our RAP context model by enhancing it with a generic policy representation technique for run-time evaluation.

1. Introduction and Related Work

The context aware systems continuously monitor, capture and interpret information related to their execution environment in order to adapt their behavior to changes. The complex nature of today's context aware systems execution environment makes the system management and adaptation processes an extremely difficult task. The self-management capabilities have become a mandatory requirement for creating systems which automatically change their observable behavior and structure according to the internal or external conditions of their evolution environment. The self-management capabilities can be achieved by using the autonomic computing paradigms (self–configuring, self–healing, self–optimizing and self-protecting) for the development and integration of self-* enhanced components into context aware systems [1].

In this paper we propose a generic self-healing algorithm that can be used to automatically detect, diagnose and repair the problems appeared during the context adaptation processes. By mapping the situation calculus theory onto context aware systems we define the self-healing property in terms of fulfilling degree for a predefined policies set that describes the systems correct behavior and should drive the system execution, accordingly. Using set and information system theories we define and formalize the concepts of context situation entropy and equivalent context situations together with a set of self-healing principles. Based on these concepts and principles, the self-healing property of a context aware system is enforced by monitoring the system execution environment in order to: (i) evaluate the degree of fulfilling the context policies for the current context situation (context situation entropy), (ii) obtain the equivalent context situations and (iii) determine the actions that have to be executed in order to keep the system in consistent functional states. The self-healing algorithm is validated on our RAP context model [2] by enhancing it with a generic policy representation method for run-time evaluation.

The current researches related to the self-healing autonomic feature, in different contexts, focus upon specifying and developing system models that should allow for: (i) identifying the causes of failures or crashes [3], (ii) identifying possible errors in the system life cycle [4] and (iii) performing run-time diagnosis and offering solutions [5]. The proposed models are based on anticipating and avoiding run-time problems as well as identifying ways to restore the system in case of failure. The adoption of biological principles such as decentralization, autonomy, natural selection or symbiosis in the process of designing and building self-healing application components or services is a novel research direction [6]. A component of an application is designed as a biological entity, equivalent to an individual bee in a

978-1-4244-5007-7/09/$26.00 ©2009 IEEE

bee colony that competes or collaborates for computing resources. Using natural selection principles, the components that don't respect a set of predefined rules or policies are banned for execution. Self-healing techniques that use reinforcement learning in dealing with crashes and denial of service have been studied in [7]. In case of a crash, the authors propose action selection techniques based on learning from previous experience. In [8] the authors propose a self-healing and optimizing mechanism for minimizing energy consumption fluctuations of network devices and thus reducing the network global energy consumption. The approach is based on defining an agent with associated energy plans for each managed device, part of the tree-like hierarchy of devices. IBM researchers R. Das and J. O. Kephart use virtualization and load-balancing techniques for achieving self-healing and self-optimizing of data centers resources [9]. By combining different types of agent based managers (power, performance and coordination), the data center available resources are efficiently administrated without SLA penalties.

The rest of the paper is organized as follows: Section 2 defines the self-healing property for context aware systems and presents the generic self-healing algorithm; Section 3 shows how the self-healing algorithm is applied on our RAP context model together with simulation results while Section 4 concludes the paper and describes the future work.

2. Defining the Self-Healing Property for Context Aware Systems

In order to define and formalize the self-healing property of a context aware system we use the *situation calculus theory* [10]. By mapping the situation calculus theory onto context aware systems, we have identified three main concepts that can be used to define the self-healing property: (i) the *situation* concept that represents the complete state of the context aware system execution environment at a moment of time, (ii) a *set of conditions* that guide and control the system execution and (iii) the *action plans* taken by the context aware system in order to enforce the set of conditions for a specific situation.

A context aware system **situation** represents the system internal state together with a snapshot of its execution environment taken at a specific moment of time. Usually, the context aware system captures its situation related information using a set of real or virtual context resources (sensors networks, GPS, RFID etc.).

From now on we refer the *context aware system situation* as *context situation* and represent it with **s**. The set of all context situations **s**, for a context aware system is represented with **S.**

The **set of conditions** used to control the context aware system execution is represented using a set of context policies. A context policy is notated with **p** while the set of context polices is referred as **P**.

To enforce the set of conditions for a specific situation the context aware application must execute corresponding **action plans** notated with **a.** The set of all action plans, that a context aware system may execute is represented with **A.**

We define the self-healing property of a context aware system as a function:

$$\textbf{Self\_Healing: } P_s \rightarrow A$$
$$\forall s \in S \Rightarrow \exists a \in A, \textbf{Self-Healing}(P_s) = a \qquad (1)$$

The self-healing function is *non-injective* implying that there are distinct context situations for which the same action plan is executed in order to enforce the policy. As a result, the context situations are grouped in clusters using the selected action plan as a discriminator (see Figure 1).

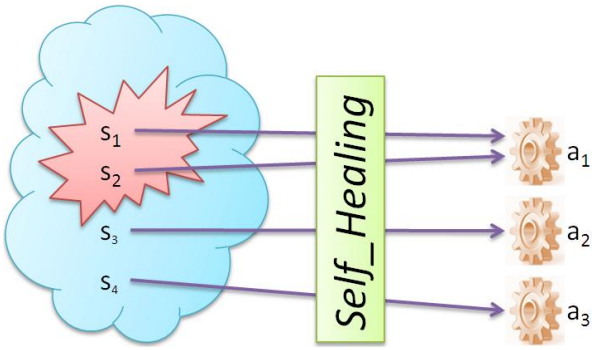

Figure 1. The Self-healing non-injective function

In the following sections we define and formalize the concepts of *context situation entropy* and *equivalent context situations* together with a *set of self-healing principles* to describe the self-healing algorithm**.** Using these concepts the self-healing property of context aware systems is enforced by monitoring the system execution environment in order to detect the context situations for which at least one context policy is broken. The self-healing property is implemented by executing three steps: (1) evaluating the context situation entropy as the degree of policy fulfilling for the current context situation, (2) obtaining the equivalent context situations and (3) determining

the actions that have to be executed in order to keep the system in consistent and healthy functional states.

2.1. Context Situation Entropy

In order to measure the degree of respecting the set of context policies for a given context situation, we define the concept of context situation entropy ($\mathbf{E_s}$) and its associated threshold ($\mathbf{T_E}$). The context situation entropy measures the level of system internal and external (environment) disorder by evaluating the degree of fulfilling the context policies. If the context entropy is below $\mathbf{T_E}$, then all context policies are fulfilled and the system is in a consistent and healthy functional state.

The entropy of a context aware system tends to increase in time when the system doesn't have any control over its internal/external state. In our approach, the control is exercised through the actions executed in order to keep the system entropy below $\mathbf{T_E}$.

To evaluate the context entropy we define a function ($\mathbf{P_{eval}}$) that takes a context situation and a specific context policy as input values and generates output values of zero or one. When $\mathbf{P_{eval}(s, p) = 1}$, the policy is not fulfilled for the **s** context situation, otherwise the policy is respected. Using the $\mathbf{P_{eval}}$ function, we can evaluate the entropy for a context situation:

$$\mathbf{P_{eval} : (s, p) \rightarrow \{0, 1\}}$$
$$\mathbf{E_s: P \rightarrow \mathbb{Z},\ E_s = \sum_{p \in P} P_{eval}(s, p)} \quad (2)$$

The entropy is used to determine the self-healing capacity of a context aware system. The context aware system has the self-healing property enabled if and only if the product of context entropy values for any two consecutive context situations is below the entropy threshold squared. This product is an invariant for the context aware system self-healing property:

$$\mathbf{E_s * E_{s++} < (T_E)^2} \quad (3)$$

2.2. Equivalent Context Situations

According to the *set theory* [11], an equivalence class is a subset of elements which respect the same collection of properties. The collection of properties that must be fulfilled by all elements part of an equivalence class defines an equivalence relation. For a context aware system we define the equivalence relation over the context situations set **S** based on the property of fulfilling and breaking context policies. Two context situations are equivalent (notation $\sim$) if both fulfill and brake the same context policies.

$$\mathbf{[s] = \{ x \in S \mid x \sim s\}} \quad (4)$$

For a context aware application, the equivalence relation over the context situation set S has the following properties:

- **Reflexivity**: $\forall\ \mathbf{s_1 \in S}$, $\mathbf{s_1 \sim s_1}$. Each context situation part of the **S** set is equivalent to itself. For the same context situation the same policies are always broken or fulfilled.
- **Symmetry**: $\forall\ \mathbf{s_1, s_2 \in S}$ **if** $\mathbf{s_1 \sim s_2}$ **then** $\mathbf{s_2 \sim s_1}$. If the context situation $\mathbf{s_1}$ is equivalent to another context situation $\mathbf{s_2}$ they both brake and fulfill the same context policies.
- **Transitivity**: $\forall\ \mathbf{s_1, s_2, s_3 \in S}$ **if** $\mathbf{s_1 \sim s_2}$ **and** $\mathbf{s_2 \sim s_3}$ **then** $\mathbf{s_1 \sim s_3}$. If the context situation $\mathbf{s_1}$ is equivalent to the context situation $\mathbf{s_2}$ and $\mathbf{s_2}$ is equivalent to $\mathbf{s_3}$, then we conclude that $\mathbf{s_1}$ and $\mathbf{s_3}$ brake and fulfill the same context policies.

To clasify the context situations in equivalence classes we use the *information system theory* [12]. 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 $\mathbf{a_f}$ that assigns to every **W** object, a value for each **Atr** attribute can be defined.

For a context aware system the non-empty set of finite objects **W** is mapped to the set of context situations **S** while the set of attributes **Atr** is mapped to the set of context policies **P**. We map the $\mathbf{a_f}$ function onto the $\mathbf{Fs_k}$ function that assigns a list of values returned by evaluating all policies using the $\mathbf{P_{eval}}$ function, to the context situation $\mathbf{s_k}$:

$$\mathbf{Fs_k: \rightarrow <P_{eval}(s_k, p_1), ,... P_{eval}(s_k, p_n)>} \quad (5)$$

Two context situations $\mathbf{s_1}$ and $\mathbf{s_2}$ belong to the same equivalence class if and only $\mathbf{Fs_1}$ and $\mathbf{Fs_2}$ generate the same output list of policy evaluation values (Table 1).

Table 1. Determining the equivalence classes for the context aware systems situations

S \ P	p_1	p_2	...	p_k
s_1	1	0	...	0
s_2	0	1		1
...	...	...	...	...
s_k	1	0	...	0

Fs_1 — Equivalent Context Situations — Fs_k

As a result, the equivalence class is defined as the set of context situations for which the same action plan a is executed in order to keep the system in a healthy state. For an equivalence class we use the notation $\mathbf{S^A}$, where **a** is the equivalence class attached action plan.

2.3. The Self-Healing Algorithm

Using the above presented self-healing concepts we can define four self-healing principles that are used to drive the self-healing process.

The indiscenability principle. The context aware system cannot discern between two equivalent context situations. It means that for the equivalent context situations, the context aware system will always take the same decisions in order to enforce the healthy state of the system.

Order preservation principle. During the self-healing adaptation processes, the context aware system must always preserve the order in which two context situations appear.

```
Input:
        P - the set of context policies
        s - the current context situation
        S^A - the set of existing equvalence classes
        T_E - the entropy threashold
Output: a healthy context aware system state

begin
  // evaluate the context entropy
  E_s = ∑_{p∈P} P_eval (s, p)

  if (E_s < T_E) then
    return   // s is a neutral context situation
  else
    s^a = determineEquivClass(P, s, S^A)
    if (s^a in S^A)
      // action plan already defined
      executeAction(a)
    else
      // generate and execute action plan
      a' = determineAction(P, s)
      s^a' = createNewEquivClass(P, s, a')
      S^A = S^A + s^a'
      executeAction(a')
end
```

Figure 2. The Self-healing algorithm

Adaptation consistency principle. Each equivalence class should be associated with an executable action plan. As a consequence, for a context situation **s** that belongs to the same equivalence class, the context aware system will always execute the same action plan.

Neutral context situation principle. A neutral context situation is a context situation in which the system is in a healthy state where no context policy is broken. As a result no action should be executed.

Figure 2 presents the proposed self-healing algorithm for a context aware system. The algorithm has three main phases: (i) evaluating the context entropy for the current context situation (ii) determining the equivalent context situations and (iii) determining the action plans that have to be executed in order to keep the system in consistent and healthy functional states.

3. Validating the Self-Healing Algorithm on the RAP Context Model

Let's consider the real world context represented by our Distributed System Research Laboratory [14]. 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, lesson hints services, print services, question-based services etc.). A sensor network captures information regarding student locations or orientation and also ambient information like the temperature or humidity. In the laboratory, a set of policies like "the temperature in the laboratory should be 22 degree Celsius" or "the loud upper limit is 80 dB" should be respected.

Next sections present how the first two phases of the proposed self-healing algorithm are validated using the above presented scenario and our RAP context model to represent the information context.

3.1. The RAP Context Model Overview

To represent a real world context in a programmatic manner (understandable for context aware applications) we use our RAP context model. This model defines the context as a triple: **C = <R, A, P>** where **R** is the set of context resources that generates and/or processes context information, **A** is the set of actors which interact with context resources in order to satisfy their needs and **P** is the set of real world context related policies. The set of context resources **R** is split in two disjunctive subsets: (i) the set of context resources attached to the real world context environment $\mathbf{R_E}$ and (ii) the set of context resources attached to the actors $\mathbf{R_A}$. In order to provide

an accurate representation of the execution environment, the following representation artifacts are defined: *specific context model*, *specific context model instance* and *context – actor instance*. The specific context model $\mathbf{C_S} = \langle \mathbf{R_S}, \mathbf{A_S}, \mathbf{P_S} \rangle$ is obtained by mapping the context model onto different closed environments and populating the sets with environment specific elements. A specific context model instance $\mathbf{C_{SI}} = \langle \mathbf{R_{SI}}, \mathbf{A_{SI}}, \mathbf{P_{SI}} \rangle$ contains the set of context resources with which the middleware interacts, together with their values in a specific moment of time **t**. The context – actor instance $\mathbf{CI_a^t} = \langle \mathbf{R_a^t}, \mathbf{a}, \mathbf{P^t} \rangle$ contains the set of context resources with which the actor can interact, together with their values in a specific moment of time **t**. The advantage of the RAP model is the ontological representation of the context model artifacts which allows for learning and reasoning in order to obtain high-level context information. The specific context model concepts are represented as sub trees of the core ontology by using is-a type relations. The context situation or the context instance is represented by the core ontology together with the specific context model concepts and their instances in a specific moment of time.

3.2. The RAP context model policy representation and evaluation

In order to create the execution conditions of the self-healing algorithm we enhance the RAP context model with an XML generic policy representation for run-time evaluation.

In the RAP context model we use two types of resources: (i) passive resources that capture and store context specific data and (ii) active resources that interact directly with the context and modify the context state. According to this classification we have defined *metrics constraints policies* and *action policies*.

The **metrics constraints policies** are defined for the set of passive context resources in order to impose 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 action policy constraints.

Both types of policies are described in XML using the following elements: *Reference*, *Subject* and *Target*.

The **XML Reference** element represents a collection of context entities (context resources and actors) on which the policy is applied. Each reference is associated a unique name. In order to be member of the context entity collection, a context entity should obey several restrictions. We have defined three types of restrictions with different degrees of generality: *domain restriction*, *type restriction* and *property restriction*. The *domain restriction* (defined by the DomainRestriction XML element and its Domain attribute) is the most general restriction type and uses the physical location to create the Reference context entities collection. The *type restriction* (defined by the TypeRestriction XML element and its Type attribute) restricts Reference context entities collection using the entities class as a criteria. The property restriction (defined by the PropertyRestriction and its Property, Values and Operator attributes) further restricts the Reference context entities collection to those context entities that satisfy a certain condition related to one or more properties, other than location. The Type, Domain and Property attributes values must be classes defined in the RAP context model OWL ontology.

For example, in our test case scenario, to define the Reference context entities collection that contains all movement sensors with their influence zone higher than 5 meters placed in DSRL laboratory, the Figure 3 structure must be used.

```
<Reference Name="DSRLMOVE">
  <Restrictions>
    <DomainRestriction Domain="DSRL" />
    <TypeRestriction
        Type="MovementSensor" />
    <PropertyRestriction
        Property="InfluenceZone"
        Value="5"
        Operator="greaterThan" />
  </Restrictions>
</Reference>
```

Figure 3. XML Reference example

The **XML Subject** element and its Name attribute specifies and identifies the context entities collection which is the subject of the policy using a reference defined in the References section. A Subject element may have two children: (i) EvaluationTrigger XML element that contains a set of events generated by the subject which can trigger the evaluation of the policy and (ii) EvaluationCondition XML element that contains a set of conditions for which the policy will be enforced.

To specify that in the DSRLTEMP Reference context entities collection the policy should be evaluated when the measured temperature has changed and is equal to 22 degrees, Figure 4 structure is defined.

```
<Subject Name="DSRLTEMP">
  <EvaluationTriggers>
    <Trigger Event="TemperatureChangedEvent"/>
  </EvaluationTriggers>
  <EvaluationConditions>
    <Condition
        Property="TemperatureValueProp"
        Operator="equal"
        Value="22">
    </Condition>
  </EvaluationConditions>
</Subject>
```

Figure 4. XML Subject example

The **XML Target** element depends on policy type. In the metrics constraint policies case, the Target element represents a collection of entities for which the goals specified by the PolicyGoals XML element applies. It uses a Reference context entities collection defined in the References section to identify the target of the policy.

The PolicyGoals are used by the context aware application to determine the action plan that has to be executed in order to enforce the policy. For action policies the Target element represents a collection of entities for which the actions specified by the Action XML element are directly executed in order to enforce the policy. In Figure 5 we present an example for a metrics constraint policy that must enforce a 22 degree Celsius temperature in the DSRL laboratory.

```
<Target Name="DSRLTEMP">
  <PolicyGoals>
    <Goal Property="TempValP" Value="22" />
  </PolicyGoals>
</Target>
```

Figure 5. XML Target example

The context policies are converted into SWRL rules [13] and evaluated using the RAP specific context model instance ontology representation. The SWRL rules are used to reason about specific context model instance ontology individuals in terms of ontology specific context model classes and properties. Rules are written in the form of an implication between an *antecedent* (body) and *consequent* (head). Both the antecedent and consequent consist of multiple atoms conjunctions.

3.3. Evaluation Results

In order to test the algorithm we have implemented a simulator that provides an editor in which an evaluation test case can be described by adding a timeline and the corresponding values for the context properties that change.

The policies are loaded from XML files and converted into SWRL rules using a policy conversion module. The resulting SWRL rules are injected into the DSRL specific context model ontology representation and evaluated using the Pellet reasoning engine [15]. While running the simulation, a policy monitor window shows which policies are not respected and should be enforced (Figure 6).

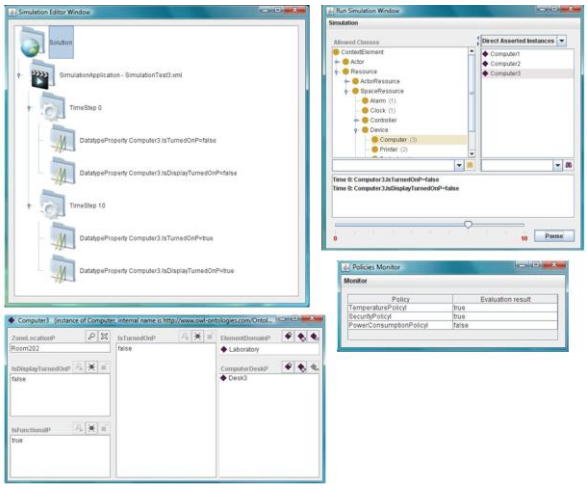

Figure 6. The policy evaluation simulator

In order to assess the performance of the context situation entropy evaluation and the equivalence classes determination, several tests with different input data of increasing complexity have been generated.

The first testing scenario determines the evaluation time for an increasing number of context policies processed all at once (parallel evaluation). For this scenario we started with a test having 3 policies as input, and then we gradually increased the number of policies up to 30 for the last test. The policies that were used had the same complexity (the corresponding SWRL rules contained approximately the same number of atoms in the antecedent). Each test had 20 runs and

an average was made in order to obtain the evaluation time for that test (Figure 7).

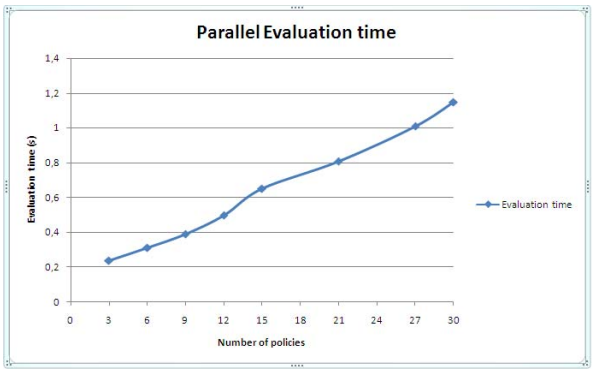

Figure 7. The performance of the context situation entropy parallel evaluation

The second testing scenario determines the evaluation time for an increasing number of context policies individually processed (sequential evaluation). The goal of this test scenario is to see how the system performance differs when performing evaluation of several policies at the same time compared to the situation when those policies are evaluated individually. For this scenario the evaluation time is computed as a sum of the evaluation time for each policy (Figure 8).

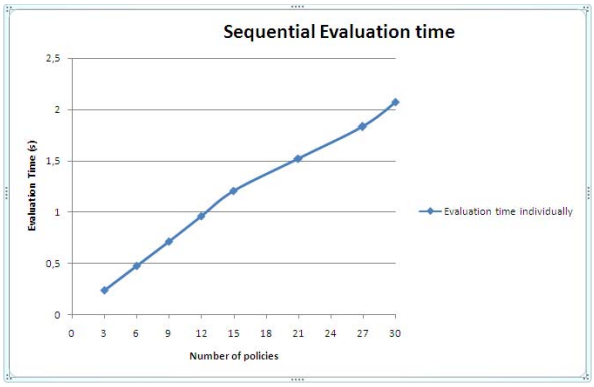

Figure 8. The performance of the context situation entropy sequential evaluation

The above presented scenario results show that the evaluation time varies linearly with the number of evaluated policies. By comparing the parallel and sequential results, it can be observed that it is more efficiently to evaluate all the policies at one time.

The last testing scenario shows how the policy evaluation depends on its complexity. The complexity of a policy is given by the number of atoms in the antecedent of the corresponding SWRL rule. For this scenario we started with a test having as input one policy with 4 atoms and we increased gradually the number of atoms up to 34 for the last test. Each test was done several times with a different policy and an average was computed.

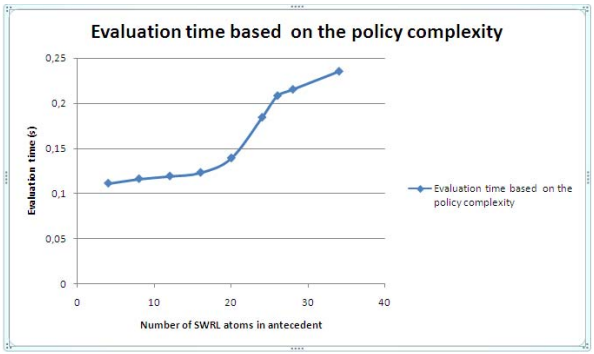

Figure 9. The performance of the policy evaluation for increasing the policy complexity

By looking at the chart (Figure 9) we can draw the conclusion that for a number of atoms smaller than 20 the evaluation time increases slowly. When the complexity of the policy increases above 20 atoms the evaluation time increases much faster.

4. Conclusions

In this paper we proposed a generic self-healing algorithm that can be used to automatically detect, diagnose and repair the problems that may appear during the context adaptation processes. The algorithm is based on the concepts of context situation entropy and equivalent context situations together with a set of self-healing principles defined using the set, situation calculus and information system theories. The self-healing property is enforced by monitoring the system execution environment in order to: (i) evaluate the degree of fulfilling the context policies for the current context situation (context situation entropy), (ii) obtain the equivalent context situations and (iii) determine the actions that have to be executed for keeping the system in consistent functional states. The self-healing algorithm is validated using our RAP context model for representing the context information and the DSRL laboratory as a smart space scenario.

For further development we intend to define, formalize and implement the action selection phase of the self-healing algorithm using reinforcement learning algorithms.

5. References

[1] Daniel A. Menascé and Jeffrey O. Kephart, "Autonomic Computing", *IEEE Internet Computing*, vol. 11, no. 1, 2007, pp. 18-21.

[2] Ioan Salomie, Tudor Cioara, Ionut Anghel and Mihaela Dinsoreanu, "RAP - A Basic Context Awareness Model", *Proceedings of 4th IEEE International Conference on Intelligent Computer Communication and Processing*, Cluj-Napoca, Romania, 2008, pp. 315-318.

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

[4] Jared Saia and Amitabh Trehan, "Picking up the Pieces: Self-Healing in reconfigurable networks", *Proc. of IEEE International Parallel and Distributed Processing Symposium*, 2008, pp. 1 – 12.

[5] Salomie I., Moga A. and Dinsoreanu M., "Enhancing Web Service Composition with Self-healing Facilities", *2007 WSEAS Trans. on Information Science and Applications*, 2007, pp. 42-50.

[6] P. Champrasert and J. Suzuki, "SymbioticSphere: A Biologically-Inspired Autonomic Architecture for Self-Adaptive and Self-Healing Server Farms", *Proc of the International Symposium on a World of Wireless, Mobile and Multimedia*, 2006, pp. 474-480.

[7] Mehdi Amoui, Mazeiar Salehie, Siavash Mirarab and Ladan Tahvildari, "Adaptive Action Selection in Autonomic Software Using Reinforcement Learning", *Proc. of the Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08)*, 2008, pp. 175-181.

[8] E. Pournaras, M. Warnier and F. Brazier, "Using intelligent agents for self-adaptation and self-optimization of energy consumption in power networks", *Proc. of 1st International Workshop on Agents for Autonomic Computing, in conjunction with ICAC 2008*, 2008.

[9] Rajarshi Das, Jeffrey O. Kephart, Charles Lefurgy, Gerald Tesauro, David W. Levine and Hoi Chan, "Autonomic multi-agent management of power and performance in data centers", *Proc. AAMAS2008*, 2008, pp. 107-114.

[10] F. Pirri and R. Reiter, "Some contributions to the metatheory of the Situation Calculus". *Journal of the ACM*, pp. 325–361, 1999.

[11] Jech Thomas, "Set Theory: Third Millennium Edition", *Springer Monographs in Mathematics*, Springer-Verlag, ISBN 978-3-540-44085-7, 2003, p. 642.

[12] Galliers R.D., Markus M.L. and Newell S. (Eds), "Exploring Information Systems Research Approaches", *NY: Routledge*, ISBN-10 041577196X , 2007.

[13] Ian Horrocks et. All, "SWRL: A Semantic Web Rule Language", *W3C Standard*, http://www.w3.org/Submission/SWRL/

[14] DSRL, *Distributed Systems Research Laboratory*, Technical University of Cluj-Napoca, dsrl.coned.utcluj.ro.

[15] Evren Sirin, Bijan Parsia, Bernardo C Grau, Aditya Kalyanpur and Yarden Katz, "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.