Journal of Theoretical and Applied Information Technology
31st March 2012. Vol. 37 No.2
© 2005 - 2012 JATIT & LLS. All rights reserved·

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

POLICY-BASED APPROACH TO DETECT AND RESOLVE POLICY CONFLICT FOR STATIC AND DYNAMIC ARCHITECTURE

[1]ABDELHAMID ABDELHADI MANSOR, [2]WAN M.N. WAN KADIR, [3]TONI ANWAR, [4]HIDAYAH ELIAS

[1]Department of Computer Sciences, Faculty of Mathematical Sciences, University of Khartoum, Sudan

[2,3,4] Software Engineering Department, Faculty of Computer Science and Information System,

UniversitiTeknologi Malaysia, Malaysia

E-mail: [1]abhamidhn@gmail.com, [2]wnasir@cs.utm.my, [3]tonianwar@utm.my,[4]shahlida@gmail.com

ABSTRACT

Current research efforts are being directed to commit with the long-term view of self-management properties for wireless telecommunications. One of the key approaches that have been recognized as an enabler of such a view is policy-based management. Policy-based management has been mostly acknowledged as a methodology that provides flexibility, scalability, adaptability and support to automatically assign network resources, control Quality of Service and security, by considering administratively specified rules. The hype of policy-based management was to commit with these features in run-time as a result of changeable network conditions resulting from the interactions of users, applications and existing resources. To detect and resolve potential static and dynamic conflicts between the rules and configurations from different administrative domains, a policy-based manager coordination PobMC framework which is based on Event- Condition-Action (ECA) is proposed in this paper. The framework is responsible of potential conflicts between applications included in the system. Furthermore, the paper provides a guideline to avoid policy conflicts between different domains configurations, and resolve the conflicts during runtime. PobMC depends on, static and dynamic analysis to reduce potential conflicts. The paper briefly outlines some of the most prominent research issues that have been discussed for large-scale adoption of policy-based systems. Then we present a description of the conflict detection and resolution algorithms that achieves the automated enforcement of obligations for resource-handling based on management policies.

Keywords—*Policy-Based Management, Policy Conflict Analysis, Dynamic Conflict, Static Conflict*, *Adaptive Software Architecture*

1. INTRODUCTION

Policy-based management has been mostly acknowledged as a methodology that provides flexibility, scalability, adaptability and support to automatically assign network resources, control Quality of Service and security, by considering administratively specified rules [1]. The wireless connection must be kept from reaching the congestion point, since it will cause an overall channel quality to degrade and loss rates to rise, leads to decrease of performance and some disconnection problems [2]. Policy-based approach has been recognized as a suitable approach to manage and improve content distribution networks and distributed systems. However, ubiquitous computing systems and autonomic systems still limited to determine the impact of policy changes on system behavior, due to the complexity of managed system scale. Furthermore, current policy-based approaches still have some limitations to adapt business and user level policies into enforceable system level policies. The hype of policy-based management was to commit with these features in run-time as a result of changeable network conditions resulting from the interactions of users, applications and existing resources. Obviously there is a limitation in developing policy-based management models that do not provide ensuing support for detecting and resolving conflicts. While a considerable attempt at static conflict detection has been presented in [3]. Moreover, current research has revealed that there is still a large class of policy conflict which simply cannot be determined statically.

The term policy is referred to as either procedure or protocol [4]. Policies can be used to "dynamically regulate the behavior of system components without changing code and requiring the consent or components cooperation". Policy also refers to the process of making important organizational decisions [5].

According to [6] a policy is represented as a means to control when a managed object moves to a new state. The subject of a policy specifies the human or automated managers to which the policies apply and which interpret obligation policies. The target of a policy specifies the objects and the actions to be performed. Domains are a means of grouping objects similar to file system directories [7]. The subject or target of a policy is expressed as a domain of objects and the policy applies to all objects in the domain; so a single policy can be specified for a group of policies. This helps to cater for large-scale systems in that it is not necessary to define separate policies for individual objects in the system, but rather for groups of objects.

Sloman[8] classified policies into access-control, obligation, goal-based and meta policies based on their purpose. Access control policies specify what actions entities can or cannot perform in a system. This type is further classified into authorization, delegation, and information filtering policies [9]. Authorization policies define what activities a member of the subject domain can perform on the set of objects in the target domain, delegation policies transfer access rights from one entity to another, and information filtering policies implement privacy by data obfuscation. For example, the location information of a mobile node can be reported with lesser accuracy to prevent the exact position from being revealed using information filtering policies.

Obligation policies specify what actions entities must or must not perform in a system [8]. In addition, they are used for fault, configuration and file system management. Goal-based policies are used to specify the final system state that should be reached from a given state. Meta-policies guide the behavior of the management system. Furthermore, they are used to modify policies, resolve conflicts dynamically and change various parameters of the management system.

Human error is one of the obstacles to accurate access-control policies; the policy authors who assign and maintain these policies are prone to make specification errors that lead to incorrect policies. Access-control policies consist of a set of rules that dictate the conditions under which users will bellow access to resources. These rules may conflict with each other.

Conflicts may arise within the set of policies or during the refinement process i.e. between the high-level goals and the implementable policies [3].For example, an obligation policy may define an activity a manager must perform but there is no authorization policy to permit the manager to perform the activity. The system must have to cater for conflicts such as exceptions to normal authorization policies. For instance, in a large distributed system there will be multiple human administrators specifying policies which are stored on distributed policy servers. Conflict detection between management policies can be performed statically for set of policies in a policy server as part of the policy specification process or at run-time [10, 11].

Conflicts between policies can be classified into four broad categories [12],they are: (i) Internal Policy Conflict, which occurs when there is incompatibility between policies assigned to a single role, (ii) External Policy Conflict, which occurs when combining of roles which in isolation of each other present no conflict, but contain policies which inco-existence are in conflict (iii) Policy Space Conflict, which occurs when more than one policy space manage the same set of subjects and attempt to enforce various and conflicting policies over them, and (iv) Role Conflict, which is expected when a user obtains a set of incompatible role assignments. Policy-based approach uses policies to govern their behavioral choices whilst satisfying the goals of the system, in addition to specify and enforce QoS management in distributed systems. Furthermore, it provides flexibility, adaptability and support to automatically assign network resources [13]. A policy-based management system must provide guarantees when multiple rules need to be concurrently enforced so that the system behavior is predictable. However, existing policy-based management systems based on Event Condition Action (ECA) rules do not contain specifications of actions required for reasoning and consequently do not provide guarantees which can lead to unpredictable system states [14].

Static and dynamic conflicts were considered as two classes of conflict which need to be understood and independently managed [12]. The distinction between these two classes is important; as detecting and resolving of conflict can be computationally intensive, time consuming and hence, costly and is most preferably done at compile-time. However, dynamic conflict is quite unpredictable, in that it may, or may not; proceed to a state of realized conflict. This class of conflict must be detected at run-time. To detect and resolve potential static and dynamic conflicts between the rules and configurations from different administrative domains, policy-based manager coordination (PobMC) framework, which is based on Event-Condition-Action (ECA) is proposed in this paper. The framework is responsible of potential conflicts between applications included in the system. Furthermore, the paper provides a guideline to avoid policy conflicts between different domains configurations, and resolve the conflicts during runtime. PobMC depends on, static and dynamic analysis to reduce potential conflicts.

Key questions that this paper will address are:

- How does the proposed algorithm compare with the existing approaches?
- How to detect and resolve conflicting policies during compile time and run time?
- What are the issues relating to detecting and resolving conflicts which can arise in the obligation policies and some ideas on how to refine high level goals?
- What is the role of obligation management and resource constraints in autonomous systems?
- How to manage conflict of obligations related to management of resource items (data and device), for self-adaptive systems?

This paper is organized as follows. Section two briefly describes the prominent works on policy-based approach. Section three presents the overview of (PobMC), and the proposed algorithms. Finally section four summarizes the paper and outline directions for future work.

2. RELATED WORKS

Most of the published works to-date with regards to policy conflict analysis and resolution concerns their implementation within single domain environments. There has been some works carried out to-date using various techniques such as static analysis to reduce potential errors [15-20], dynamic analysis to detect and resolve potential conflicts [16, 20-22], a verification of policy-conflict process [16, 20, 22], and system scalability [17]. This section presents an overview of this body of work.

Shiva [16] proposes an extended model of ECA called ECA-Post-condition (ECA-P) to enable developers and administrators to annotate actions with their effects. The ECA-P model allows deducing the action that may conflict based on conflicting post-condition; the framework also uses static and dynamic conflict detection techniques to detect failure in policy execution by using post condition to verify successful completion of policy actions. However, policy actions may not execute to completion due to various reasons such as changing active space configuration, device and component failure or software errors.

Wu et al [18] introduce dynamic analysis mechanism to ensure consistency among the enforced policies. They use Event Calculus (EC) in their dynamic policy conflict analysis to detect and control the dynamic conflicts in trust services. However, their work does not take targets constraints into account, while some of these conflicts are caused by overlapped elements. Davy et al. [15] present an efficient policy selection process for policy conflict analysis to improve the performance depend on the nature of the relationships between deployed policies. Their process targets pre-deployment identification of potential conflicts between a modified or newly created policy and already deployed policies. They use a tree based data structure to reduce the number of comparisons and therefore reduce runtime complexity in subsequent iterations by maintaining a history of previous policy comparisons. Their conflict analysis algorithm initiates a relationship pattern matrix between candidate and deployed policies, and matches these patterns against a conflict signature. However, this approach is not intelligent and repeats over all deployed policies to ensure that the deployed policy does not cause a potential conflict. Also the algorithm is still limited to detect only conflicts that can be represented as relationships among policies.

In another related work [17] Davy et al. produce a policy conflict analysis approach makes extensive use of information models and ontologies to make it a flexible tool to analyze for conflict in a range of applications. Furthermore, they introduce a novel pre-analysis policy selection to reduce the number of more comprehensive policy analysis operations required. Similar to their previous work, they use heuristics and historical information from previous comparisons to eliminate group of policies from analysis. Moreover, they separate the definition of a policy conflict from the definition of the conflict analysis algorithm; thereby the approach is extensible and efficient. However, this algorithm needs further improvement; because it eliminates policies instead of refine them. Eliminating some policies does not achieve the system goals and reduce the scalability.

Ma et al. [23] propose conflict detection and resolution in workflow management systems (WFMSs) approaches to help workflow designers in constructing a flexible, consistent workflow authorization schema. In this work a new type of constraint, context constraint, is proposed since context constraints can meet the complicated requirements of security policies in WFMSs. Moreover, they define an effective set of rules to detect and resolution of static and dynamic conflict for authorization policies in WFMSs. Furthermore, they classify conflicts into two broad categories (i) policy-policy conflicts which occur when two or more authorization policies are considered incompatible, (ii) policy constraint conflicts which occur when the performance of two or more authorization policies will lead to situations that are prohibited by other constraints (e.g., separation-of-duty constraints) in the system. However, their work do not put into account conflicts in authorization policy itself, in addition to policies are considered to assign by different administrators.

Mohan et al [19] propose an attribute-based authorization framework that supports changing the rules and policy combination algorithm dynamically based on contextual information. The framework eliminate the need to re-compose the policies when the combination algorithm changes. Moreover, it provides a method to add and remove specialized policies dynamically, in addition to its capability to reduce the set of potential target matches, thus increasing the efficiency of the evaluation mechanism. Furthermore, to resolve the conflicts they use Policy Combination Algorithms (PCA). These algorithms take the authorization decision from each policy as input. However, in a highly dynamic environment these algorithms will lead to reduced performance.

Khakpour et al. [20] present analysis using Rebeca[24] which is an actor-based language for modeling concurrent asynchronous systems which allows to model the system as a set of reactive objects called rebecs interacting by message passing. In order to introduce a new classification of conflicts may occur during governing policies. They also propose Linear Temporal Language LTL [25], which express each type of conflicts and enable to automate detection of conflicts patterns to classify conflict types, thereby to automate a significant portion of policy analysis process. Moreover, they introduce a number of correctness properties of the adaptation process in the context of their models. Then, they use static analysis of adaptation policies in addition to model checking technique to verify those properties. Whenever an event which requires adaptation occurs, relevant managers are informed. However, the adaptation cannot be done immediately and when the system reaches a safe state, the manager switches to the new configuration. While their system includes many different managers each manager use set of policies to govern system sensors and actors. There may be more than event which are require adaptation occur simultaneously, this will reduce the system scalability.

Barron et al. [21] describe how conflicts between newly specified (deployed or updated) federation-level policies and previously deployed local/federation policies can be detected. They transformed the DEN-ng federated domain model into an OWL-DL representation which allows for additional domain specific semantics (entity relationships) to be added to the model, and can then be reasoned over to detect relationships and potential conflicts between policy pairs. Moreover, the main objective of this work is to apply the policy selection and conflict analyses concepts to federated domain environments. However, this work is only concerned with conflict detection aspects in both static and dynamic environments. But it does not take target constraints into account; while newly specified policies may restricted by resource constraints. Furthermore, they use information models while they are limited in the type of information, and do not suitable to model dynamic environment.

To the best of our knowledge, detecting conflicts statically are resolved by the user before the enforcement of policies. Few researchers concentrate on static analysis to reduce the potential errors. It is very important to use static analysis, because after a policy is compiled detected conflicts resolved by the user before generate the policy object file. Detecting conflicts among rules by matching these rules events and condition statically, to determine matching it is required to compare event symbols and types of the rules parameters. Furthermore, dynamic analysis during runtime is required since all rule conflicts cannot be detected during the static analysis done at the compilation stage. After that rules must be combined to use dynamic conflict technique to detect potential conflicts during run time. Very few works decouple the rule into policies but the classification of rules is not included explicitly. Decoupling of rules helps static detection technique to identify the conflicting rules by classifying them into groups. There are some approaches classify policies into groups, but no decoupling technique is used. The above bereaving shows that there are some works combine policies before execution in order to evaluate and resolve potential conflicts.

Furthermore, avoiding errors and conflicts between policies, and addressing the scalability issues when policies assign by more than one application remain as the main challenges of current research. Moreover, scalability of the system needs to be checked when different types of policies assigned by different administrators, to further improve of system performance and adaptability.

Our framework in this paper can be used as a flexible approach to detect and resolve errors and conflicts during compilation time. The flexibility of our approach can help us to handle conflicts between policies at runtime. Moreover, we have defined two algorithms to handle errors and conflicts statically and dynamically.

3. POLICY-BASED MANAGER COORDINATION FRAMEWORK (PobMC)

3.1 OVERVIEW OF PobMC

PobMC framework is implemented as an active area services. Figure 1 illustrates the architecture of (PobMC) framework. The framework contains a Self-coordinator component that coordinates the interactions among various components of (PobMC). Since policies created by the policy author, stored in the Policy Repository which is a simple database and the actions stored in the Action Library to be called when an event occurs.

The proposed framework[26] provides support for architectural adaptation for both behavioral and structural changes based on the policies that govern the system. In a conceptual perspective, the framework includes various components; a brief explanation of PobMC components is as follow:

Policy verification depends on decoupling of the adaptation logic from its functional logic (its business logic). Thus, adaptation layer can be verified independently from the actor layer provided. Policy verification verifies the action and purpose specified by the user; in PobMC we assume that what is stated by the user is correct.

The role of Context Monitor is to monitor operating environment, checking for structural and behavioral changes. The examples of operating environments include sensor and actor states, malfunction of devices or new devices, in addition to the number of working sensors and the states of non-working sensors. Collected information which stores in the views helps managers to govern system changes. Moreover, in this activity more information about managers and their states provided to help managers to coordinate their tasks.

The Context Monitor allows users to register and log in and query the system for resources using various APIs and receives requests. Moreover, checks if the detected event is allowed or denied based on the setup time information that it has received from the policy analyzer. If allowed, it also checks if there is an obligation mandated by the relevant rule. If yes, then the Context Monitor informs the Self-Coordinator (which is the obligation enforcement component in PobMC) of the obligation, the Self-Coordinator marks the resource item in the corresponding file, based on the resource type. Subsequently, the Self-coordinator informs the request Context Monitor on the ‘Allow’ or ‘Deny’ ruling, as applicable. The Context monitor then displays the permitted results to the user. In addition to the mentioned functions, Context monitor monitoring the execution of obligation over the runtime periods, since some obligations could be defined to take effect much later in time than the time of resource access.

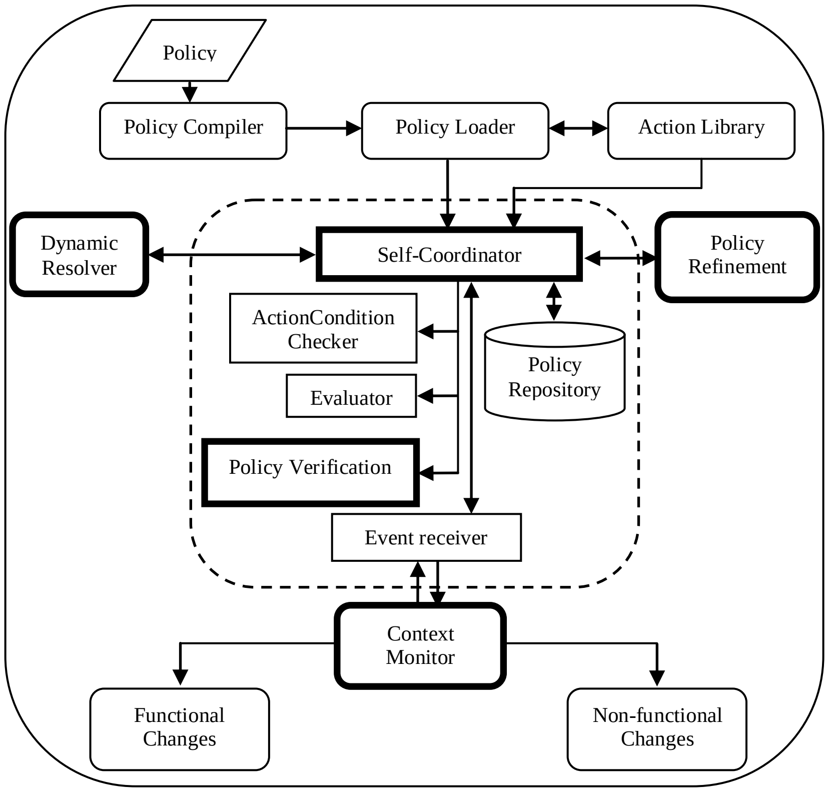

Figure 1. High level view of PobMC framework

Each policy checked first by Self-Coordinator before triggering execution of processing. Self-Coordinator represents the core of the system coordinates and checks all the activities during runtime. Moreover, any task taken by each process must be checked in this process in order to take the right decision. The self-coordinator determines the triggered rules, and uses the ActionCondition checker to test the action and rule condition expressions. If a condition evaluates to true the rule is added to the policy live list.

Once the static conflicts have been detected and resolved, the policy compiler compiles the policies including the resources constraints then generates a policy object file. The policy loader loads the generated object file into the Self-coordinator component before evaluated to detect and resolve potential dynamic conflicts.

A library of actions stored in Action Library that can be invoked from the action part of the policy rule. When an event occurs in a situation where condition is true, then the action is a call to a method in a library of actions where each action is annotated with a post-condition by the programmer. These post-conditions of the actions are used for conflict detection. Event Receiver is responsible for subscribing and receiving events since they occurred and have been detected by the Context Monitor. Then Event Receiver verifies the types of the parameters in the events and notifies the self-coordinator of the event occurrence along with the parameters.

3.2 POLICY CONFLICT DETECTION AND SOLUTION TECHNIQUES

Before we delve into finer details we declare some of the definitions and notations that we will use to define the proposed conflict management. We use a description that is similar to XACML standard. Each rule defines whether a resource access is allowed or denied, based on the kind of resource being accessed, the user role, the intent and intended action. In addition, each obligation on the system is mandated by a particular rule.

An event E is represented as $E = \{u, d, a, g\}$ where u is the user category to which the user (who is requesting to access the resource) belongs, each user of the system is assigned to one or more user categories, based on the user roles, d is a resource category to which the resource item belongs, a is the action that the user wants to perform on the resource, users may access the resource items to perform one or more of the specified set of actions. A representative set of actions could be {'View', 'Access', 'Modify', 'Remove'}. Each resource item that is to be privacy protected is assigned to a specific data category. The user and resource categories could be hierarchical, and g is the purpose for which the resource is being accessed, a representative set of purposes could be {'Shopping', 'Searching', 'Administrative'}.

Note that where the user is mapped to the user category and the data item is mapped the resource category, the user request maps internally to an event. The Condition is clause associated with some rules, which needs to be true, for the rule to 'Allow' access, the Permission is a binary predicate that can take one of the two values {'Allow', 'Deny'}, identifying whether the user request is allowed or denied, and the Obligation is a task that specifies what action needs to be performed after the user request is fulfilled. Besides introducing obligations to safeguard the integrity of the system, it is targeted to ensure that all the commitments with respect to various business policies are fulfilled. The nature of the tasks enables the classification of obligations in the resource or data-handling domain into the following types, (i) Notification-related, which requires the system to inform or notify the system when specific resource items are accessed. No conflicting obligations are expected to arise, when multiple notification-related obligations are defined on a resource. (ii) Retention- related, which specifies actions related to life-cycle of the resource item accessed. Conflicts are expected to arise due to specifying different requirements by different obligations on the retention of the resource item.

i. Policy Refinement Process

A goal graph structure of the high-level goals that the system can handle is elaborated in the process of this component. The process of this component depends on the application and is carried out by the administrator or developer, and it is carried out during the design and implementation of the system. This may reduce the potential errors, but during execution of policy this activity is intended to check each type of conflict and use the priority of rules execution, elimination rules or changing them according to the request is a part of this activity role. This process aims to score high degree in scalability by avoiding conflicts between different applications.

Conflicts arise, when more than one obligation which cannot be executed together, enforced on a resource item. Resolution of conflicts at static time aims to identify which obligations must apply. As it is clear in Figure 2 , the objective is to eliminate the many-to-one map from obligations to resource items, to a one-to-one map with respect to the constraints on the resource item, where each obligation applies to at most one data item, and not inconsistent with the constraints. One way to select the most relevant obligation is to assign priority degree to each obligation, and select the one with the highest or lowest degree; these degrees can be assigned by the administrator.

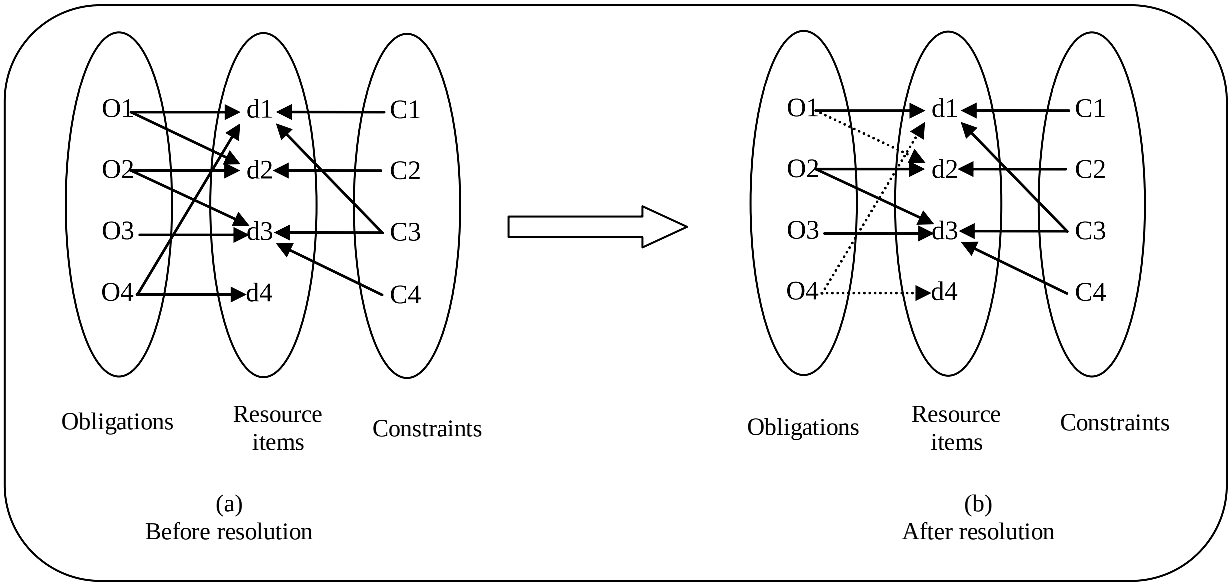

Figure 2. Resolution of conflicts including the constraints

The algorithm of this process may now be described as follows. Given a set of rules (the policy) and a set of resources, the algorithm first identifies, for each resource, based on the rules in the policy the set of obligations that apply to that resource. Subsequently, for each resource and associated set of obligations, the algorithm identifies if any of the obligations potentially conflict with each other; then these obligations are identified. Two or more obligations conflict with each other if one or more of them has a temporal components regarding when the task is to be carried out. If none of the obligations associated with a resource have a temporal component, then no conflict is possible, in this situation, the set of attributes are used to capture these requirements. Using the object notation, the individual components of task, predicate and attributes of an obligation O may be referred to by O.tasks, O.predicates and O.Attributes respectively, where Attributes list the set of obligation attributes, the task defines the action to be taken, and the predicate associates the task with a resource item.

If there is a detected conflict, the algorithm provides a way to solve conflicts at two levels. By identifying the conflicting obligations and constraints then redefining those rules which are causes conflict. A feedback to policy authors is required at some situations, due to the changes at policy level. And by determining which rule of the conflicting obligations to apply to a constrained resource item, when more than one obligation (which cannot be executed together) is defined on the item. By using policy degree (the priority value with respect to other policies).

The algorithm is formally defined as follows.

StaticDetectAndResolve()

1. Let r be a rule in the policy, $o.obligation$ refers to the set of obligations mandated by r. Let *n be the number of resources in the system. Let*$O_i (1 \le i \le n)$ represents a set of obligations that apply on that resource for r, initially $\forall i,\ O_i = \emptyset$.

2. For each resource d_i which the rule r applies, if $r.obligation \neq \emptyset$, then $O_i = O_i \cup r.obligation$ we have the mapping M, where $M(d_i) \to \{O_i\}\ , \forall i$.

3. Let O_{Ci} be the set of rules that conflict with each other$(1 \le i \le n)$. O_{Ci} can be detected as follow:
   a. $O_{Ci} = \emptyset, \forall i.$
   b. Let O_{Ni} the notification related obligation which does not contain conflicting rules, and O_{ri} the retention-related obligation. $\forall i, \{O_i\} = \{O_{Ci}\} \cup \{O_{Ri}\}$.
   c. $\{O_{Ci}\} = \emptyset$ if and only if $|O_{Ri}| \le 1$,
   d. $if\ |O_{Ri}| > 1$,then for each $O_l, O_k \in O_{Ri}$ where $O_l \neq O_k$. There is no conflict between O_l andO_k if and only if $O_l.Attribute = O_k.Attribute = \emptyset$. Because two or more obligations conflict with each other if one or more of them has attributes that define temporal components.
   e. If conflict is detected goto 5

4. If there are any listed constraints C_{id} on resource d $(C_d \equiv a\ combination\ of\ all\ constraints\ on\ resource\ d)$
   i. Let $C_{1d}, C_{2d}, C_{d3} \dots, C_{md} \in \{C_d\}$, $(\ m \equiv number\ of\ constraints\ on\ d)$
   ii. Let $O_i = O_i \cup C_d$
   iii. Goto step 3

5. $\forall\ d_i$, determine O_{ci} and O_i which are trying to access constrained resource item.
   i. If $O_{ci} = \emptyset$ then stop else proceed with step (ii).
   ii. Order the obligations using the policy degree.
   iii. Select the obligation with highest degree to enforce from O_{ci}.
   iv. Select those obligations which are trying to access a constrained resource for redefine.

6. The mapping $M(d_i) \to \{O_i\}\ , \forall i$ defines each resource d is mapped to a set of rules O that potentially causes the conflict. One-to-one mapping for M is a sufficient condition for a conflict-free system. In this situation, each resource item maps to at most one obligation, and no conflict is possible.

7. For each resource item, we have a list of obligations that could potentially cause conflicts when modified. Also the mapping $M(d_i) \to \{C_d\}\ , \forall i$ defines each resource d is mapped to a set of constraints C_{id} that potentially restricts some obligations.

End of algorithm

ii. Dynamic Conflict Resolver Process

Other actions like policy composer and evaluation must occur before this process a combination and evaluation of policies, we considered these activates as a part of this main process. Since each policy gives a single decision, the policy combination algorithms (PCAs) combine these decisions into a single policy decision. PCAs use to resolve conflicts during runtime. These algorithms take the authorization decision from each policy as input and apply some standard logic to come up with a single decision. There is a need to include algorithms such as these as PCAs in authorization languages to provide more functionality and flexibility in defining policies. Resolution rules which are the rules that specify the post-condition that the system can reach from a given condition used to resolve conflicts, since they determine the rule to be executed from a set of conflicting rules. A resolution policy is a set of resolution rules can be specified using if-then statement.

if {condition} then {Action}

This means that: “if the system state satisfied by the condition, then the system is preferred to reach the state represented by post-condition”. The resolution technique prioritizes one rule over another by stating that if two conflicting post-conditions can occur, then one of the post conditions is preferred over the other.

Example

if {there is no users inside the range of Sensor i} then {turn off Sensor i}

In this situation the system checks all working sensors until finds the required sensor then executes the related action. However, if the execution of the above example occurred while the neighbor sensor was failed at the same time, the system may stop the execution of this policy. Because this may leads to some network problems such as coverage, interference or load balancing.

The algorithm of this component requires monitoring user requests as they enter the system, to ensure if they necessitate any obligations or not, then checks and resolve conflicts as they arise. While a request to access resource is entered the system, if the request is allowed, and authorizes an obligation in the resource, then add this request to the set of live obligations (L_d) for that resource. When the set of live obligation is not empty a set of steps take place to check for potential runtime conflicts between obligations in the list. When more than one obligation is mandated (which cannot be executed together), conflicts arise and detected, a set of steps must be called. Conflict resolution is the set of steps to determine which one of the conflicting obligations to execute on a resource item. After successful execution of requests on the resource, must be removed from live list. Conflict detection during runtime performed when a new obligation is added to the live obligation list. Constraints C_d of resource d must be checked, combined C_d and detect for conflicts with resource requests to avoid potential errors.

In the case of dynamic resolution, the system does not have to plan for all potential obligations, only the obligations that actually arise have to be checked for conflicts. When none of the access rules or constraints is violated, for every additional request in the live list, a different view of the resource V_d associated with a set of obligations that apply to the request, and used to serve that request. In this sense, resolution of conflicts at runtime provides the additional flexibility that the many-to-one mapping of obligations to data items need not be converted to a one-to-one mapping with respect to the resource constraints. It may be sufficient to create a view of a one-to-one mapping at runtime, provided no rules are violated.

Any obligation O_i could override another obligation O_j if and only if O_i has a higher priority degree than O_j and both cannot be executed together. For instance, if O_i specifies that a resource item is to be removed immediately on access, and if O_j specifies that the resource item is to be turn on at 8 am “while it was off”, then we may decide, based on the context that O_i overrides O_j. During the dynamic resolution of conflicts, user requests must be tracked as they arise and obligations, if any, are associated with the respective resource items. When a new obligation is defined on a resource item which already has one or more live obligations associated with it, the new obligation may/may not override the current set of live obligations.

DynamicDetectAndResolve()

```
1. Initially $L_d = \emptyset$, $V_d = \emptyset$
2. If nof $(L_d)$ = nof$(L_d)$ + 1, (the number of requests in $L_d$increased by one new request) then
 a. If the new request is allowed and mandates on obligation $O$ on resource $d$ then
    Let $L_d = L_d \cup O$
```

```
b. For $\forall i, (1 \le i \le n)$, $O_{Ci}$ can be detected as follow:
   i. $O_{Ci} = \emptyset, \forall i$.
   ii. $\forall i, \{O_i\} = \{O_{Ci}\} \cup \{O_{Ri}\}$.
   iii. $\{O_{Ci}\} = \emptyset$ if and only if $|O_{Ri}| \le 1$,
   iv. $if\ |O_{Ri}| > 1$, then for each $O_l, O_k \in O_{Ri}$
       where $O_l \neq O_k$, there is no conflict
       between $O_l$ and $O_k$ if and only if
       $O_l.Attribute = O_k.Attribute = \emptyset$,
       else goto step 3

c. If there are any constraints $C_{id}$
   $(C_d \equiv a\ combination\ of\ all\ constraints\ on\ resource\ d)$
   i. Let $C_{1d}, C_{2d}, C_{d3} \dots, C_{md} \in \{C_d\}$,
      $(m \equiv number\ of\ constraints\ on\ d)$
   ii. Let $L_d = L_d \cup C_d$
   iii. goto step 2(b)

3. if a request on d is allowed then 3(i) else (5)
   i. if $L_d \neq \emptyset$ then $L_d = L_d \cup O_i$
      $(O_i \equiv the\ existing\ obligation\ in\ L_d)$.
   ii. if $O_j$ overrides $O_i$ then
          $L_d = L_d - O_i$
      $(O_j \equiv the\ new\ obligation)$.
   iii. If $O_i$ overrides $O_j$ then $L_d = L_d - O_j$

4. If $C_d \neq \emptyset$ and $L_d \neq \emptyset$ then
   i. Update the live list $L_d = L_d \cup C_d$
   ii. If $C_i$ overrides either $O_i$ or $O_j$ then
       $L_d = L_d - \{O_i, O_j\}$.
   iii. If either $O_i$ or $O_j$ override $C_i$ then
       $L_d = L_d - C_i$.
   iv. If 3(ii) , 3(iii), 4(ii) and 4(iii) are false then
       create a view for the resource $v$, and let
       $V_d = V_d \cup v$

5. Any set of obligations $O_{Cd}$ that conflict with
   each other, or conflict with a combination of
   constraints $C_d$ for a resource d, are identified as
   dynamic conflicts for that resource.
6. Remove obligations from the live list of
   resource d.
7. Modify the constraints list if updated.

End of algorithm
```

4. CONCLUSION AND FUTURE WORK

This paper, proposes PobMC framework which is based on Event-Condition-Action (ECA) rules for policy-based management of autonomous computing system. Furthermore, the paper covers the issues relating to detecting and resolving conflicts which can arise in the obligation policies and some ideas on how to refine high level goals.

StaticDetectAndResolve() and DynamicDetectAndResolve() algorithms have been defined for detect and resolve conflicting policies during compile time and run time. It is observed that if the conflicts are detected statically or dynamically, it is important to modify the effect of some of the rules through the conflict resolution mechanism.

Conflict detection and resolution has also been studied in the case of business policies for authorizing various activities. We believe our work is one of the important to discuss the role of obligation management and resource constraints in autonomous systems and to present algorithms for conflict management of obligations related to management of resource items(data and device), for self-adaptive systems.

The comparison of the most prominent approaches that have been presented in this paper, highlights some important issues have been that have been discussed for large-scale adoption of policy-based systems. Moreover, we show that existing policy-based systems do not reason about concurrent rule enforcements and define no enforcement ordering. Furthermore, do not verify action execution and assume that rule enforcement was successful. In addition to all these drawbacks most of previous works do not thoroughly investigate the effects of different policy.

However, this work does not verify the proposed algorithms. In the future, we plan to develop a static and dynamic analysis for administrators to specify policies easily.

ACKNOWLEDGEMENT

The authors would like to express their deepest gratitude to UniversitiTeknologi Malaysia (UTM) for their financial support under Research University Grant Scheme (Vot number Q.J130000.7128.02H13).

REFERENCES

[1] A. A. Mansor and W. M. N. Wan-Kadir, *A Comparative Evaluation of State-of-the-Art Approaches in the Design of an Adaptive Software System*. Kuala Lumpur, Malaysia, ASME Press, 2011.

[2] O. O. A. Oyebisi T.O, "Developmentof Congestion Control Schemefor Wireless Mobile Network " *Journal of Theoretical and Applied Information Technology* vol. Vol4No10, p. 8, 2008.pp.965-972

[3] E. C. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," *Software Engineering, IEEE Transactions on*, vol. 25, pp. 852-869, 1999.

[4] J. Young and E. Mendizabal, "Helping researchers become policy entrepreneurs," 2009.

[5] G. Tonti, *et al.*, "Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder," *The SemanticWeb-ISWC 2003*, pp. 419-437, 2003.

[6] J. Strassner and J. S. Strassner, *Policy-based network management: solutions for the next generation*: Morgan Kaufmann, 2004.

[7] M. Sloman and K. Twidle, "Domains: A framework for structuring management policy," *Network and Distributed Systems Management*, pp. 433–453, 1994.

[8] M. Sloman, "Policy driven management for distributed systems," *Journal of Network and Systems Management*, vol. 2, pp. 333-360, 1994.

[9] N. C. Damianou, "A policy framework for management of distributed systems," Citeseer, 2002.

[10] E. H. Sibley, *et al.*, "The role of policy in requirements definition," 1993, pp. 277-280.

[11] J. B. Michael, *et al.*, "Integration of formal and heuristic reasoning as a basis for testing and debugging computer security policy," 1993, pp. 69-75.

[12] N. Dunlop, *et al.*, "Dynamic conflict detection in policy-based management systems," 2002, pp. 15-26.

[13] C. Wang, *et al.*, "A policy-based approach for QoS specification and enforcement in distributed service-oriented architecture," in *2005 IEEE International Conference on Services Computing, SCC 2005, July 11, 2005 - July 15, 2005*, Orlando, FL, United states, 2005, pp. 307-310.

[14] C. S. Shankar, "Policy-based pervasive systems management using specification-enhanced rules," vol. 67, ed, 2006.

[15] S. Davy, *et al.*, "Efficient Policy Conflict Analysis for Autonomic Network Management," 2008, pp. 16-24.

[16] C. S. Shankar, *et al.*, "An ECA-P policy-based framework for managing ubiquitous computing environments," 2005, pp. 33-42.

[17] S. Davy, *et al.*, "On harnessing information models and ontologies for policy conflict analysis," 2009, pp. 821-826.

[18] Z. Wu, *et al.*, "Dynamic policy conflict analysis in operational intensive trust services for cross-domain federations," 2009, pp. 1-6.

[19] A. Mohan and D. M. Blough, "An attribute-based authorization policy framework with dynamic conflict resolution," 2010, pp. 37-50.

[20] N. Khakpour, *et al.*, "Formal analysis of policy-based self-adaptive systems," in *25th Annual ACM Symposium on Applied Computing, SAC 2010, March 22, 2010 - March 26, 2010*, Sierre, Switzerland, 2010, pp. 2536-2543.

[21] J. Barron, *et al.*, "Conflict analysis during authoring of management policies for federations," Dublin, Ireland, 2011, pp. 1180-1187.

[22] Z. Wu and Y. Liu, "Automatic policy conflict analysis for cross-domain collaborations using semantic temporal logic," pp. 1-8.

[23] C. Ma, *et al.*, "Conflict detection and resolution for authorization policies in workflow systems," *Journal of Zhejiang University-Science A*, vol. 10, pp. 1082-1092, 2009.

[24] M. Sirjani, *et al.*, "Modeling and verification of reactive systems using Rebeca," *Fundamenta Informaticae*, vol. 63, pp. 385-410, 2004.

[25] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems: Specification* vol. 1: springer, 1992.

[26] A. A. Mansor, *et al.*, " Policy-based Approach for Dynamic Architectural Adaptation: A Case Study on Location-Based System", in MySec011, Johor Bahru, Malaysia, IEEE 12-14 December 2011.