# STATIC ANALYSIS TO AVOID OVERLAP OF POLICIES IN POLICY-BASED MANAGEMENT SYSTEMS

Abdehamid Abdelhadi Mansor[1] and Wan Mohd Nasir Wan Kadir[2]

[1]Department of Computer Science, Faculty of Mathematical Sciences, University of Khartoum, Khartoum, Sudan
[2]Department of Software Engineering, Faculty of Computing , Universiti Teknologi Malaysia, Johor, Malaysia

***ABSTRACT***

*A management policy evolves over time by addition, deletion and modifications of rules. Policies authored by different administrators may be merged to form the final system management policy. Since policies are used to govern the system behavior, conflicts may arise in the set of policies and also may arise during the refinement process, between the high-level goals and the implementable policies. Moreover, policy conflict can result from propagation, action composition and other constraint policies, which cannot be detected by simply comparing authorization policies. Static and dynamic conflicts are considered as two classes of conflict which need to be understood and independently managed. Furthermore, the distinction between these two classed is important; as detecting and resolving of conflict can be computationally intensive, time consuming and hence, costly. However, a dynamic conflict is quite unpredictable, in that it may, or may not; proceed to a state of a realized conflict. In this paper we present static analyses to address the overlap cased when there are two or more policies are enforced simultaneously. Moreover, the paper provides temporal specification patterns to avoid each type of conflicts, and to ensure that policies are enforced correctly.*

***KEYWORDS***

*Policy-conflict, overlap, policy-based management, static analysis*

## 1. Introduction

Policy-based management is a well-established approach where policies are specified as Event-Condition-Action (ECA) rules which specifying the management actions to be performed when certain situations occur. However, the main challenge limits the development of policy-based approach is the policy conflicts. Conflicts may arise in the set of policies and also may arise during the refinement process, between the high-level goals and the implementable policies [1]. The system must be able to handle 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 a set of policies in a policy server as part of the policy specification process or at run-time [2].

DOI : 10.5121/ijsea.2014.5610 

In policy-based management system, policies are specified by the system manager to govern system behavior. Such governing policies evolve over time by policy composition, rule modifications and due to system dynamisms [2]. Multiple policies can become simultaneously eligible for enforcement in a situation and the order of enforcement may determine the final system state. However, in such cases, changes in policies at run time may result in system instability, overlap and cycles among rules which are lead to policy conflicts [3]. The resulting conflicts can be detected and avoided during system design. In this paper, the relationship between policies is a crucial to our discussion of policy conflicts as it is our contention.

Overlap may arise in many situations related to sharing of resources for which both domains have applied to policies that have management responsibilities on the same set of object [4]. Overlapping domains reflect the fact that multiple managers can be responsible for that multiple policies apply to the object or an object. Obviously this can lead to conflicts between policies or managers. Existing approaches to conflict detection are limited in scope and can only detect conflicting actions if they are explicitly stated. In addition, current techniques do not detect overlaps in management policies. This paper presents static analyses to specify and detect potential overlap in order to be avoided earlier since the design time, where most of the required specification can be detected. Moreover, the paper provides temporal specification patterns to avoid the potential overlaps, and to ensure that policies are enforced correctly.

In sections 2 and 3 of this paper, briefly presents an introduction to policy conflict and classification of conflicts. Introduction of PobMC and the smart mall system introduced in section 4. Sections 5 presents system architecture management. In section 6 overlap Analysis is presented and discussed. Sections 7 provides and discusses conflict analysis. Results and discussion presented in section 8. Related work is presented in section 9. Finally, conclusions and further work are discussed in section 10.

## 2. Policy Conflict

A policy [5] is represented as an information which can be used to modify the behaviour of a system when a managed object moved 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 means of grouping objects and are similar to file system directories [6]. 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.

A policy is a finite collection of rules, consisting of event, condition and action. PobMC uses policies that are formulated as follow:

on <event> if <condition> then <action>

which can be read as: while an event happens, the action is executed when the condition is true. A typical example may look like, “if the active area is unoccupied, then turn the sensor off”. When the last mobile node leaves the active area, an event is generated in the system. The system receives the event and checks the sensor; if it is running then the system will turn it off.

Policy conflicts can arise when multiple policies control a system behaviour. A conflict occurs when an event triggers multiple actions that cannot occur together as specified by the system administrator. Human error is one obstacle to accurate access-control policies; the policy authors who assign and maintain these policies are prone to making specification errors that lead to incorrect policies. Access-control policies consist of a set of rules that dictate the conditions under which users will be allowed access to resources. These rules may conflict with each other. For example an obligation policy may define an action that must be performed but there is no authorization policy to permit the manager to perform this action. Conflict detection between management policies can be performed statically for a set of policies in a policy server as part of the policy specification process or at run-time [7-8].

Policy conflicts have been categorized into four broad categories [9]. Each category may present itself either statically or dynamically. First, internal policy conflict, which occurs when there is incompatibility between policies which are assigned to single roles, second, external policy conflict, occurs when combining roles which are in isolation of each other present no conflict, but contain policies which in co-existence are in conflict. Third, policy space conflict, occurs when more than one policy space manage the same set of subjects and attempt to enforce various and conflicting policies over them, and fourth role conflict, expected when a user obtains a set of incompatible role assignments.

Static and dynamic conflicts are considered as two classes of conflict which need to be understood and independently managed [9]. Furthermore, 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, a dynamic conflict is quite unpredictable, in that it may, or may not; proceed to a state of a realized conflict. This class of conflict must be detected at run-time.

## 3. Policy Conflict Classification

For any set of policies $\{p_i, q_j, O_k\}$ has been enforced in the system, the term policy conflict can be defined as follow:

Two policies $p_i$ and $q_j$ are in conflict if and only if one of the following cases takes place:
$p_i$ and $q_j$ have been enforced simultaneously, then the system cannot choose a policy to enforce.
The execution of $p_i$ violates the action of $q_j$.

Executing $p_i$ that makes $q_j$ impossible to be enforced and vice versa (eg. turn-on and turn-off for the same device simultaneously).

Executing of $p_i$ before $q_j$ while it must be executed after $q_j$ (the ordering). For instance, $q_j$ is "authorize the user" and $p_i$ is "download the system files". While in the system specification, the system must authorize the user before he gets the system files.

In order to detect the conflicting policies, first we must identify and define conflicting actions explicitly. Then, the simultaneous triggering of those policies should be investigated. Second, the ordering of events and actions should be identified clearly. Third, the inconsistent policies should be identified to prevent them from simultaneous execution. Finally, all system policies should be checked to identify policies which make the action of other policies by violating their conditions.

For instance, in our SMALLS example if $O_k$ is the policy that "identifying the mobile phone location", while the mobile phone is currently attached to the corresponding APs, no policy that disable the database server must be applied before the policy that "send the required information to the mobile phone".

According to the definition of conflicts and the above mentioned cases to avoid the potential conflicts we need some information to define each type of conflicts. We should introduce classification of various conflicts that expected among system policies.

A. *Modality Conflicts,* which expected when there is a triple overlap between the set of subjects, targets and actions, of two or more policies with modality of opposite sign to the same subjects, actions and targets. For instance, "the subjects are authorized and forbidden to perform the action on the target objects". Another example, "the subjects are forbidden but required to perform the actions on the object". As an example in the SMALLS example, "The manager is allowed to mount a device's file system onto the active area file system, while the device is not authorized".
B. *Inconsistencies,* which occur due to omissions, errors or conflicting requirements of the manager specifying the policies. For instance, "an obligation policy defines an activity that must be performed, but there is no authorization policy to perform the activity". As an example in the SMALLS example, "the same manager cannot authorize the user and turn on the sensor".
C. *Multiple Managers' Conflicts,* overlapping of domains related to sharing of resources such as a gateway between two networks, a service between two or more applications, etc. Overlapping leads to conflicts between policies when managers can be responsible for an object or that multiple policies apply to the object. In some situations overlapping is prevented by creating a new domain with an independent manager and all objects from the overlapping set are moved into this new domain. E.g. "at a specific time a manager is allowed to turn off all active area sensors, while another manager is demounting the file system when the device is leaving the active area".

This paper presents static analyses to address the; multiple managers' conflicts, when there are two or more policies are enforced simultaneously. Moreover, the paper provides temporal specification patterns to avoid each type of conflicts, and to ensure that policies are enforced correctly.

# 4. The Application of PobMC

## 4.1 Introduction to PobMC

In PobMC [10], policies are used to control the system behavior. Policies provide a high-level of abstraction and allow us to decouple the adaptation concerns from the application code. Thus, we can change the system behaviour as well as adaptation schemas by changing policies. PobMC is composed of a set of modules; called Self-Managed Module (SMM) is the policy-based building block of PobMC. In our case study, we consider three SMMs including SenModule, LocModule and SecModule to manage sensors, locations and security constraints respectively. An SMM is a set of actors which can manage their behavior autonomously according to predefined policies. PobMC supports interactions of an SMM with other SMMs using well-defined interfaces in the model.

Each module in PobMC consists of managers and actors. Actors, which are manage their behaviour autonomously according to predefined policies. Managers, which are manage and provide autonomic behaviour to corresponding actors. Interaction among managers is supported in PobMC.

### 4.2 Illustrative Case Study

This section, briefly presents an example that we use throughout the paper to illustrate the supported concepts of adaptation and the underlying modelling techniques.

The Smart Mall System (SMALLS) is a system that allows users to navigate their location in the mall. The users could be able to query the place that they are heading such as baby area, shoes area, food area, banking services area etc. The system directs user how to find the area. SMALLS operation can be summarized as follows. Each user carries a mobile device such as a smart phone as well as a wireless sensor. In addition, locations in the environment shopping area or services area are associated with their own wireless sensors. The sensors determine which area is closest to the user at a given moment and pass this information to a server, which provides specific Web services for each individual object.

SMALLS is required to adapt its behaviour according to the changes of the environment. To achieve this aim, we suppose that the system runs in normal, vacation and failure modes and in each context it enforces various sets of policies to adapt to the current conditions. For the reason of area, here we only identify policies defined for sensing control module while the system runs in normal or failure modes.

## 5. System Architecture Management

The main concepts of our system architecture management are grouping objects “domain”, support the specification “a policy service” and reflect the organizational structure “storage of policies and roles”, responsibilities and relationships between the system components “managers”, “coordinator”, and “managed elements”.

Domains are used to group system objects according to object type, responsibility and authority. A sub-domain is a member of another domain “parent domain”. However sub-domain is not subset of the parent domain, an object or sub-domain may be member of multiple parent domains, figure 1 illustrates the relationship between SMALLS domains. In Figure 1, all the objects in sub-domains SenModule, LocModule and SecModule are members of parent domain SMALLS_Management which therefore overlap.

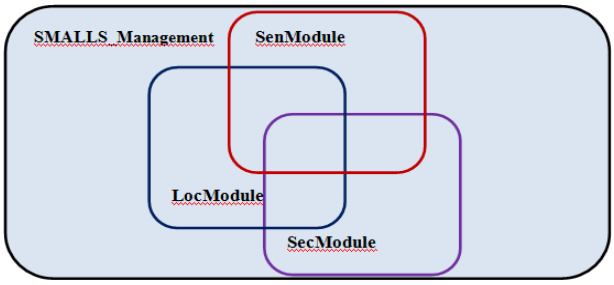


Figure 1. SMALLS Domains

Domains can be identified by path name such as:

/SMALLS_Management/SenModule , // which identify the domain SenModule policy applying to domain SenModule will also apply to members of domain SMALLS_Management. Some policies which applying to domain SenModule can be applied to domains SecModule and LocModule. However, all system policies are applying to domain SMALLS_Management.

Using union $\vee$, intersection $\wedge$ and difference $-$ operators, domains can be combined to form a new set of objects for applying a policy. The advantage of combining domains is that deletion and addition of objects from/to the domains can be done without changing the policies.

Two domains overlap if there are objects which are members of both domains, for example SenModule & LocModule in figure 1. Overlap arises in many situations related to sharing of resources for which both domains have applied to policies that have management responsibilities on the same set of object. Overlapping domains reflect the fact that multiple managers can be responsible for that multiple policies apply to the object or an object. Obviously this can lead to conflicts between policies or managers.

A policy, whether it is concerned with obligations or with authority, has the following attributes [11]:

(i) **Modality**; each policy has positive or negative modality (see figure 2), which are important and adequate for the analysis in this work,
(ii) **Policy subjects;** which define a set of users to whom the policy is directed,
(iii) **Policy target objects**; define the set of objects at which the policy is directed,
(iv) **Policy goals**; can be expressed as high-level goals which specify what the manager should achieve in abstract terms which do not identify how to achieve the goals, and

**Policy constraints** can be expressed in terms of system properties, such as extent or duration, or some other condition.

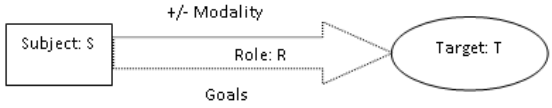


Figure 2. Policy Configuration

For any set of policies $\{p_i, q_j, O_k\}$ has been enforced in the system, the term policy conflict can be defined as follow:

Tow policies $p_i$ and $q_j$ are in conflict if and only if one of the following cases takes place:

(i) $p_i$ and $q_j$ have been enforced simultaneously, then the system cannot choose a policy to enforce.
(ii) The execution of $p_i$ violates the action of $q_j$.
(iii) Executing $p_i$ makes $q_j$ impossible to be enforced and vice versa (eg. turn-on and turn-off for the same device simultaneously).
(iv) Executing of $p_i$ before $q_j$ while it must be executed after $q_j$ (the ordering). For instance, $q_j$ is "authorize the user" and $p_i$ is "download the system files". While in the system specification, the system must authorize the user before he gets the system files.

In order to detect the conflicting policies first we must identify and define conflicting actions explicitly. Then, the simultaneous triggering of those policies should be investigated. Second, the

ordering of events and actions should be identified clearly. Third, the inconsistent policies should be identified to prevent them from simultaneous execution. Finally, all system policies should be checked to identify whether policies make the action of another policies by violating their conditions. For instance, in our SMALLS example if $O_k$ is the policy that "identify the mobile phone location", while the mobile phone is currently attached to the corresponding APs, no policy that disable the database server must be applied before the policy that "send the required information to the mobile phone".

## 6. Overlap Analysis

In this work, static analysis is used to determine whether an event specified in the policy condition matches received event. A trigger graph is created after the policy compilation to identify the overlap between set of subjects, targets and actions (see figure 3), in simultaneously triggered policies. Furthermore, specifying the overlap will eventually avoid modality conflicts and multimanager conflicts, thereby improves system scalability. Static analysis is capable to evaluate only potential conflicts rather than actual conflicts. However, static analysis is limited to evaluate policy constraints, because of that constrains are completely depending on run-time state; moreover domain membership may change at run-time.

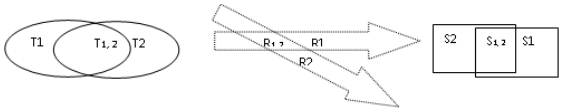


Figure 3. Overlapping Target (T) Role (R) and Subject (S)

### 6.1 Overlapping of Subjects

This occurs when the subject of two or more obligations or authority policies overlap, this means that it is expected in some cases the same subject may manage different group of targets. Figure 4 shows that P1 applies $\{s_1, t_1, r_1\}$ and p2 applies $\{s_2, t_2, r_2\}$, while there are some subjects $\{s'\}$ are included in both P1 and P2, this means that, the subject of p1 is $(s' \cup s_1)$ and the subject of P2 is $(s' \cup s_2)$. Both p1 and p2 applies $\{s', t_1, r_1\}$ and $\{s', t_2, r_2\}$ respectively.

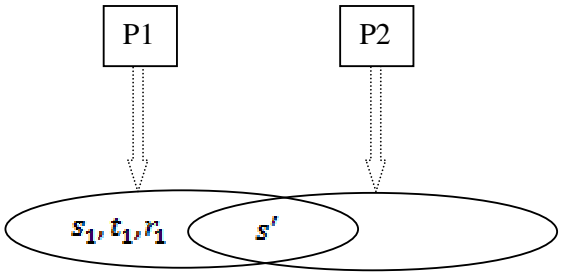


Figure 4. Overlap of Subjects

**Example1** in our SMALLS scenario the same manager may enforce two different policies, the first policy to govern a group of Wi-Fi access points, while the other policy is to govern a group of users in the SMALLS active area as follows.

P1: “turn off all the sensors in the supermarket shopping area from 12:00 pm to 7:59 am”
P2: “*Users with the description name Security are allowed to perform any action on any resource at anytime from anywhere in the mall”*
*Overlap of Roles*

This occurs when the roles of two or more obligations “O” or authority “A” policies overlap, this means that it is expected in some cases the same object may be directed by different actions. The roles of such policies are in conflict if-and-only-if for any two policies p1 and p2 in one of these forms {O-/O+, A-/A+, O+/A-}, such that (+) indicates that the policy is permitted and (-) indicates that the policy is forbidden. Figure 5 shows that P1 applies $\{s_1, t_1, r_1\}$ and p2 applies $\{s_2, t_2, r_2\}$, while there are some roles $\{r'\}$ are included in both P1 and P2, this means that, the role of p1 is $(r' \cup r_1)$ and the role of P2 is $(r' \cup r_2)$. Both p1 and p2 applies $\{s_1, t_1, r'\}$ and $\{s_2, t_2, r'\}$ respectively.

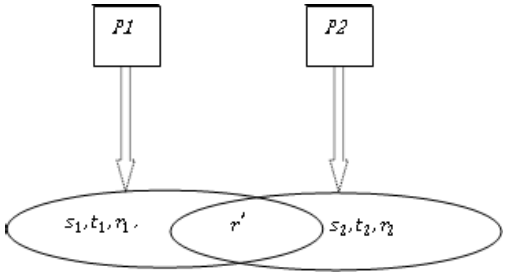


Figure 5. Overlap of Roles

## 6.2 Overlap of Targets

Similarly, when the targets of two or more obligations “O” or authority “A” policies overlap, means that it is expected in some cases the same target may be managed by different set of policies. The targets of such policies are in conflict when there are some constraints on the target. Figure 6.6 shows that P1 applies $\{s_1, t_1, r_1\}$ and p2 applies $\{s_2, t_2, r_2\}$, while there are some targets $\{t'\}$ are included in both P1 and P2, this means that, the role of p1 is $(t' \cup t_1)$ and the role of P2 is $(t' \cup t_2)$. Both p1 and p2 applies $\{s_1, t', r_1\}$ and $\{s_2, t', r_2\}$ respectively.

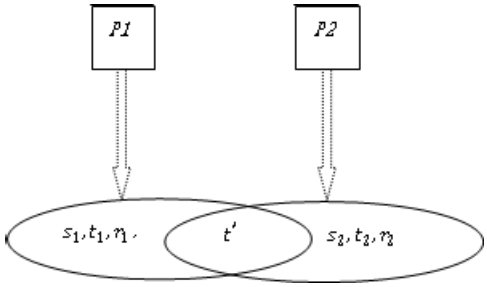


Figure 6. Overlap of Targets

## 7. Conflict Analysis

This work is concentrating on static conflict analyses which assist to specify policies, roles and relationships. In the following sections, we show how to avoid the modality conflicts, inconsistency and multi manager conflicts.
We discuss overlap as the most important factor to policy conflicts. There are several possibilities for overlapping between policies (see figure 7), triple overlap "the set of subjects, targets and actions, of two or more policies with modality of opposite sign to the same subjects, actions and targets overlap"; double overlap "both the subjects and the target of the policies overlap"; subjects-targets overlap "the subjects of one policy and the target of another policy overlap", target overlap and subjects overlap.

Modality conflict is expected when there is a triple overlap; here we give some example using our case study to show that modality conflict can arise due to different modalities in the set of policies.

In SMALLS system there are different managers (managers and coordinators), managers' tasks are coordinated by a coordinator. Moreover, each manager has some responsibilities such as manages and governs system services (positioning service), resources (Wi-Fi access points, computer servers and smart phones). Both managers and coordinators use policies to manage and control the system. However some policies enforced by coordinators may restrict the managers from performing their tasks, and managers' policies may restrict each other. When two or more policies applying to a tuple, there is a potential conflict and the policies can be checked to see whether there is an actual conflict "positive and negative policy with the same subjects, targets and actions".

Let $S = \{s_0, s_1, s_2, \ldots, s_n\}$ be a list of subjects, $T = \{t_0, t_1, t_2, \ldots, t_m\}$ be a list of targets and $R = \{r_0, r_1, r_2, \ldots, r_k\}$ be a list of roles in the system. For any two policies P1 and P2, let P1 being negative and P2 is positive. Figure 7 below show the overlapping between p1($s_1, t_1, r_1$) and p2($s_2, t_2, r_2$) for common subjects, actions and roles.

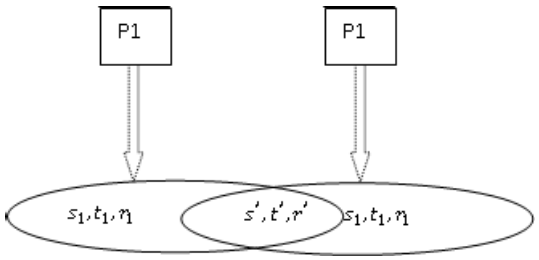


Figure 7. Triple Overlap Between Two Policies

Modality conflicts are expected if and only if a triple overlap between the policies -P1 and +P2 occurs and create the following tuples to which different sets of policies apply:

P1 applies $(s_1, t_1, r_1) + (s', t', r')$, while P2 applies $(s_2, t_2, r_2) + (s', t', r')$, what mean that both P1 and P2 applies $(s', t', r')$.

Example2, in SMALLS system there will be some policies pertaining to all staff and customers as well as some more specific policies related to part of the staff. Let us assume P1 and P2 be three polices assigned by different managers.

P1: "Users are not authorized to send requests or perform any action on any resource from 12:00 pm to 7:59 am"
P2: "Users with the description name ADMIN are allowed to perform any action on any resource at any time"

Modality conflicts can be avoided either by changing one policy or block system managers from the managed objects [11]. However, changing policies is not desired in the system, due to the fact that rewriting a policy is time consuming and may not be convenient or desirable in the general case. Blocking system managers means preventing them from controlling objects for a while, however this way is not desirable because the system is completely governed by policies enforced by managers. Thus, the best way to solve the conflicts between P1 and P2 in example 2, is allowing both of them as they enforced and determine which policy should be enforced first. Therefore we can identify the priority for each single policy of each conflicting pair. The assigned priority helps to determine which policy should be ignored when at least one of the policies actions is constrained.

**Definition** 1, Definition 6.1 let $\rho_i = \{pr_i, e_i, c_i, a_i\}$ and $\rho_j = \{pr_j, e_j, c_j, a_j\}$ are two policies, where $pr$ is the priority, $e$ is the event, $c$ is the condition and $a$ is the action. The action $x.m$ means the message $m$ is sent to object $x$, where $a_i = x_i.m_i$ and $a_j = x_j.m_j$. We assume $\rho_i$ and $\rho_j$ which have simple actions are enforced by manager $Mgr_k$, and $Trig_{\rho i}$ denotes the triggering of policy $\rho_i$.

$$Trig_{\rho_i} \Leftrightarrow (e_i \wedge c_i) \wedge O(\neg e_i) \quad \text{..... (1)}$$

Formula (6.1) illustrates that policy $\rho_i$ is triggered if and only if its event and conditions are true, $O(\neg e_i)$ is to reset event after enforcing the policy.

To identify where a conflicts occurs and where potential conflict should be resolved in a set of policies, we should enumerate all subjects, targets which have a different set of policies applying to them. Moreover, overlapping area should be determined explicitly. Therefore, the simultaneous triggering of $\rho_i$ and $\rho_j$ policies should be investigated. The LTL formula 2 requires policies $\rho_i$ and $\rho_j$ not to be triggered simultaneously by enforcing the higher priority policy first.

$$\square[(pr_i > pr_j)?\ \rho_i : \rho_j] \quad \text{..... (2)}$$

The overlap detection algorithm in figure 8, marks the triggered events of all managers to prevent calling the conflicting rules twice.

## 8. Results and Discussion

The algorithm in figure 8, was executed using ponder language under the Linux redhat operating system, for three sets containing 100 " SecModule", 150 " LocModule", 50 " SenModule" rules.

The output reports 91.8% of the conflicts between managers. The execution was repeated for different number of policies.

Each of the evaluation was measured four times, assuming the number of policies in the location is the same throughout the execution. In figure 9, we can see that the average time required for the 4 times executions according to the execution stages are as follow:

- generate the object file 0.7888s,
- send a query to managers 0.5278s,
- retrieve context information 0.5677s, and
- send back result to the mobile 0.575s.

The amount of time required to perform static conflict avoidance at compile time is 2.46s.

**Algorithm Modality_Conflict (q ,P[],Ev[], Dom[])**

```
1: Let Pr: array[1…MaxDomSize] of
   priority;
2: Let overlap:=false; mc:=0;
3: while q is not empty
4: trigger(Ev[i], Ev[j], Ev[k] ); //push event in the queue q
5: Let Ev[i]:=i;
6: Let Ev[j]:=j;
7: Let Ev[k]:=k;  //mark triggered events
8: for all m in the domains do
9: Trigger(p[i], p[i+1]);
10:if pr[i+1]>pr[i] then
11:Let overlap:=true;
12:Let mc:=mc+1;    //increment of conflicts mc
13:end if;
14:end for;
15:Let Ev[i]:=i+1; Ev[j]:=j+1; Ev[j]:=j+1;
16:end while;
17:return(mc);
```

Figure 8. Overlap Ddetection Algorithm

The evaluation result shows that the performance of PobMC is better than the previous works. Less than a second was the enough to perform every task as individual. Furthermore, by this evaluation, it is possible to compare PobMC to other existing approaches in term of its avoiding overlap and policy conflicts.

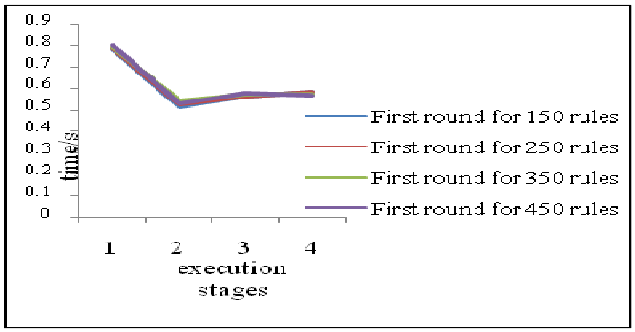


Figure 9. Performance Results for Overlap Algorithm

## 9. Related Work

There are some techniques to static conflict detection discussed in the literature. Shiva [12] proposed an extended model of Event-Condition-Action (ECA) called ECA-Post-condition to enable developers and administrators to annotate actions with their effects. The ECA-P framework 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. Also Khakpour et al. [13] presented an analysis using Rebeca [14] which is an actor-based language for modelling 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 this, a new classification of conflicts may occur during governing policies. Moreover, they introduced a number of correctness properties of the adaptation process in the context of their models. Then, they used static analysis of adaptation policies in addition to model checking technique to verify those properties. While their system includes many different managers, there may be more than event,

While a considerable attempt at static and dynamic conflict detection has been presented in previous work, the very complex and crucial issue of dynamic conflict detection in policy-based management has gone largely unresolved. Moreover, current research has revealed that there is still a large class of policy conflict which simply cannot be determined statically. The current state of the art in policy-based approach suffers from two main limitations. Firstly, they have limited ways of detecting and resolving conflicts in policies. Secondly, they do not have mechanisms to ensure that policies are enforced or executed correctly. These limitations severely limit the effectiveness of policies as a way of managing ubiquitous computing environments.

One approach to avoid conflicts in authorization rules is presented by Yu et al, in [15]. They argue that a large number of rules may apply to a service and detecting and resolving conflicts in real time can be a daunting task. However, their system is completely static and assumes that is it always possible to determine priorities ahead of time and avoid conflicts. Another approach for avoiding conflicts in policy specification is proposed by Agrawal, et al, for defining authorization policies for Hippocratic databases [16-18]. Their system allows system administrators to specify system policies for administration and regulatory compliance and these policies have the highest priority. Moreover, the system allows users to manage their privacy preference as their policies do not conflict with the system policies.

While a considerable attempt at static and dynamic conflict detection has been presented in previous work, the very complex and crucial issue of dynamic conflict detection in policy-based management has gone largely unresolved. Moreover, current research has revealed that there is still a large class of policy conflict which simply cannot be determined statically. The current state of the art in policy-based approach suffers from two main limitations. Firstly, they have limited ways of detecting and resolving conflicts in policies. Secondly, they do not have mechanisms to ensure that policies are enforced or executed correctly. These limitations severely limit the effectiveness of policies as a way of managing ubiquitous computing environments.

In our framework, the potential cycles specified and avoided earlier since the design time, here most of the requirement can be detected and catch during the analysis. The users policies may override other polices or be overridden based on context information.

## 10. Conclusion and Future Work

The analysis of the policy conflicts is implemented using policy specification language called PONDER to check and detect policy cycles. Same as other existing approaches described in sthis paper, PobMC is evaluated using SMALL case study, based on PobMC modeling and policy conflict results. Our experiments show that the PobMC framework leads to effective policy-based management and is a feasible approach. In addition, our evaluation with PobMC has the ability to enhance the existing approaches to support software adaptation. PobMC which enables the coordination among system managers in order to adapt to system changes and avoid the potential overlap is the main contribution of this paper.

Our future work includes the static analysis to avoid inconsistency when a set of rules is enforced by different managers which are managing the same system.

## References

[1] E. Lupu, et al., "Autonomous pervasive systems and the policy challenges of a small world!," in 8th IEEE International Workshop on Policies for Distributed Systems and Networks, POLICY 2007, June 13, 2007 - June 15, 2007, Bologna, Italy, 2007, pp. 3-7.
[2] E. Lupu and M. Sloman, "Conflict analysis for management policies," 1997, pp. 430-443.
[3] E. Lupu and M. Sloman, "A policy based role object model," 1997, p. 36.
[4] C. Shankar and R. Campbell, "A Policy-based Management Framework for Pervasive Systems using Axiomatized Rule-Actions," in Network Computing and Applications, Fourth IEEE International Symposium on, 2005, pp. 255-258.
[5] J. Strassner and J. S. Strassner, Policy-based network management: solutions for the next generation: Morgan Kaufmann, 2004.
[6] M. Sloman and K. Twidle, "Domains: A framework for structuring management policy," Network and Distributed Systems Management, pp. 433–453, 1994.
[7] E. H. Sibley, et al., "The role of policy in requirements definition," San Diego, CA, 1993, pp. 277-280.
[8] 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.
[9] N. Dunlop, et al., "Dynamic conflict detection in policy-based management systems," in Enterprise Distributed Object Computing Conference, 2002. EDOC '02. Proceedings. Sixth International, 2002, pp. 15-26.
[10] A. Mansor, et al., " Policy-based Approach for Dynamic Architectural Adaptation: A Case Study on Location-Based System," pp. 171-176, 12-14 December 2011.
[11] E. C. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," Software Engineering, IEEE Transactions on, vol. 25, pp. 852-869, 1999.
[12] S. Chetan Shiva, et al., "An ECA-P policy-based framework for managing ubiquitous computing environments," in The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2005. MobiQuitous 2005. , 2005, pp. 33-42.
[13] 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.
[14] M. Sirjani, et al., "Modeling and verification of reactive systems using Rebeca," Fundamenta Informaticae, vol. 63, pp. 385-410, 2004.
[15] W. D. Yu and E. Nayak, "An algorithmic approach to authorization rules conflict resolution in software security," 2008, pp. 32-35.
[16] R. Agrawal, et al., "Managing disclosure of private health data with hippocratic databases," IBM Research White Paper, Januray, 2005.
[17] D. Agrawal, et al., "Policy-based management of networked computing systems," Communications Magazine, IEEE, vol. 43, pp. 69-75, 2005.

[18] R. Agrawal, et al., "Extending relational database systems to automatically enforce privacy policies," 2005, pp. 1013-1022.

## Authors

**Abdelhamid Abdelhadi Mansor** is an Assistant Professor in the Department of Computer Science in University of Khartoum, Khartoum, Sudan. His research interest in the area of Software Engineering area covers self-adaptive software system, Policy-Based Management, service oriented architecture, and Formal Methods.

**Wan M.N. Wan Kadir** is an Associate Professor in Software Engineering Department – Universtiti technologi Malaysia. His research interest covers various Software Engineering knowledge areas based on the motivation to reduce the cost of development and maintenance as well as to improve the quality of large and complex software systems.