UTM UNIVERSITI TEKNOLOGI MALAYSIA
RESEARCH UNIVERSITY

International Journal
of
Innovative Computing

Journal Homepage: http://se.cs.utm.my/ijic

Policy Overlap Analysis to Avoid Policy Conflict in Policy-based Management Systems

Abdehamid Abdelhadi Mansor
Department of Computer Science
Faculty of Mathematical Sciences
University of Khartoum
Khartoum, Sudan
abhamidhn@uofk.edu

Wan M. N. Wan Kadir, Hidayah Elias and Ahmed Elsawi
Department of Software Engineering
Faculty of Computing
Universiti Teknologi Malaysia
Johor, Malaysia
wnasir@cs.utm.my, shahlida@gmail.com, elsawi@gmail.com

***Abstract*— PobMC is an adaptive scalable approach which uses policies to control and adapt the system behaviour. Moreover, PobMC has the capability to decouple the adaptation concerns from the application code. 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. Furthermore, policy conflict may result from propagation, action composition and other constraint policies, which cannot be detected by simply comparing authorization policies. In this paper we classify our system policy conflicts to verify that policies enforced correctly. Then, we present a static analysis to address the overlap of domains when there are two or more policies are enforced simultaneously. In addition, the paper provides temporal specification patterns to detect each type of conflicts. The evaluation result shows that the performance of PobMC is better than the previous works. Less than a second is enough to perform every task as individual.**

Keywords - policy conflict, static analysis, overlap, policy classification, adaptive policy-based.

I. INTRODUCTION

Current software systems increasingly rely on dynamically adaptive software due to changes in the operational environment, user requirements, upgrades of software modules and failure or substitution of devices [1]. However, there are several challenges in developing self-adaptive systems which must be addressed appropriately. Evolution occurs with high cost, if it is not carefully planned. Ideally, the evolution should occur without interrupting the system execution. Furthermore, avoiding errors and conflicts between policies and addressing the scalability issues remain as the main challenges of current research. Adaptive software is often a complex system with a great degree of autonomy. Providing mechanisms to ensure whether the system is operating correctly is a fundamental challenge.

Policy-based approach has been well acknowledged as a methodology that provides flexibility, scalability and adaptability, control Quality of Service and security, by considering administratively specified rules. The hype of policy-based management was to commit with these features during run-time as a result of changeable network conditions resulting from the interactions of users, applications and existing resources.

Static and dynamic conflicts were considered as two classes of conflicts which need to be understood and independently managed [2]. 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. Static analysis is used by the policy compiler to detect specification errors. Moreover, to reduce run-time conflicts which occurs among rules; whose event and condition parts can be statically matched. It may not be able to evaluate policy constraints, as conflicts may depend on the run-time state of the system [3]. While dynamic analysis makes use of meta-information at runtime to detect and control potential conflicts among different policies which cannot be detected during the compilation time [4].

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.

Policy-based Managers Coordination (PobMC) is an adaptive approach which is proposed in our previous works [5, 6]. Policies are used to manage and dynamically change PobMC behavior. There are two sets of policies used to govern PobMC. First set is consisting of obligation "management policies", which are enforced by managers to govern the system behavior. Second set is authorizing "coordination policies" which are used to coordinate managers' tasks by changing the management policies.

In this paper, an overlap analysis is presented based on the policy conflicts classification that presented in our previous work [7] which proposed static analysis to address the inconsistencies of policies. In this paper also we proposed Temporal logic patterns to express and avoid overlap of domains. The Linear Temporal Logic (LTL) patterns enable the automation of a significant policy conflict analysis. Moreover, the LTL helps to introduce a number of correctness properties of avoiding the potential conflicts in the context of PobMC.

In Section 2 of this paper, we explained the details of case study, Section 3 we give more details of the policy conflict classification. Section 4 presents an introduction of overlap analysis. In Section 5 we discussed and classified policy conflict. Section 6 explains our approach to conflict detection. Section 6 discusses related work. Conclusions and further work are discussed in section 7.

II. Smart Mall System (SMALLS)

Smart Mall System(*SMALLS*) [8] 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.

In our SMALLS scenario, there are some identical clients (Smart Phones SP) that need some specific service, which is provided by three identical servers. Each client sends its requests to the corresponding manager (which plays the role as a load-balancer), instead of communicating directly with the servers. The responsibility of a manager is to distribute the incoming requests evenly among the servers. As a result, the servers receive an equal number of service requests. After finishing the requested service, the servers reply directly to the clients. Then the clients may ask for service again.

A Self Management Module (SMM) structure consists of three layers. First, Actors Layer which is dedicated to the functional behaviour of SMM and contains computational actors. Actors are governed by managers using policies to achieve predefined goals. Second, Managers Layer managers are meta-actors that can operate in different configurations each is consist of two types of policies: obligation "management" policies to direct the behaviour of actors, and authorization "coordination policies" to specify what activities a subject is permitted or forbidden to do to a set of target objects, in order to coordinate managers' tasks. Third, View Layer which provides actors required state information to the relevant managers.

III. Policy Conflict Classification

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. 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 [9, 10].

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:

a. p_i and q_j have been enforced simultaneously, then the system cannot choose a policy to enforce.
b. The execution of p_i violates the action of q_j.
c. 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).
d. 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".

IV. 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, 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.

A. 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 1 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.

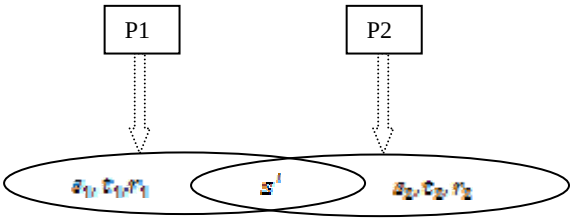

Fig. 1. Overlap of Subjects

Example 6.1 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*"

B. 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 2 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.

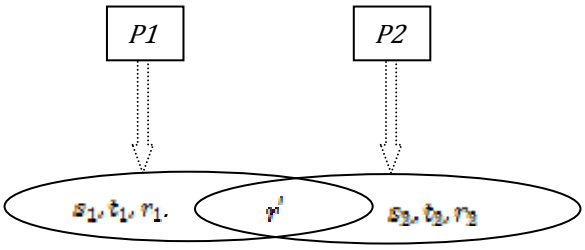

Fig. 2. Overlap of Roles

C. 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 3 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.

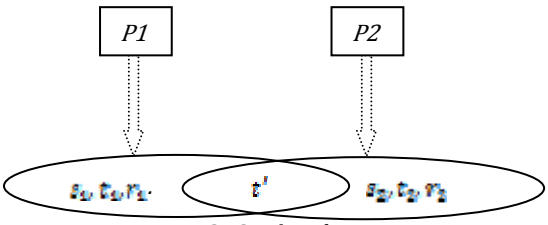

Fig. 3. Overlap of Targets

V. Check and Avoid the Overlap Behavior

Several temporal logics including the Interval Temporal Logic (ITL) [11] are capable of expressing adaptation behavior. However, these logics are both too complex and do not have direct notation support. The Adapt operator-extended LTL (A-LTL), an extension to LTL, is introduced to specify an adaptation [12, 13]. A-LTL is used in this paper to present PobMC adaptation semantics formally. A-LTL semantics is similar to LTL semantics because each LTL formula is also an A-LTL formula. Moreover, A-LTL operators are defined as those used in LTL.

To specify PobMC adaptation behavior, this work used the adapt operator ($\underline{\Omega}$). Informally, if $\mu \lambda$ and Ω are three temporal logic formula, then $\mu \underline{\Omega} \lambda$ means that initially the system satisfies μ until it stops satisfying μ and starts to satisfy λ. The Ω notation is used to specify additional safe conditions in which the adaptation occurs and logical connections between the behavior before and after adaptation. Although in some cases, the extra constraints are not used (where Ω = true). For example, $\Omega \equiv$ resource-restricted is used to constrain that the resource must be restricted when the adaptation occurs.

Overlapping of domains is 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 overlap is prevented by creating a new domain with an independent manager and all objects from the overlapping set are moved into this new domain.

When the targets of two policies overlap, there is a potential conflict arising from multiple managers of a single object, when the goals of the policies are semantically incompatible. For instance, in SMALLS, we consider the following cases:

a. if the 'security manager' requires no users should exist inside the stores while the 'LBS manager' identifies some destination positions to be the stores, at a specific time a manager is allowed to turn off all active area sensors, while another manager is unmounting file system when the device is leaving the active area".

b. any two policies which oblige subjects to do both simultaneously are in direct conflict.

There is also a potential conflict that is often tolerated, which arises from multiple managers having authority over the same object. In some cases, multiple managers of an object are forbidden on the grounds of potential conflict, e.g. generally each group of actors has a manager. In other cases it is positively encouraged, e.g. there are normally at least two managers with security administrator authority for a computer system, to cover malfunctions and holiday.

Multiple managers should be authorised to operate upon a single target, a coordinator is used to coordinate different managers' tasks in order to ensure that there is no simultaneous conflict of obligations. When such coordination is carried out between managers, it may be informal and even unformulated, but when the managers are automated it is necessary to formalise the way in which the policies are controlled, by ensuring that each policy only applies to one subject at a time.

An obligation policy which is enforced by manager in PobMC requires an authorization policy which is enforced by coordinator to permit the action. To avoid this type of conflict authorization policies should be designed to permit managers to enforce their policies if there are no conflicts caused by their policies. The target objects of ρ_i, which enforced by Mgr_v and ρ_j, which is enforced by Mgr_u should not overlap, if their goals are semantically incompatible. Also, multiple managers having authority over the same object should forbid from enforcing together.

$$= \left(Trig_{\rho_i} \wedge Trig_{\rho_j} \right) \qquad (1)$$

The Multi Managers Algorithm in Fig. 4 marks the triggered events of all managers to prevent calling the conflicting rules twice.

```
Algorithm Multi_Managers(q, Ev[], Dom[])
1: Let overlap:=false;
// in the queue q
2: Let conflict:=false; c1:=0; c2:=0;
3: while q is not empty then
4: trigger(Ev[i], Ev[j], Ev[k] );
//push event
5: Ev[i]:=i; // in the queue q
6: Ev[j]:=j;
7: Ev[k]:=k;
//mark the triggered events
8: end while;
9: for all m in Domains do
10: if O in Dom1 ∧ O in (Dom2 ∨ Dom3)then
11: Let conflict:=true;
12: Let c1:=c1+1;
//increment of type1 conflicts
13: end if;
14: if O in Dom1 ∧ O in (Dom2 ∨ Dom3)then
15: Let conflict:=true;
16: Let c2:=c2+1;
//increment of type2 conflicts
17: end if;
18: Let Ev[i]:=i+1; Ev[j]:=j+1; Ev[k]:=k+1;
19: end for
20: return (c1, c2);
```

Fig. 4. Multi Managers Algorithm

In PobMC, there will be a variety of managers having responsibility for the same object but fulfilling different roles and operating in dissimilar configurations. For instance, SenModule, SecModule, and LocModule managers in SMALLS have different responsibilities for the same APs. These overlapping responsibilities must be detected to avoid errors and policy-conflict. The managed elements, "APs," represent the target objects and the managers' modules represent the source objects. The overlap behavior starts

when more than one manager starts to enforce policies to direct the same elements. A "domain" includes a manager and the group of elements directed by this manager. Two or more domains overlap when there are objects that are members of each domain.

For instance, let m_1, m_2, and m_3 be three different managers and S_1, S_2, and S_3 be the groups managed by m_1, m_2, and m_3, respectively and let S be the set such that $S \in S_1 \cap S_2 \cap S_3$. The overlap starts if at least two managers are directing their object groups simultaneously.

A restriction condition R_{end} must be applied to safeguard the required system behavior. The restriction condition should ensure that the object source reaches a safe state. Initially, S_{sp} must be satisfied,

$$(S_{sp} \wedge (\Diamond A_{rq} \overset{\cap a}{\sqsubseteq} R_{end})) \overset{\cap b}{\sqsubseteq} true \qquad (2)$$

such that when an adaptation request A_{rq} is received, the system should start to satisfy both the target object T_{sp} and the restriction condition R_{end}

$$(\Diamond A_{rq} \overset{\cap a}{\sqsubseteq} (S_{sp} \wedge (R_{end} \overset{\cap b}{\sqsubseteq} true))) \qquad (3)$$

as depicted in Fig. 5 equation (4) showing that when the system reaches a safe state of the source objects, the system stops being obliged by S_{sp} and R_{end}.

$$\left((S_{sp} \wedge (\Diamond A_{rq} \overset{\cap a}{\sqsubseteq} R_{end})) \overset{\cap b}{\sqsubseteq} true\right) \wedge (\Diamond A_{rq} \overset{\cap a}{\sqsubseteq} (S_{sp} \wedge (R_{end} \overset{\cap b}{\sqsubseteq} true))) \qquad (4)$$

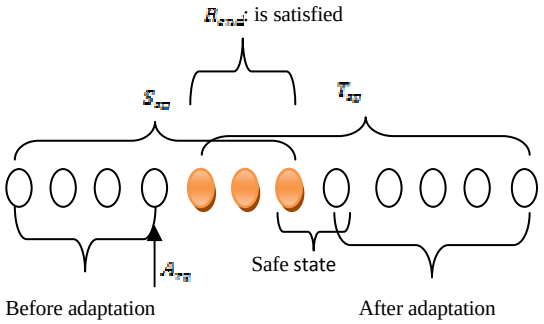

Fig. 5. An Abstraction of the Overlap Behaviour

VI. Results and Discussion

The algorithm in Fig. 4 was executed for three sets containing 100 "SecModule", 150 " LocModule", 50 " SenModule" rules. The output reported 91.8% of the selected couple of rules was overlapped. 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 was the same throughout. The average time required for the four times executions according to the execution stages were as follows:

- 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 was 2.46s.

In SMALLS there are some general policies pertaining to all users as well as more specific policies relating to staff in a department or section. As staff may also be members of many different domains. Here detecting the triple overlaps between policies with modalities of opposite signs, do not result in actual conflicts. As in the following policies:

users can not reboot () the workstations
administrators can reboot () the workstations

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 policy cycles.

VII. Related Works

There are some techniques to static conflict detection discussed in the literature. [3] 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 [14] presented an analysis using [15], 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.

Obviously there is a limitation in developing policy-based management approaches that do not provide ensuing support for detecting and resolving conflicts. While a considerable attempt at static conflict detection has been presented in [16], the very complex and crucial issue of dynamic conflict detection in a policy-based management has gone largely unresolved. Moreover, current research has revealed that there is still a large class of policy conflict which cannot be determined statically. Static conflicts detection is considered as the most important class of conflict which needs to be understood and independently managed [17]. It is used to detect specification errors and to reduce run-time conflicts which occur among rules; whose event and condition parts can be statically matched. It may not be able to evaluate policy constraints, as conflicts may depend on the run-time state of the system [3].

One approach to avoid conflicts in authorization rules is presented by [18]. 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 [19-21] for defining authorization policies for Hippocratic databases. 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 approach, the potential overlap 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.

VIII. Conclusion and Future Work

In this paper, we present a static analysis technique to address the inconsistencies, scalability when there is more than one manager controlling the system behaviour. Then, we classify our system policy conflicts, to detect the conflicts, and to verify that policies are enforced correctly. Moreover, we provide temporal specification patterns to detect each type of conflicts. The paper also discuss another aspect of policy analysis relates to determining the policies applying to a particular subject or target. Our policies explicitly identify both subject and target and the domain service maintains the list of policies applying to a domain so that it is comparatively easy to do.

The paper concentrates on the static analysis of policies, but in the near future we plan to present dynamic conflict analysis to avoid the overheads of a potentially complex analysis every time an obligation is triggered or an authorization checked. The need for dynamic analysis is that a domain membership may change dynamically and some constraints can only be evaluated at run-time as they may depend on object states or current time.

Acknowledgement

The authors would like to express their deepest gratitude to Universiti Teknologi Malaysia (UTM) for their financial support under Research University Grant Scheme (Vot number Q.J130000.7128.01H13).

References

[1] S. P. Reiss, "Evolving evolution," Lisbon, Portugal, 2005, pp. 136-139.

[2] N. Dunlop, J.Indulska, and K.Raymond, "Dynamic conflict detection in policy-based management systems," 2002, pp. 15-26.

[3] S. Chetan Shiva, R. Anand, and R. Campbell. "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.

[4] W. Zhengping and L. Yuanyao, "Automatic policy conflict analysis for cross-domain collaborations using semantic temporal logic," in Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on, 2009, pp. 1-8.

[5] A. Mansor, W. M. N. W. Kadir and H. Elyas. " Policy-based Approach for Dynamic Architectural Adaptation: A Case Study on Location-Based System," 12-14 December 2011, pp. 171-176.

[6] A. Mansor, W. M. N. W. Kadir and H. Elyas. "Policy-based Approach to Detect and Resolve Policy Conflict for Static and Dynamic Architecture " Journal of Theoretical and Applied Information Technology, vol. Vol.37 No.2, 31st March 2012, pp. 1-10.

[7] A. Mansor, W. M. N. W. Kadir and H. Elyas and S. Shafay. "Analysis of Adaptive Policy-based Approach to Detect and Avoid Policy Conflicts," vol. The 19th Asia Software Engineering Conference APSEC 2012, Hong Kong, Dec. 4-7 2012, pp.200-210.

[8] O. O. A. Oyebisi T.O, "Development of Congestion Control Scheme for Wireless Mobile Network " Journal of Theoretical and Applied Information Technology vol. Vol4No10, 2008, pp.1- 8.

[9] E. H. Sibley, R. L. Wexelblat, J. B. Michael, M. C. Tanner and D. C. Littman. "The role of policy in requirements definition," San Diego, CA, 1993, pp. 277-280.

[10] J. B. Michael, E. H. Sibley and D. C. Littleman. "Integration of formal and heuristic reasoning as a basis for testing and debugging computer security policy," 1993, pp. 69-75.

[11] A. Cau, B. Moszkowski and H. Zedan. "Interval temporal logic," URL: http://www. cms. dmu. ac. uk/~ cau/itlhomepage/itlhomepage. html, 2006.

[12] J. Zhang and B. H. C. Cheng, "Specifying adaptation semantics," 2005, pp. 1-7.

[13] J. Zhang and B. H. C. Cheng, "Using temporal logic to specify adaptive program semantics," Journal of Systems and Software, vol. 79, 2006, pp. 1361-1369.

[14] N. Khakpour, R. Khosravi, M. Sirjani and S. Jalili. "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.

[15] M. Sirjani, A. Movaghar, A. Shali and F. S. de Boer. "Modeling and verification of reactive systems using Rebeca," Fundamenta Informaticae, vol. 63, 2004, pp. 385-410.

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

[17] N. Dunlop, J. Indulska and K. Raymond. "Dynamic conflict detection in policy-based management systems," in Enterprise Distributed Object Computing Conference, 2002. EDOC '02. Proceedings. Sixth International, 2002, pp. 15-26.

[18] W. D. Yu and E. Nayak, "An algorithmic approach to authorization rules conflict resolution in software security," 2008, pp. 32-35.

[19] R. Agrawal, D. Asonov, R. Bayardo, T. Grandison, C. Johnson and J. Kiernan. "Managing disclosure of private health data with hippocratic databases," IBM Research White Paper, Januray, 2005.

[20] D. Agrawal, K. W. Lee and J. Lobo. "Policy-based management of networked computing systems," Communications Magazine, IEEE, vol. 43, 2005, pp. 69-75.

[21] R. Agrawal, P. Bird, T. Grandison, J. Kiernan, S. Logan and W. Rjaibi. "Extending relational database systems to automatically enforce privacy policies," 2005, pp. 1013-1022.