2009 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing

A Policy-based Context Aware Self-Management Model

T. Cioara, I. Anghel, I. Salomie and M. Dinsoreanu
Computer Science Department, Technical University of Cluj-Napoca
Cluj-Napoca, Romania
{tudor.cioara, ionut.anghel, ioan.salomie, mihaela.dinsoreanu}@cs.utcluj.ro

***Abstract*—This paper proposes a generic policy based self-management model that can be used to automatically detect and repair the problems appeared during the context adaptation processes. To successfully capture and evaluate the dynamic rules that govern the context aware adaptation processes we have defined an generic context policy representation model and its associated reasoning language conversion model for run-time evaluation. To evaluate the run-time degree of respecting the context policies we define and formalize the concept of context entropy. The context information is modeled in a system programmatic manner using both the set based and ontology based representations provided by our RAP (Resources, Actors, Policies) context model. The context model artifacts are generated and administrated at run time by a management infrastructure based on BDI (Believe, Desire, Intentions) agents. The model was tested and validated within the premises of our Distributed Systems Research Laboratory smart environment.**

Keywords—context awareness; self-management; context policy; context entropy; agents.

I. INTRODUCTION AND RELATED WORK

An important challenge in developing context aware systems is the complexity and dynamic nature of their execution environment which 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]. There is an evident need for rules to automate / govern the management processes and to define strictly regulated access to the context aware system various resources. The use of policy models that take into account the dynamicity and complexity of the new context aware systems are crucial to ensure context awareness. To make context information accessible and to support reasoning for evaluating the context governing rules, the context information should be modeled in a uniform way, so that it can be unambiguously interpreted and further combined.

In this paper we propose a generic policy based self-management model that can be used to automatically detect, diagnose, and repair the problems appeared during the context adaptation processes. To fulfill our objective we have defined a generic policy model that successfully capture and represent the dynamic rules that govern the context aware adaptation processes. The context information is modeled in a system programmatic manner using both the set based and ontology based representations provided by our RAP context model [2]. The policies are converted into SWRL [3] rules that reason about the RAP specific context model instance ontology individuals in terms of ontology specific context model classes and properties. We have also defined and formalized the concept of context *entropy* to evaluate the run-time degree of respecting the context policies. In *our vision*, the context entropy in the context aware computing field, should measures the level of system internal and external (environment) disorder by evaluating the degree of fulfilling the context policies.

For *context representation*, generic models that aim at accurately describing the system execution context in a programmatic way are proposed [4]. Key-Value models represent context information using a set of attributes and their associated values [5]. Markup models enable structuring context information into a hierarchy. Tags describe context attributes and associated values [6]. Object models structure context into object classes and their implicit relationships [7]. In [8], the concept of multi-faceted entity is defined and used to model the set of context properties. A facet represents the effective values of context properties to which the context sensitive application has access. The main drawback of this approach is the lack of semantic information encapsulated in the facet concept. As a result, inferring new context related knowledge is difficult. An original approach to the context modeling problem is the use of parametric state machines for representing a context aware system [9]. The context is modeled using context functions that modify the context aware system's state. The main disadvantage of this approach is the complexity of the parametric state machine associated to real systems. The use of ontologies and XML representations is a new direction for context modeling. The context properties are represented as ontological concepts during design time and instantiated with run-time sensor captured values [10] [11]. The main disadvantage of this approach is the high degree of inflexibility determined by the human intervention in the context representation phase. O'Connor proposes in [12] the construction of a system situation space where system execution context is represented as a group in this space. Using learning algorithms, the system may infer the action to be executed when a new situation appears by placing it in a situation space group.

978-0-7695-3964-5/09 $26.00 © 2009 IEEE
DOI 10.1109/SYNASC.2009.28

Regarding the *context aware policies*, the research was focused on policy specification, evaluation and enforcement. As mentioned in [13], there are some properties that any language used for specifying policies should fulfill: (i) expressiveness - supporting a large set of policy requirements, (ii) simplicity - the policies should be easily defined, (iii) enforceability - the policies should be easily implemented, (iv) scalability - ensure adequate performance, and (v) analyzability - allow reasoning. The Ponder language is a declarative, object-oriented language that allows for the specification of security policies with role-based access control, as well as management policies for describing what should be done when specific events occur within the system [14]. The adoption of ontologies in policy representation brings some considerable advantages [13]. Ontologies simplifies the problem of managing the behavior of complex environments, allowing to represent entities and behaviors at multiple levels of abstraction. However, Semantic web languages used for ontology representation still present a complex syntax. Also, ontology-based policy specification can be difficult to implement because high level specification of policies can make complicated their enforcement. The KAoS policy management framework uses ontological representation and reasoning in order to specify, enforce and analyze policies [15]. The framework provides generic and application/platform specific functionality. The generic functionality includes creating and managing the set of core ontologies and also storing, de-conflicting and enforcing policies. KaoS can be extended by defining new ontologies for describing application-specific entities.

The rest of the paper is organized as follows: in Section 2, the proposed context aware policy model is described; Section 3 details the framework used to evaluate and test our policy model; Section 4 shows the evaluation results in the premises of our Distributed Systems intelligent laboratory (DSRL) environment [16], while Section 5 concludes the paper and presents the future work.

II. THE CONTEXT AWARE POLICY MODEL

Let's consider the real world context 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, lesson hints services, print services, question-based services, etc.). A sensor network captures information regarding students location 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.

In the next sections we give a brief overview of the RAP context model, we show how the context aware policies can be defined and we present the concept of context entropy used to evaluate the run-time degree of respecting the context policies.

A. The RAP context model

To represent a closed environment context information in a programmatic manner we have used our RAP context model [2]. This model defines two types of context information representation: *set based* and *ontology based*.

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.

A *context resource* is a physical or virtual entity which generates and / or processes context information.

A *context actor* represents a physical or virtual entity that interacts directly with the context or uses the context resources to fulfill its needs. The actor is a context information generator, has a unique identity and can be annotated with semantic information.

A *context policy* represents a set of rules that must be followed by the actors or resources located in the context influence zone.

In the ontology based representation the relationships between the context model sets are modeled in a general purpose context ontology core (see Fig. 1). The domain specific concepts are represented as sub-trees of the core ontology by using *is-a* type relations.

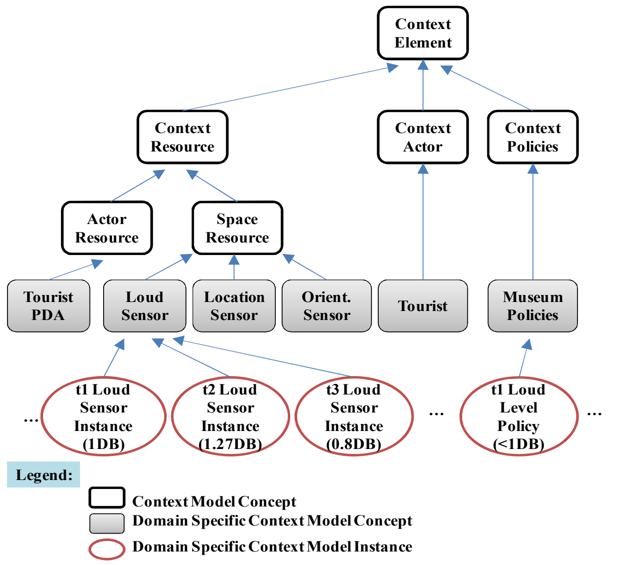

Figure. 1. The context model ontology representation

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 types of representing the context (set based and ontology based) are equivalent and need to be kept synchronized. The set based context model is used to evaluate the conditions under which the context management agents should execute self-* processes in order to enforce the autonomic properties (self-configuring, self-healing, self-optimizing and self-protection). The ontology based model will be used for reasoning purposes.

In order to provide an accurate representation of the real world context, the following context representation artifacts are defined: *specific context model*, *specific context model* *instance* and *context – actor instance*. The specific context model C_S is obtained by mapping the context model onto different real contexts and populating the sets with real context specific elements: $C_S = <R_S, A_S, P_S>$. 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*: $C_{SI} = <R_{SI}, A_{SI}, P_{SI}>$. The specific context model represents the context situation to which a pervasive application 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 *t*: $CI_a^t = <R_a^t, a, P^t>$.

B. The context aware policy representation

For the RAP context model we have identified 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 *metric constraints policies* and *action policies*.

The *metric constraints policies* are defined for the set of passive context resources in order to impose some restrictions to the captured context specific data. The context aware application needs to automatically determine what actions or plans of actions should be executed in order to enforce and maintain these constraints. The *action policies* are defined for the context elements that can directly modify the context state (active resources or actors) and specify the actions that should be performed to satisfy the policy constraints.

The 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 has a unique name that can be accessed from all the other parts of the policy. The reference context entities collection is defined by applying several restrictions. All the elements that satisfy the restrictions will be considered as part of the collection. 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 location to create the Reference context entities collection. The *type restriction* (defined by the TypeRestriction XML element and its Type attribute) restricts the 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.

As an 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 XML policy representation shown in Fig. 2 must be used.

```
<Reference Name="DSRLTEMP">
  <Restrictions>
    <DomainRestriction Domain="DSRL"/>
    <TypeRestriction
                Type="TemperatureSensor"/>
    <PropertyRestriction
        Property="InfluenceZone" Value="10"
        Operator="greaterThan" />
  </Restrictions>
</Reference>
```

Figure 2. An XML Reference example

The *XML Subject* element and its Name attribute specifies and identifies the context entities collection that is the subject of the policy using a reference defined in the References section. The Subject element can 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.

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

Figure 3. An XML Subject example

Fig. 3 details the structure needed 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.

The *XML Target* element depends on policy type. In the metric 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.

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

Figure 4. A metric constraint policy XML Target example

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 Fig. 4 we detail an example for a metrics constraint policy that must enforce a 22 degree Celsius temperature in the DSRL laboratory.

C. The context aware policies evaluation

The context policies are converted into SWRL rules 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.

Obtaining the SWRL antecedent. The process of obtaining a SWRL rule antecedent from the XML context policy description consists from two phases: (i) the representation of the SWRL atoms used for identifying the specific context model instance ontology individuals involved in the reasoning process and (ii) the conversion of the XML policy evaluation condition in SWRL.

```
XML Policy:
<Subject Name="referenceName">

SWRL atom:
ContextElement(?referenceName)
```

Figure 5. The ContextElement SWRL atom

The ontology individuals identification. The *first step* for this phase is to impose the condition that all the individuals involved in the reasoning process are instances of the ContextElement ontology class. Fig. 5 shows how the ContextElement SWRL atom is obtained.

```
XML Policy:
<TypeRestriction Type=" TypeAttributeValue"/>

SWRL atom:
TypeAttributeValue(?referenceName);
```

Figure 6. Converting the TypeRestriction XML element

The *second step* is to convert the set of restrictions specified by the Reference XML element into SWRL atoms. The TypeRestriction element is transformed in SWRL by mapping the value of the Type attribute as SWRL predicate.

```
XML Policy:
<PropertyRestriction
   Property="PropertyAttributeValue"
   Value=" ValueAttributeValue"
   Operator=" OperatorAttributeValue"/>

SWRL atoms:
PropertyAttributeValue(?referenceName,
                             ?elemPropValue) ^
swrlb:OperatorAttributeValue(?elemPropValue,
                        ValueAttributeValue)
```

Figure 7. Converting the PropertyRestriction XML element

The TypeAttributeValue must be a class into the specific context model instance ontology (see Fig. 6).

Next the PropertyRestriction element is converted into SWRL atoms using the values of its Property, Operator and Value attributes (see Fig. 7). The PropertyAttributeValue is the name for a property associated with the class of the ?referenceName individual. The OperatorAttributeValue is the the name of a valid SWRL operator. The ValueAttributeValue must have a valid type accepted by SWRL.

The *forth step* is to map the XML DomainRestriction element into SWRL using the value of the Domain (see Fig. 8). ElementDomainProp is an OWL property associated with the ContextElement class while DomainNameProp is an OWL property associated with the Domain class from the ontology.

```
XML Policy:
<DomainRestriction
                 Domain="domainAttributeValue"/>

SWRL atoms:
ElementDomainProp(?referenceName, ?domain)  ^
DomainNameProp(?domain, domainAttributeValue)
```

Figure 8. Converting the DomainRestriction XML element

The evaluation conditions. The EvalutationConditions XML element is a child of the Subject XML element and contains a set of conditions for the evaluation process (represented by the Condition element). If a condition is broken the context aware application executes actions in order to enforce the policy. The Condition element is transformed into SWRL similarly with the PropertyRestriction element (see Fig. 9).

```
XML Policy:
<Condition Property=" PropertyAttributeValue"
           Operator=" OperatorAttributeValue"
           Value=" ValueAttributeValue">
</Condition>

SWRL atoms:
PropertyAttributeValue(?referenceName,
                              ?elemPropValue)  ^
swrlb:OperatorAttributeValue(?elemPropValue,
                          ValueAttributeValue)
```

Figure 9. Converting the EvalutationConditions XML element

Obtaining The SWRL Consequent. The specific context model contains a Policy class which holds references to the all XML context policies defined for a specific context model. The number of Policy class individuals in the specific context model instance ontology must mach the number of context policies represented in XML. The Policy class has two main attributes: Name and EvaluationResultProp.

```
SWRL atom:
EvaluationResultProp(PolicyNameAttributeValue,
                                          true)
Example: temperature policy
XML Policy:
<MetricConstraintPolicy
                        Name="TemperaturePolicy">
SWRL atom:
EvaluationResultProp(TemperaturePolicy, true)
```

Figure 10. Converting the EvalutationConditions XML element

The class Policy Name attribute value must always match the value of the XML Policy element Name attribute for all the specific context instance ontology individuals. As an example having a policy with the value of its XML Policy Name attribute TemperaturePolicy implies that an instance of the Policy class in the ontology with the name TemperaturePolicy must exist (see Fig. 10).

The class Policy EvaluationResultProp attribute value uses Boolean information regarding the degree of fulfilling the policy. After executing the SWRL rules corresponding to the XML policies, this attribute is set to true for all the individuals that correspond to XML broken policies.

D. *The Context 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 (E_s) and its associated threshold (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 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 T_E.

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

$$P_{eval} : (s,p) \rightarrow \{0, 1\}$$
$$E_S : P \rightarrow \mathbb{N},\ E_S = \Sigma_{p \in P} P_{eval}(s,p) \tag{1}$$

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 (see relation (2)).

$$E_s * E_{s++} < (T_E)^2 \tag{2}$$

This product is an invariant for the context aware system self-healing property, meaning that an increase of the system entropy over the threshold must not last more than one step.

III. The Policy Management Framework

In the next sections we present a brief overview of the RAP context model agent-based management infrastructure and describe the infrastructure usage for managing and evaluating the context aware policies.

A. *The RAP context model management*

The context model management infrastructure layer is based on four intelligent, cooperative BDI type agents [17]: *Context Model Administering Agents, Context Interpreting Agents, Request Processing Agents* and *Execution and Monitoring Agents*.

The *Context Model Administering Agent (CMAA)* is the specific context model manager. Its main goal is the synchronization of the context model specific artifacts with the system execution environment. This agent is also responsible for negotiating processes that take place when an actor or resource is joining the context. The *Context Interpreting Agent (CIA)* semantically evaluates the information of a context instance and tries to find the context instance "meaning" for the pervasive application. The *Request Processing Agent (RPA)* processes the actor requests. This agent identifies and generates the action plans that must be executed for serving an incoming request. The RPA agent uses the specific context model instance to identify the proper plan to be executed by the *Execution and Monitoring Agent (EMA)* or for generating a new plan. EMA processes the plans received from the RPA agent and executes every plan action using the available services. After mapping action plans onto services, a plan orchestration (smart workflow) which can be executed using transactional principles is obtained.

The context management infrastructure agents are implemented using the Java Agent DEvelopment Framework platform, a middleware, based on the agents paradigm, for the development and run-time execution of P2P applications [18].

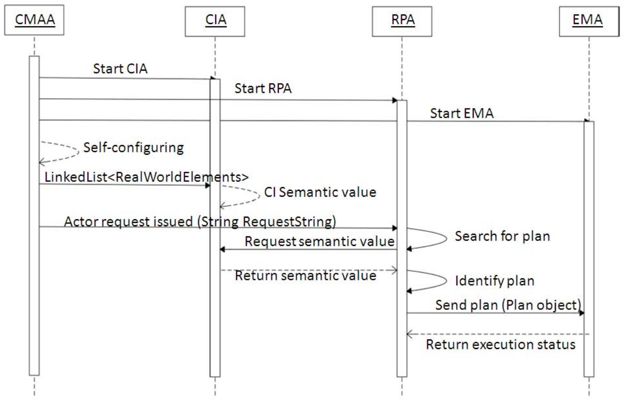

Figure 11. The RAP context model agent based management

Fig. 11 shows how the four context management infrastructure agents communicate and coordinate their actions in order to manage the context representation process.

When the middleware is deployed, CMAA is the first running agent. It instantiates the CIA, RPA and EMA context management agents and sends them the real world context representation. Also CMAA agent keeps the context model synchronous with the real world context.

CIA agent uses reasoning algorithms and the context model ontological representation provided by CMAA agent to infer high level context information. When a context actor issues a request, the RPA agent is activated. It uses the context information provided by CIA agent, to construct / find a plan that can be executed as a response to the actor's request. The plan is then forwarded for execution to the EMA agent.

B. *The context policies management*

The policy management is being done by the CMAA agent which has three cooperative sub-agents as main components: *Simulation Agent*, *Ontology Repository Agent and Policy Evaluation Agent* (see Fig. 12). Each agent uses Jena [19] in order to manipulate a local ontology. Jena is a Java framework for building Semantic Web applications which provides support for RDF, RDFS and OWL and includes a rule-based inference engine.

The *Simulation Agent* consists of a *Simulation Editor* which is used to create a simulation. In this simulation, users are allowed to add simulation times and for each simulation time the new values for the properties of the individuals in the ontology that change at that moment may be added. The *Simulation Manager* can be used to store the simulation in an XML file and later load the data for simulation. For each simulation time the *Simulator* module modifies the values of the properties in the ontology accordingly. The changes in the ontology are then propagated to the *Ontology Repository Agent* by the *Ontology Synchronization Module*. During simulation the properties of the individuals in the ontology can be visualized.

The *Policy Evaluation Agent* loads the policies from an XML file using the *Policy Loader* module. The *Policy Converter* takes these policies and converts them into SWRL rules. These rules are injected into the ontology by the *Policy Injector* module. Each change in the ontology will trigger the evaluation of the policies. The *Policy Evaluation Module* evaluates the polices using the Pellet reasoning engine, an open source Java reasoner for OWL 2 DL with Jena support [20]. In this manner the *Policy Evaluation Agent* decides which policies are not respected and therefore should be enforced. During the agent runtime, the policy monitor window shows which policies are not respected.

The *Ontology Repository Agent* loads the ontology from an OWL file using the *Ontology Loader* module. Using the *Central Ontology Synchronization Module* it receives updates to the ontology from agents and propagates these changes to the central ontology that it maintains and also to the other agents. This agent is also responsible for distributing the central ontology. When an agent starts, it sends a message to the *Ontology Repository Agent* requesting the ontology. As a response, the *Ontology Repository Agent* sends the ontology to the requesting agent in OWL format.

IV. EVALUATION RESULTS

In order to test the algorithm we have implemented the 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. While running the simulation, a policy monitor window shows which policies are not respected and should be enforced.

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

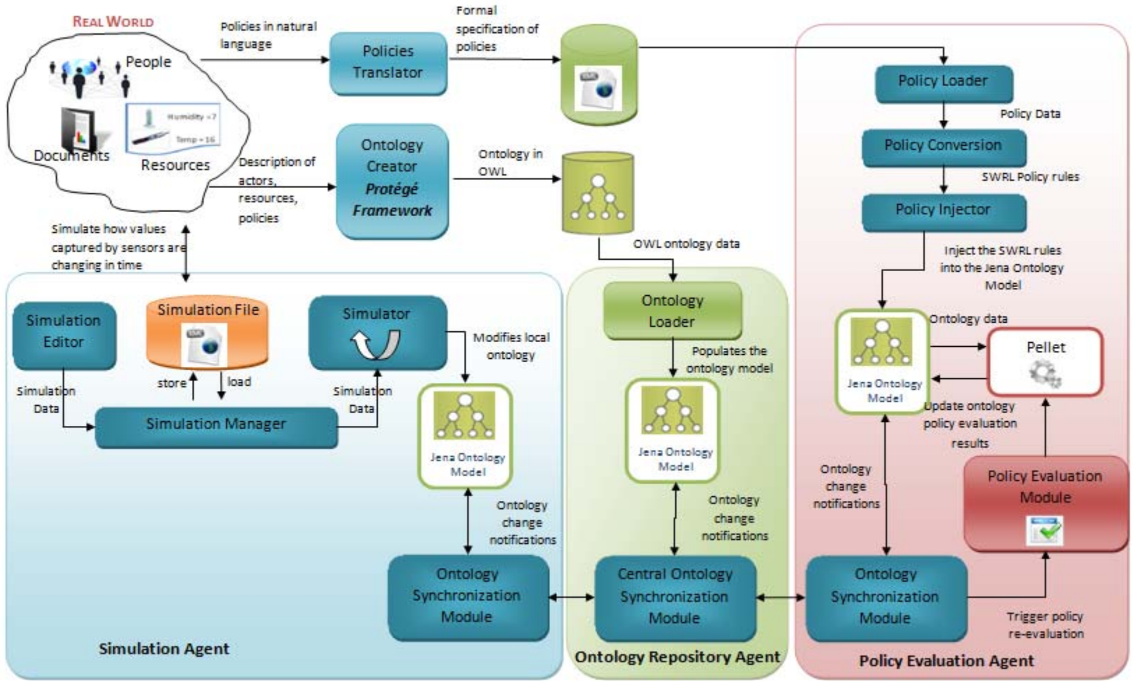
REAL WORLD
People
Documents
Resources
Humidity = 7
Temp = 16
Policies in natural language
Policies Translator
Formal specification of policies
Description of actors, resources, policies
Ontology Creator Protégé Framework
Ontology in OWL
Simulate how values captured by sensors are changing in time
OWL ontology data
Policy Loader
Policy Data
Policy Conversion
SWRL Policy rules
Policy Injector
Inject the SWRL rules into the Jena Ontology Model
Simulation Editor
Simulation File
Simulator
Modifies local ontology
store
load
Simulation Data
Simulation Manager
Simulation Data
Jena Ontology Model
Ontology Loader
Populates the ontology model
Jena Ontology Model
Jena Ontology Model
Ontology data
Pellet
Update ontology policy evaluation results
Ontology change notifications
Ontology change notifications
Ontology change notifications
Policy Evaluation Module
Ontology Synchronization Module
Central Ontology Synchronization Module
Ontology Synchronization Module
Trigger policy re-evaluation
Simulation Agent
Ontology Repository Agent
Policy Evaluation Agent

Figure 12. The Context Model Administering Agent architecture

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 (see Fig. 13).

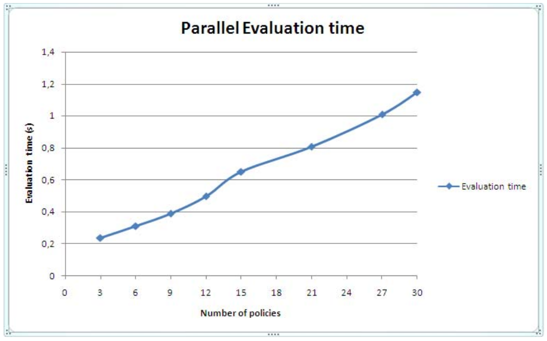

Figure 13. The performance of the context situation entropy parallel evaluation

The second testing scenario determines the evaluation time for an increasing number of context policies individually processed (sequential evaluation).

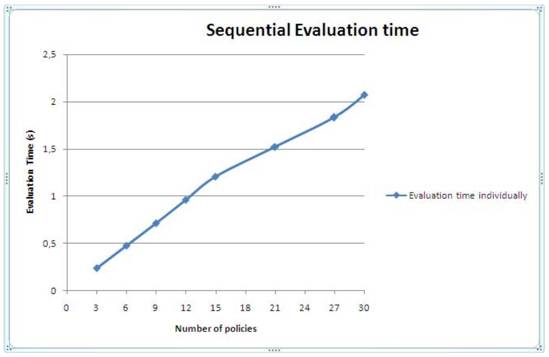

Figure 14. The performance of the context situation entropy sequential evaluation

The goal of this test scenario is to see how the system performance differs when performing all at once evaluation for a set of policies compared to the situation when these policies are evaluated individually. For this scenario the evaluation time is computed as a sum of the evaluation time for each policy (see Fig. 14). The 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 efficient 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.

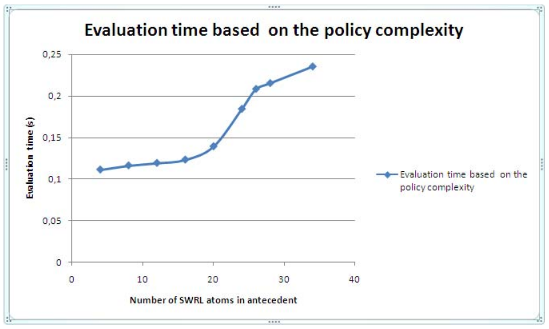

Figure 15. The performance of the policy evaluation for increasing the policy complexity

By looking at the chart (Fig. 15) 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.

V. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed a generic policy based self-management model that can be used to automatically detect and repair the problems appearing during the context adaptation processes. The policies are generically represented in XML and converted into the SWRL reasoning language for run-time evaluation. We have also defined the concept of context entropy to evaluate the run-time degree of fulfilling the context policies for a specific context situation. We have chosen the BDI type agents as processing elements for the self-management model due to their reactivity, collaboration, inference and adaptability properties. The policy based self-management model 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 corrective action selection phase as a response for broking the context policies using a reinforcement learning approach.

REFERENCES

[1] D. A. Menascé and J. O. Kephart, "Autonomic Computing", IEEE Internet Computing, vol. 11, no. 1, 2007, pp. 18-21.
[2] I. Salomie, T. Cioara, I. Anghel and M. Dinsoreanu, "RAP - A Basic Context Awareness Model", In Proc. Of 4th IEEE Int. Conf. on Intelligent Computer Communication and Processing, ISBN: 978-1-4244-2673-7, 2008, pp. 315- 318.
[3] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "Swrl: A semantic web rule language combining owl and ruleml", Acknowledged W3C Member Submission, May 21, 2004, NRC Publication Number: NRC 48058.
[4] D. Fournier, S. Ben Mokhtar and N. Georgantas, "Towards Ad hoc Contextual Services for Pervasive Computing", Proceedings of the 1st workshop on Middleware for Service Oriented Computing, ISBN:1-59593-425-1, 2006, pp. 36 – 41.
[5] K. M. Anderson, F. A. Hansen and N. O. Bouvin,"Templates and queries in contextual hypermedia", In Proceedings of the Proceedings of the seventeenth conference on Hypertext and hypermedia, Denmark, 2006, pp. 99 – 110.
[6] D. Raz, A. T. Juhola, J. Serrat-Fernandez and A. Galis, Fast and Efficient Context-Aware Services, Wiley Series on Communications

Networking & Distributed Systems, ISBN-13: 978-0470016688, 2006, pp. 5-25.

[7] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann and W. Retschitzegger, ”Context-awareness on mobile devices - the hydrogen approach”, In HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences, USA, 2003, pp. 292.

[8] A. Rarau, K. Pusztai and I.Salomie, “MultiFacet Item based Context-Aware Applications”, Inernational Journal of Computer and Information Sciences 3(2), , 2006, pp. 10-18.

[9] I. Y.L. Chen, S. J.H. Yang and J. Zhang, “Ubiquitous Provision of Context Aware Web Services”, IEEE International Conference on Services Computing, ISBN: 0-7695-2670-5, 2006, pp.60-68.

[10] K. C. Lee, J. H. Kim and J. H. Lee, “Implementation of Ontology Based Context-Awareness Framework for Ubiquitous Environment”, Int. Conf. on Multimedia and Ubiquitous Engineering, April 2007, pp. 278 – 282.

[11] H. Chen, T. Finin and A. Joshi, ”The SOUPA Ontology for Pervasive Computing”, In Ontologies for Agents: Theory and Experiences, July 2005, Birkhäuser Basel, pp. 233-258.

[12] N. O’Connor, R. Cunningham and V. Cahill, “Self-Adapting Context Definition”, First Int. Conference on Self-Adaptive and Self-Organizing Systems , 2007, pp. 336-339.

[13] G. Tont, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri and A. Uszok, ”Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder”, In book The SemanticWeb, ISWC 2003, pp. 419-437.

[14] N. Damiano, N. Dulay, E. Lupu and M. Sloman, “The Ponder Policy Specification Language”, Proc. Policy 2001:Workshop on Policies for Distributed Systems and Networks, UK, Jan. 2001, Springer-Verlag LNCS 1995.

[15] A. Uszok, J. M. Bradshaw, J. Lott, M. Breedy, L. Bunch, P. Feltovich and M. Johnson, ”New Developments in Ontology-Based Policy Management: Increasing the Practicality and Comprehensiveness of KaoS”, In IEEE Workshop on In Policies for Distributed Systems and Networks, 2008, pp. 145-152.

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

[17] J. Thangarajah, L. Padgham and J. Harland, ”Representation and reasoning for goals in BDI agents”, Proceedings of the twenty-fifth Australasian conference on Computer science - Volume 4, 2002, pp. 259 – 265, ISSN:1445-1336.

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

[19] Jena Semantic Web Framework, http://jena.sourceforge.net/.

[20] Pellet Reasoner, http://clarkparsia.com/pellet/.