

A Generic Policy-Conflict Handling Model

Taner Dursun

Tübitak Uekae
tdursun@uekae.tubitak.gov.tr

Abstract. This paper presents a policy-conflict handling model, which is independent of both technical details of policy enforcement architecture and syntax of policy-specification language. The model primarily focuses on handling the application-specific (semantic) conflicts [1]. This model has been applied to the POLICE Policy-Based Network Management (PBNM) framework in [2-4]. The POLICE framework does not suffer from the modality conflicts [1] since it does not support negative policies whereas application-specific (semantic) conflicts are handled according to the model described in this paper. We are able to rely on a common schema in our model since almost all policy specification languages include subject, target and action concepts and the fact that all languages can be modeled in an approximately similar way allows our model to be designed independently of other details of the language.

1 Introduction

Policy-Based Network Management (PBNM) concept simplifies the management of systems by using business-level policies. A *policy* is defined as combination of one or more sets of rules that dictate actions according to conditions to help automate network, system, and service management. In other words, policies dictate the conditions that must be met before the specified actions can be taken. Research activities in PBNM community have generally focused on languages for specifying policies and architectures for managing and deploying these policies. However, achieving a complete consistency between the policies used within a PBNM framework is a difficult problem waiting to be solved. When the number of policies increase, the ability to handle conflicting policies becomes crucial. However, in the conflict detection and resolution field, a perfect solution or a standard meeting all the requirements has not appeared yet.

There has been considerable amount of research about handling modality conflicts [1], which can be detected automatically by analyzing the policies. However, proposals for handling semantic conflicts stayed limited due to the fact that researchers showed more interest in modality conflicts than in semantic conflicts, except a few groups [1], [5].

A formal model proposed in [5] aims detecting and solving modality conflicts along with some semantic conflicts by using policy constraints. A well-known study employs meta-policy concept [1] for semantic conflicts. However, both of these studies do not consider runtime conditions in conflict handling since they rely on static policy specifications. This paper introduces a unique solution to fulfill the requirements for handling semantic conflicts. The primary design goal of our model is

to make conflict handling independent of the implementation details of both PBNM system and policy language syntax.

Remainder of this paper is organized as follows. Sections 2 and 3 introduce some background for the reader. The conflict detection and resolution issues within our model are discussed in Section 4. Section 5 includes evaluation of the related work and the last section presents our conclusions.

2 PBNM Background

In order to facilitate policy management activities it is necessary to use an easy-to-understand policy language and an integrated toolkit for the deployment and enforcement of the policies within the system. Recent works like Ponder [6], [7], [8] have introduced object-oriented languages to specify general-purpose management policies. Almost all objects in these languages represent elements of managed systems and the methods of these objects function as management interfaces of these elements. The elements in a managed system may be substituted for subjects and targets. According to the well-known definition, the term *Subjects* refers to users or automated manager components, which have management responsibility, and the term *Targets* refers to objects affected by the activities of subjects. Method invocations on targets allow subjects to access targets. Policies can be defined as the set of rules managing these invocations by means of *Actions* that specify what must be performed or what is permitted according to conditions of the system.

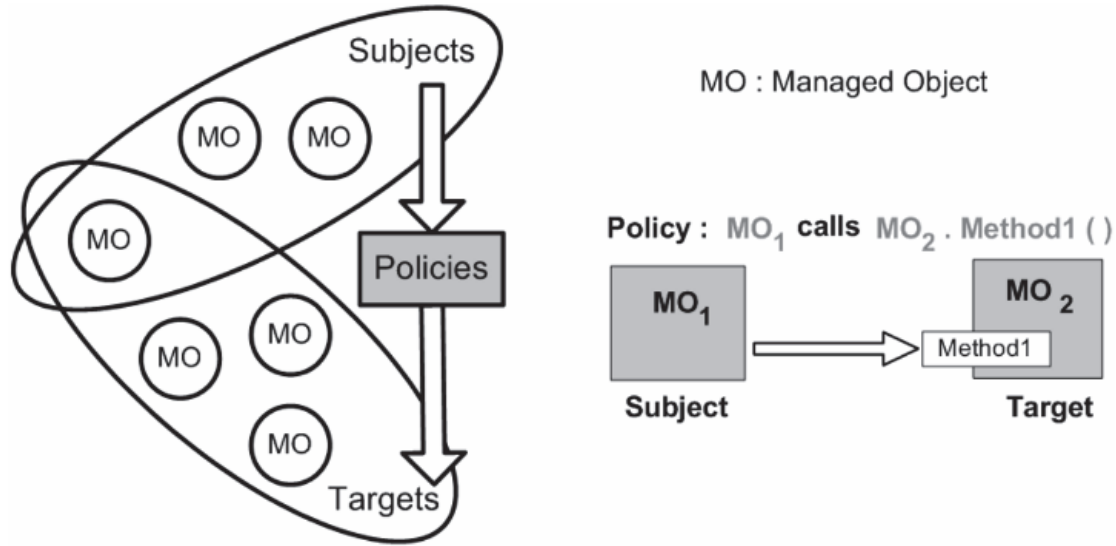


Fig. 1. Visualization of policy concept

Our conflict handling model assumes that all policies relate to objects which have interfaces defined in terms of methods as shown in Fig. 1.

Policies are basically classified into two main categories in the literature: Authorizations provide subjects with legitimate power to perform actions, Obligations give subjects the imperative to carry out actions.

In a generic PBNM system, three types of elements, Policy Decision Point (PDP), Repository and Policy Enforcement Point (PEP), are involved in the execution process. PDP acts as policy manager, which stores policy in a policy repository and distributes them to PEPs. Once policies are received from PDP, PEPs analyze them and extend their behavior. In the PEP architecture, all access requests originated from subjects to targets are intercepted and checked whether they are granted required permissions according to authorization policies. In the case of obligation policies, if their constraints are fulfilled, the subjects are ordered to perform access operations.

Policy repository is a specific data store that holds policy rules, their conditions and actions, and related policy data. In our model, the policy related data includes the information model of the managed system, so-called *System's Information Model (SIM)*. SIM models managed objects as the instances of *Model Classes*. A model class includes methods which serve as a model of the management interface of the corresponding real managed object. In addition to managed objects, the application specific (semantic) relationships (R_i) between managed objects are also modeled in SIM. Incidentally, our conflict handling model relies on these relationships. Fig. 2 shows an example of SIM.

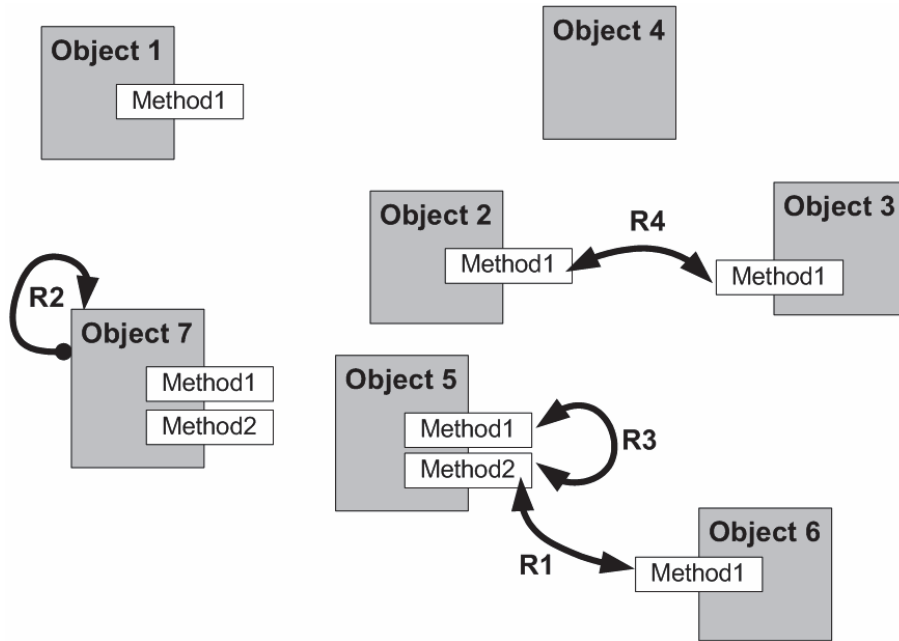


Fig. 2. Sample scenario including semantic relationships between managed objects

The content of SIM is kept up-to-date with a node discovery mechanism running on PEPs, and with administrator intervention for the nodes that are not supporting auto discovery. The vendor/deployer of a system prepares a node profile including objects for the managed elements and special relationships between them. When this node joins PBM system SIM is extended with the addition of the node profile. Thus, the managed system itself provides semantic relationships and the program pieces to be run to ensure that policies are enforced considering these relationships. Some relationships and the profiles of nodes that are not supporting node discovery are entered into the system manually.

3 Policy Conflicts

A conflict may occur if one policy prevents the activities of another policy from being performed or if the policies interfere in some way that may cause the managed objects to be put into unwanted states [9]. A policy conflict occurs when the conditions of two or more policy rules that apply to the same set of managed objects are simultaneously satisfied, but the actions of two or more of these policy rules conflict with each other. Basically, the conflicts are classified into two groups: Modality and Semantic conflicts. (A detailed conflict classification can be found in [9])

Conflict of Modalities: Modality conflicts [9] are inconsistencies in the policy specifications, which may arise from existence of both a positive and a negative authorization or obligation policies that apply to the same set of managed objects. Conflicts of modalities are independent of the exact identities of the managed objects and objectives of the policies. Without any knowledge about the managed system or activities, syntactic analysis of the policies can detect conflicts of modalities prior to deployment of the policies. Existing studies in detecting conflicts mostly focuses on modality conflicts.

Semantic Conflicts (Application-Domain Specific Conflicts): Semantic conflicts refer to the inconsistency between what is contained in the policies (i.e., which subjects, targets and actions are involved) and external criteria such as limited resources, topology of the system, or overall policies of the system. They arise from the semantics of the policies and cannot be determined directly from the syntactic analysis of the policy specifications without the description of the conflicts. The *Meta-policy* concept is proposed as a method to describe this additional information in [1,9], which defines a constraint about permitted policies (policies about policies).

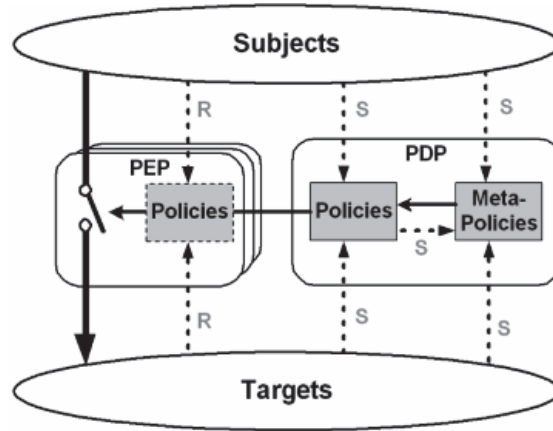


Fig. 3. Meta-policy based conflict handling

Fig.3 illustrates meta-policy based conflict handling. As shown in the figure, meta-policies are located and evaluated at PDP. The meta-policy evaluation process uses only *static data* including policy definitions and identities of subjects/targets of policies. According to the result of this evaluation, some of the ordinary policies are

eliminated due to possible semantic conflicts. Remaining policies (subset) are deployed to PEPs, evaluated and enforced there to manage the interactions between subjects and targets. The dashed lines represent inputs of related blocks, and the dark lines show the control flows. The letter R means runtime data which is related to the status of managed objects and available only at policy enforcement time, whereas the S represents static data which is related to the policy definitions and available at any time (before and after policy deployment).

4 Proposed Conflict Handling Model

The relationships between subjects or targets show implicitly the relationships of policies applied to them. In [9-12], it is claimed that a conflict may occur between two or more policies if there is an overlap between the subjects or the targets. Thus, they assert that if there are no common subjects and targets between two policies (disjoint policies), there is no possibility of conflict. However, these assumptions are true for only modality conflicts, and they may be false for semantic conflicts since there may be system-specific associations between the subjects or the targets of the policies even these policies are disjoint (implicit conflicts). In such cases, the enforcement of these policies may result in semantic conflicts. As specified in the previous section, semantic conflict detection process needs to have additional knowledge of the managed system.

Although the meta-policies are created to handle semantic conflicts, they cannot always fulfill the requirements to solve the problem. Therefore, the administrator must specify meta-policies for all possible, runtime situations, which may result in conflicts. It is almost impossible for the administrators to take into account all implicit relationships that may cause conflicts in the managed system. Moreover, in many cases, the fact that there are policies related explicitly or implicitly to each other does not mean a conflict will certainly occur at runtime because the runtime conditions required for conflicts may not be fulfilled exactly. Meta-policies are based on static data instead of runtime data. Thus, the conflicts detected via meta-policies prior to policy enforcement, may not appear at runtime.

The current design of our model does not address modality conflicts. In case of semantic conflicts, however, as an alternative to meta-policy based approach, we have developed a policy conflict handling mechanism based on the relationships between the objects of the system being managed. In order to make our model run, all associations between managed objects (subjects/targets) must be modeled in the Repository (SIM) prior to policy enforcement. By evaluating these associations, *direct or indirect relationships* between the policies to be enforced on these managed objects can also be determined. Our model wraps these evaluation processes in special scripts, so-called as *Evaluation Code* (E-Code). There are two possible ways to add E-Codes into conflict investigation process. These scripts can be obtained from the managed system itself during node discovery process performed by PEPs. In this case, the E-codes which have been placed into the managed device or system by vendor or administrator are passed on to related PEP in addition to the information model of the managed system or device. In the other case, administrators can write E-Codes via management console of PDP. The PEPs investigate the likelihood of

conflicts at runtime by executing E-Codes. In order to do this, the PEPs must first look for any relationship involving the subjects or targets of the policies for which these PEPs are responsible. If there is a relationship, PEP has to get the E-Code related to this relationship from the repository and run. E-Codes may include any command that can run on the system to determine the real conflict. If needed, the PEPs can communicate with other PEPs or PDPs. PEPs enforce their subjects according to the results of the executions of E-Codes. E-Codes make it possible to figure out the runtime picture of the conflicts accurately.

As shown in Fig. 4, E-Codes running on PEPs use both static and runtime data to perform conflict analysis. The meanings of the lines and letters in Fig. 4 are the same as those in Fig. 3. E-Codes do not only use static data such as policy definitions (represented with letter S) but also run time data (represented with letter R) related to the managed system such as attributes of managed objects (subjects, targets), subjects and targets of policies, system time, and so on.

Unlike Ponder, performing central and static conflict analysis on policies at PDP, postponing the decision on whether there is any conflict (conflict decisions) until the enforcement time makes our model more consistent and accurate on determining the real conflicts. In addition, using E-Code concept instead of meta-policies provides more flexibility because our model does not involve any restrictions for policy specification language. In contrast to meta-policies, the syntax of the E-Codes is independent of policy specification language and an E-Code can include more complex statements of any interpreted computer language such as Java, TCL. We prefer BeanShell [13] which is a script language based on Java. In contrast to the flexibility provided with E-Codes, meta-policies can contain only limited number of pre-defined statements, in order to stay analyzable by an automated software. E-codes need not to be analyzed due to the fact that they are independent of the policy language.

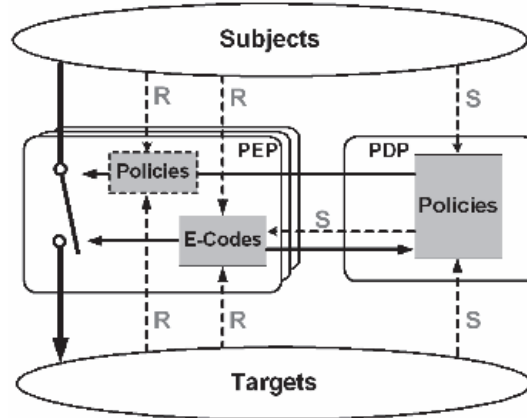


Fig. 4. Our policy conflict handling model

Our model has been first applied to POLICE policy based management framework which has been developed in TÜBİTAK [2]. In order to provide a standard format in E-Codes' statements, we have also developed an E-Code Application Programming Interface (API) including the interfaces in order to access to PEP, PDP, Repository

and other modules of PBNM framework. In order to Provide an abstraction layer between E-Codes and PBNM systems, this API includes various abstract classes such as PEP, PDP, ManagedObject, Method, Relationship, Policy, PolicyList, Filter, and so on. Any PBNM framework conforming to this API can run the E-Codes prepared for conflict handling. The main points highlighting advantages of our model over meta-policy model are listed below:

- The administrators specifying policies do not have to worry about possible conflicts (faster and easier specification). The policy conflict analysis logic is embedded into the managed system itself and performed by PEPs.
- Our policy conflict tests are executed independently of any change in the policy data stored in the Repository, but the evaluation of meta-policies is repeated in each modification (adding, deleting policies or managed objects) of the policy data stored in a central Repository.
- Our conflict tests (E-Codes) are executed if there is an application-specific relationship related to the subjects or the targets of the policies being enforced. Only The e-code related to this relationship is run.
- Our conflict test covers only a small subset of the entire policy set stored in the Repository whereas meta-policies are evaluated at PDP and on entire policy set.
- If the policies are edited (add | remove | update) frequently, meta-policy model, which has a central conflict analysis repeated after each edit operation, becomes unscalable, whereas our model remains scalable because policy edits may affect only the policy list which will be involved in the next run of any E-Code. However, if the policy set is not changed frequently, the conflict test overhead of the meta-policy based model is zero. Nonetheless, Meta-policies always require specifications and these specifications are difficult to define.
- In our conflict tests performed during enforcement operations, better suited data to detect actual conflicts (data at enforcement time) is used.
- The meta-policies can include logic for only conflict detection, whereas E-Codes can also include conflict resolution logic.

The sample problem explained below indicates how both meta-policy based model and our model handle semantic conflicts:

Conflict of duty which is a well-known application specific conflict type in literature can be expressed as “*There should be no policy pair authorizing a subject to perform two separate operations that must be performed by different subjects*” [1].

Meta-policies are used to define application specific constraints in Ponder model to limit the permitted policies in the system, or prohibit the simultaneous execution of conflicting policies. The Ponder meta-policy for detecting the conflicts similar to the one mentioned above is shown below:

```
inst meta budgetDutyConflict raises conflictInBudget(z) {
  [z] = self.policies → select (pa, pb |
    pa.subject → intersection (pb.subject) → notEmpty and
    pa.action → exists (act | act.name = "submit") and
    pb.action → exists (act | act.name = "approve") and
    pb.target → intersection (pa.target) → oclIsKindOf (budget))
  z → notEmpty ;
}
```

This meta-policy prevents a conflict of duty in which the same person both approves and submits a budget. It searches for policies with the same subject acting on a target *budget* in which there are actions *submit* and *approve*. If it is found, `conflictInBudget` event is raised which may trigger an obligation policy.

Unlike the usage of meta-policies in Ponder model, we embed the conflict detection logic into PEPs. Instead of catching conflicts by defining and analyzing meta-polices at PDP, we prefer a flexible and distributed mechanism in which policy analysis is performed by PEPs. In case of “*separation of duty*” type conflicts, the E-Code which will be executed by the PEPs is as follow:

```
boolean EC_DutySeperation ( Method submit, Method approve) {
    p1 = PDP.getAllPolicies();

    f1 = new Filter(ACTION, EQUAL, submit);
    f2 = new Filter(ACTION, EQUAL, approve);
    p2 = p1.search(f1); // find policies whose action is submit
    p3 = p1.search(f2); // find policies whose action is approve

    conflictedList= new PolicyPairList();
    //find policies from p2 and p3 not having same subject
    for (policy:p2.iterator()){
        PolicyList tmp = p3.search(new
            Filter(SUBJECT, NOT_EQUAL, policy.getSubject() ));
        //match policy with tmp policies
        conflictedList.addPair(policy,tmp);
    } // for
    return (conflictedList.size()==0);
}
```

4.1 Handling of Well-Known Semantic Conflicts

A detailed scenario including five policies is given in Fig. 5 to explain how our model is capable of handling semantic conflicts specified in [1].

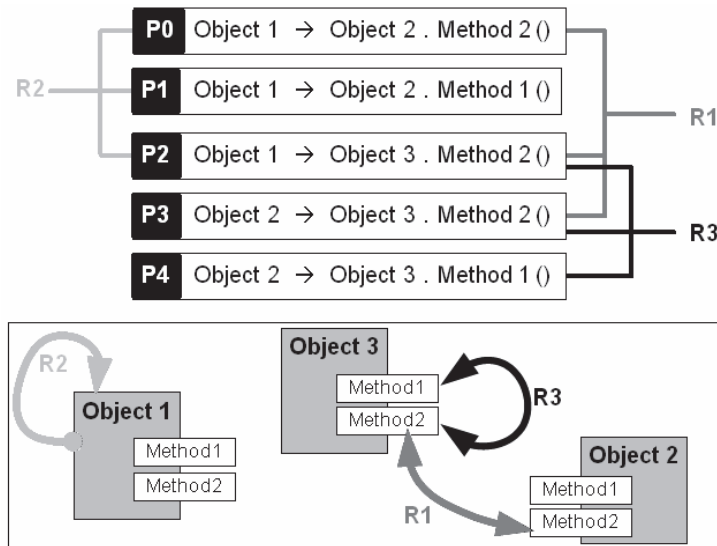


Fig. 5. Sample scenario

- **Conflict (separation) of duties:** This type of conflict occurs, if some tasks must be performed by different subjects but assigned to the same one. Let $\text{Object}_2.\text{Method}_2$ be check-signing operation and $\text{Object}_3.\text{Method}_2$ be the authorizing payment of check. In order to avoid a conflict, the system must guarantee that only different subjects can call these two methods. Therefore, a relationship specifying this exclusion is represented by $\mathbf{R}_1$ in Fig. 5. Then, the policy sets $\{\mathbf{P}_0, \mathbf{P}_3\}$ or $\{\mathbf{P}_2, \mathbf{P}_3\}$ ordering different subjects to carry out these two methods may cause a conflict. The E-Code for this conflict is already presented in the previous section.
- **Multiple Managers:** Accessing a common target, multiple subjects may cause a conflict when the management operations to be performed on this target object are not independent. Let $\text{Object}_3.\text{Method}_1$ and $\text{Object}_3.\text{Method}_2$ be dependent on each other (for example, both may modify the common variable). Then, different subjects accessing these two methods may affect the operation of each other. This case is represented with $\mathbf{R}_3$. So, the policies set $\{\mathbf{P}_2, \mathbf{P}_4\}$ and $\{\mathbf{P}_3, \mathbf{P}_4\}$ must be enforced carefully.

```
boolean EC_MultipleMng ( Method method1, Method method2) {
    p1 = PDP.getAllPolicies();
    f1 =new Filter(ACTION, EQUAL, method1);
    f2 =new Filter(ACTION, EQUAL, method2);

    p2 = p1.search(f1); //find policies whose actions is method1
    p3 = p1.search(f2); //find policies whose actions is method2

    conflictedList = new PolicyPairList();
    for (policy:p2.iterator()){
        PolicyList tmp = p3.search( new Filter(SUBJECT,
                                                NOT_EQUAL,
                                                policy.getSubject()));
        conflictedList.addPair(policy, tmp);
    }
    return (conflictedList.size()==0);
}
```

- **Conflict of Resources/Priorities:** This type of conflict occurs when the amount of resources (target objects) available is limited. Object accesses are performed in the control of the Resource Controllers at PEP's. The Resource Controllers can already eliminate this kind of conflicts. Thus, the limit on the critical resources must be determined by an E-Code similar to the one shown below so that Resource Controller can keep the usage of the critical resources under control. In order to achieve this, the resources are marked by associating them with relationships and each relationship are associated with an instance of the E-Code shown below.

```
boolean EC_ResourceLimit ( ManagedObject mol, int amount) {
    return PEP.setAccessLimit(mol,amount); //resolution of conflict!
}
```

- **Conflict of interests:** This describes a situation where a single subject has tasks relating to two different targets, and carrying out both tasks together conscientiously may be impossible. Let be a requirement that Object_1 can manipulate only either of the $\{\text{Object}_2, \text{Object}_3\}$. In this case, $\mathbf{R}_2$ is described to

specify that Object_1 can call only the methods of either Object_2 or Object_3 . So, the policy sets $\{P_0, P_2\}$ or $\{P_1, P_2\}$ ordering Object_1 to access different targets may cause a conflict and they must be enforced carefully.

```
boolean EC_Interest ( ManagedObject mo){
    //find policy pairs whose subjects is mo but targets are
    //different

    p1 = PDP.getAllPolicies();
    f1 = new Filter(SUBJECT, EQUAL, mo);
    p2 = p1.search(f1); //find policies whose subject is mo

    conflictedList= new PolicyPairList();
    //find policy pairs whose targets are different
    for (policy:p2.iterator()){
        tmp=p2.search(new Filter(TARGET,NOT_EQUAL, policy.getTarget()));

        //match policy with tmp policies
        conflictedList.addPair(policy, tmp);
    }
    return (conflictedList.size()==0);
}
```

5 Related Work

Although majority of working groups from PBNM community have realized policy conflicts can occur, neither have they distinguished between modality and semantic conflicts nor do any of them say how conflicts will be detected. However, some groups tried conflict handling by means of various methods.

The Ponder framework includes a conflict analysis tool performing static analysis of policies [1], [11]. The tool also allows various forms of precedence to be specified to resolve policy conflicts. By using this tool, static analysis is repeated for each new policy to see if it conflicts with existing ones. Such an operation results in long delays when the amount of policies grows. For semantic conflicts, Ponder model benefits from meta-policy concept.

The technology independent model proposed in [5] aims detecting and solving modality conflicts and some semantic conflicts by means of policy constraints. However, it is not flexible because it is only based on policy constraints for the semantic conflicts.

On the other hand, there has been considerable effort in logic-based approaches to specify policies, as described in [12], [15], [16], [17]. The policies are in a logical representation, existing formal calculus methods, and running tools in this area provide a means to check the consistency of policies [12], [18]. However, they do not easily map onto implementation mechanisms.

Representing policies by using IETF format [19] may allow using logic based tools, an expert system or theorem-proving approach. However, using formal methods in the analysis of policies may result in unreasonable completion time in the multi-purpose and complex systems. Moreover, static analysis of policies during specification phase may not be right approach to detect all conflicts since some kinds of conflicts may or may not arise at runtime depending on the conditions.

Another study [20] examines the conflicts that may occur in multi-agent systems. In a multi-agent system, agents share and carry out tasks collaboratively. Because of the possible dependencies between the tasks, the task distribution should be done

appropriately. While the agents perform their tasks, possible conflicts can be solved by defining policies for the agents. Conflict resolution typically entails a conversation in which agents negotiate over the interaction, generally making an agreement to either avoid the interaction through a change in the planned activities or by temporally sequencing their activities. We conclude that the model used to solve the conflicted tasks in [20] can also be employed in PBNM systems to detect and solve the conflicted policies. Moreover, our SIM concept may provide an infrastructure for the multi-agent systems because they define associations between the tasks as we do while establishing our relationships between managed objects.

6 Conclusions and Future Work

There are two main approaches about policy specification and implementation issue; implementing policies as a piece of code and interpreting them as objects [21]. *Policy as code* is more flexible using a standard programming language. However, Verma stated that in case of policy as code, it is quite difficult to determine if the policies are mutually consistent [12]. Policies specified in *policy as object fashion* are easier to analyze than policies specified as computer programs [11].

Although some authors [1], [12] suggest keeping the policy specifications simple, the majority of the working groups think that such systems, not supporting policies defined as computer programs, have several limitations and it will be inevitable to take into account more complex policies, even though their analysis is so hard.

In the conflict detection and resolution field, an excellent solution or a standard meeting all the requirements of both *policy as code* and *policy as object* fashions has not appeared yet. Our proposition, however, is able to overcome this shortage since it provides a conflict detection mechanism which is independent of the complexity of the policies. Thereby, it offers a novel conflict detection model fulfilling most of the requirements for comprehensive policy-based management. However, our model can be extended further, especially in the following areas:

- Using intelligent and/or mobile [22] enforcement agents to handle conflicts,
- Improving E-Code API especially in the issue of PEP inter-communication,
- Testing how the proposed model meets the requirements of various management domains and works in practical applications,
- Further exploration of the relationships between policies, in relation to conflicts between them,
- Running/Integrating our model within a framework such as Ponder which has already a modality conflict handling mechanism,
- Then, adding support for modality conflicts,
- Scalability analysis of E-Codes.

References

1. Lupu, E., Sloman, M.: Conflicts in Policy-based Distributed Systems Management. IEEE Trans. On Software Engineering, Vol 25. No 6 (1999) 852-869
2. Dursun, T., Örencik, B.: POLICE: A Novel Policy Framework. Lecture Notes in Computer Science, LNCS 2869 (2003) 819-827.

3. Dursun, T., Örencik, B.: POLICE Distributed Conflict Detection Architecture. Proceedings of the IEEE International Conference on Communications (ICC04), Paris, June 20-26, 2004
4. Dursun, T., Nasır, D., Örencik, B. : Relation Based Policy Conflict Detection. Proceedings of the 3rd Asia Pacific Int'l. Symp. on Information Technology (3rd APIS), İTÜ, İstanbul, Jan. 13-14, 2004, 192-198.
5. Baliosian, J., Serrat, J.: Finite State Transducers for Policy Evaluation and Conflict Resolution. Policy Workshop 2004
6. Damianou N., Dulay, N., Lupu, E., Sloman, M.: Ponder: The Language Specification – V2.2. Research Report DoC 2000/1, Imperial College of Sci. Tech. and Medicine, London, (2000)
7. Dulay, N., Lupu, E., Sloman, M., Damianou, N.: A Policy Deployment Model for the Ponder Language. IEEE/IFIP Int..Symp. on Integrated Network Management (2001)
8. Sloman, M., Lupu, E.: Security and Management Policy Specification. IEEE Network Policy based management special issue (2002) 10-19
9. Moffett, J.D: Policy Hierarchies for Distributed Systems Management. IEEE JSAC Special Issue on Network Management, Vol 11, No. 9, (1993)
10. Sloman, M.: Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, Plenum Publishing Co. Vol 2, No.4 (1994)
11. Moffett J.D., Sloman M.S: Policy Conflict Analysis. Jnl. of Org. Computing (1993)
12. Verma, D.C., Calo, S., Amiri, K.: Policy-Based Management of Content Distribution Networks. IEEE Network Policy based management special issue (2002) 34-39
13. BeanShell, <http://www.beanshell.org>
14. Chalvy, L., Cuppens,F.: Analyzing, Consistency of Security Policies. IEEE Symp. Security and Privacy (1997)
15. Ortalo, R.: A Flexible Method for Information System Security Policy Specification. 5th European Symposium on Research in Computer Security (ESORICS 98), Springer-Verlag, Louvain-la-Neuve, Belgium, (1998)
16. Barker, S.: Security Policy Specification in Logic. International Conference on Artificial Intelligence, Las Vegas, USA (2000)
17. Ahn, G.-J., Sandhu, R.: The RSL99 Language for Role-based Separation of Duty Constraints. Fourth ACM,Workshop on Role-Based Access Control, Fairfax, Virginia, ACM Press (1999)
18. Michael, J.B.: A Formal Process for Testing the Consistency of Composed Security Policies. Dept. of Info. and Software Sys. Eng., George Mason Univ., Fairfax, VA (1993)
19. Yavatkar, R., Pendarakis, D., Guerin, R.: A Framework for Policy-based Admission Control. RFC 2753, Intel, IBM, U. of Pennsylvania (2000)
20. Wagner, T., Shapiro, J., et al: Multi-Level Conflict in Multi-Agent Systems. Proc. of AAAI Workshop on Negotiation in Multi-Agent Systems (1999)
21. Martinez, P. et al.: Using the Script MIB for Policy-based Configuration Management. IEEE NOMS (2002) 203-219
22. Pham, V.A., Karmouch, A.: Mobile Software Agents: An overview. IEEE Communication Magazine, July (1998) 26-37