Managing Pervasive Systems using Role-based Obligation Policies

Chetan Shankar and Roy Campbell
Department of Computer Science, University of Illinois at Urbana-Champaign
{chetan, roy}@cs.uiuc.edu

Abstract

Pervasive systems are complex distributed systems containing heterogeneous and mobile devices, services and applications. Policy-based management is an effective approach for managing these systems. The dynamism of a pervasive system due to mobility of devices and applications makes policy specification and management a difficult problem. Policies need to be modified when devices and applications are added, removed or migrated across pervasive systems. This causes significant policy management overhead and improper management may lead to policy conflicts, cycles and other undesirable system behaviors.

In this paper, we present a role-based approach for managing pervasive systems. Roles group related entities and enable logical separation of entities from policies. Policies and entities are assigned to roles and every entity belonging to a role enforces policies assigned to that role. Policies and entities can be independently modified and this flexibility simplifies policy management. Roles are further organized into hierarchies that enable policy reuse. We present our management framework and middleware based on roles that is being developed for managing our prototype pervasive system.

1. Introduction

The complexity of pervasive systems necessitates a management system to orchestrate the functioning of various entities according to well-defined system guidelines. Commercial deployment of these systems in smart buildings and aware infrastructures requires a management middleware to enforce organizational requirements. Policy-based management has been found to be an effective approach for managing network switches [1], distributed systems [2] and content distribution networks [3]. Some projects have also used policies to manage pervasive systems [4, 5] with encouraging results.

Our prototype pervasive system is a physically-bounded collection of devices, applications and services, called *Active Space* [10]. A distributed meta-operating system provides essential services for resource discovery and sharing, event communication, fault detection and various other functionalities in an active space. We are developing a policy-based management system to manage the configuration of resources of an active space. Policies guide the behavior in different situations such as when mobile devices are brought into the active space, applications are started, device file systems are mounted and so on [4, 6].

A management policy, also called *obligation policy,* [7] is defined as a set of obligation rules specified using the Event-Condition-Action (ECA) framework. These rules specify the management actions to be executed when specified events are observed and corresponding conditions are satisfied. A typical rule would read as "if a guest user enters an active space, start authorization application". When any user wearing a location-sensing badge enters our active space, the location system generates an event. The management system checks to see if the user is a guest and if so starts the authorization application. The ECA rule is shown below:

```
on(ObjectEnter (Person p, Space s))
if (rolePerson(p) = = "guest")
do(startAuthorizationApp(s));
```

We have extended the ECA framework with pre- and post-conditions to enable advanced reasoning [4, 6, 11].

Entities of an active space are constantly added or removed necessitating frequent policy redesign. This increases the overhead associated with managing policy assignment and revocation. For example, devices and applications in an active space need to send periodic *heartbeat* messages on a well-defined channel to announce their presence in the space. If the device is owned by a guest user, the device should report its location periodically to the active space. Therefore, when a new device is brought into the space the device must be configured to fulfill the above obligations. Rules must be designed for the above obligations making administration of an active space a laborious process. Similarly, when a device leaves the space these rules must be revoked.

Though active space entities change frequently, the types of entities present in a space remain fairly unchanged. For example, cell phones, PDAs and laptops are added and removed from an active space as users enter and leave the space but the entity type *Mobile Device* is always present in the set of types supported by an active space. In such situations, role-based management is an appropriate technique of decoupling entities from policy specification by introducing a logical separation using *roles*. A role groups related entities based on certain common characteristics. Therefore, we introduced the notion of roles, based on entity types, for active space management. Role-based management was proposed in [7] for managing distributed systems. Roles were created for different kinds of administrative positions and obligation policies were assigned to these roles. Users and automated agents assigned to these roles were required to fulfill the role obligations. We extended this notion of roles to entity types such as devices, applications, services and so on by creating a role for each type. Obligation policies and entities are assigned to these roles and entities are required to fulfill role obligations. For example, the obligation to send periodic heartbeats can be assigned to the roles *Device* and *Application.* When a device, such as a laptop, is brought into an active space and assigned to the role *Device*, it takes up the obligation of sending heartbeats.

Role-based access control (RBAC) [8] is used extensively in computer security to assign access permissions to users using authorization policies. RBAC uses a notion of subjects, which are processes running on behalf of users [9], and provides a one-to-many mapping from users to subjects. Permissions assigned to users percolate to subjects. Role-based management (RBM) extends the role concept to obligation policies. While authorization policies are based on users, obligation policies should be based on users and entities since different entities have different obligations to fulfill. Therefore, active space roles are based on entity types and user types. In this paper, we generalize our discussion by only considering entity roles since users can also be viewed as entities with regard to roles.

Roles are organized into a hierarchy that enables policies to be inherited across roles thus facilitating policy reuse. Roles can also contain constraints that place additional restrictions on roles such as limiting the number and type of obligations applied to the roles and defining role-based priorities for conflict resolution.

In section 2, we describe our role-based management framework with brief descriptions of the policy language, enforcement process and role models. In section 3, we discuss many challenges that are currently being addressed and suggest possible solutions. We describe the architecture and required middleware support in section 4 and finally conclude the paper in section 5.

2. Role-based Management Framework

2.1. Policy Structure

Our management framework uses policies that are formulated as sets of ECA rules of the form

on *event* **if** *condition* **do** *action*

A policy rule is read as: "When event occurs in a situation where condition is true, then execute action". A policy is loaded into the management system that subscribes to various rule events. When an event occurs the policy is evaluated to determine triggered rules and execute corresponding actions. The action is a call to a method in a library of actions where each action is annotated with a pre-condition and a post-condition by the action developer (programmer). These rules, called Event – Condition – Precondition – Action – Postcondition (ECPAP) rules, enable complex reasoning such as conflict analysis due to side-effects of actions [4], termination analysis [6], monitoring successful enforcement of rules and ordering rule enforcement when multiple rules are simultaneously triggered [11].

A typical ECPAP rule is shown in figure 1. The pre- and post-conditions are shown italicized in braces above and below rule action for reading convenience and are not specified as part of the rule.

```
on(ObjectEnter (Person p, Space s))
if(rolePerson(p) = = "guest")
{statusSpace(s, running)}
do(startAuthorizationApp(s));
{statusApp(authorizationApp(s), running)}
```

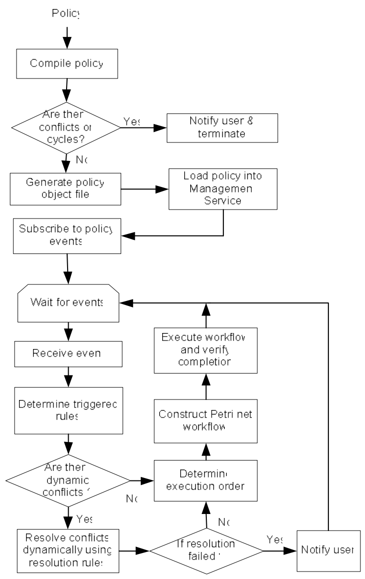
Figure 1. Example ECPAP rule

2.2. Policy Enforcement

The flowchart of policy enforcement is shown in figure 2. Policies are compiled and statically checked for conflicts and cycles [4, 6]. If the policy is free of static errors a policy object file is generated. This is loaded into the management system that stores it in a policy store. The management system is implemented as a service of our active space. The management system subscribes to various rule events of the policy and waits for event occurrence. When an event is received, the policy is analyzed to determine triggered rules. The rules are checked for dynamic conflicts [4] and resolved using conflict resolution rules. The conflict-free rules are analyzed to determine the order of enforcement [11] and a workflow of rule actions is generated. This workflow is executed by an execution engine and the management system waits again for events to occur.

2.3. Roles

In role theory, a role is defined as a collection of rights and duties [7]. These correspond to authorization and obligation policies, respectively. We do not consider authorization policies in this paper.



Polic
Compile policy
Are ther conflicts o cycles?
Ye
Notify user & terminate
N
Generate policy object file
Load policy into Managemen Service
Subscribe to policy events
Wait for events
Receive even
Execute workflow and verify completion
Determine triggered rules
Construct Petri net workflow
Are ther dynamic conflicts
Determine execution order
N
Ye
N
Resolve conflicts dynamically using resolution rules
If resolution failed
Ye
Notify user

Figure 2. Flowchart of policy enforcement

We define four different role-based management models that correspond to the models of RBAC [8]. The base model, RBM_0 contains only roles. RBM_1 supports role hierarchies and enables policy inheritance. RBM_2 supports role constraints that specify restrictions on roles. RBM_3 supports both role hierarchies and constraints. Currently, our framework supports RBM_1 and we are extending it to support RBM_3.

Roles are created based on entity types and assigned rule templates. Rule templates are ECPAP rules designed with reference to roles. When an entity is assigned to a role, the rule template gets instantiated based on the entity reference. The template for the rule to send a heartbeat is shown in figure 3.

```
on(TimerEvent())
if(true)
{true}
do(sendHeartbeat(Role));
{true}
```

Figure 3. ECPAP rule template

The syntax of rule templates is similar to that of rules except that entities are replaced by the *Role* keyword. When an entity is assigned to a role, the role management system generates a rule by substituting the *Role* keyword with the entity reference.

Roles are managed by an active space service called *Role Manager*. A rule template can be added and removed from a role dynamically when the active space is running using the role manager interfaces. When an entity is instantiated or brought into an active space, it is assigned to a role. Currently, we use manual techniques for role assignment. For example, when an application is started in an active space, the user chooses the role for the application through a user interface to the role manager. The role manager instantiates the rule templates and adds them to the management system. In addition, the role manager maintains a list of entities assigned to a role for all roles and is used for revocation of rules.

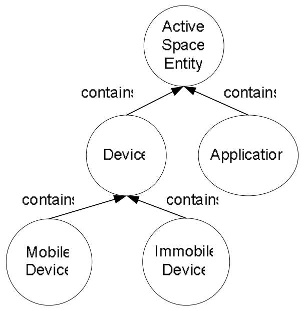


Figure 4. Partial active space role hierarchy

2.4. Role Hierarchies

Roles may contain some or all obligations of another role. For example, both devices and applications need to send heartbeats to periodically announce their presence. This is a common obligation shared between the roles *Device* and *Application*. Mobile devices in addition have to periodically send their location information in addition to sending out heartbeat messages. Therefore, the obligations of the role *MobileDevice* is a super set of that of *Device*. It would be convenient if common obligations can be specified once and reused by multiple roles. Organizing roles hierarchically enables obligation reuse and makes policy management simpler. A role hierarchy defines roles that have unique obligations and may implicitly include the obligations associated with another role [9]. The role hierarchy is defined using a partial order called "contains". Role r_1 *contains* role r_2 if the set of obligations of r_1 is a superset of that of r_2. r_1 is called a *sub-role* of r_2 and r_2 is called a *super-role* of r_1. Sub-roles inherit obligations from super-roles. A partial hierarchy of roles used in our active space is shown in figure 4. The role *ActiveSpaceEntity* is a super-role for roles *Device* and *Application*. Therefore, the common obligation to send heartbeats can be assigned to the role *ActiveSpaceEntity*.

2.5. Role Constraints

Role constraints define restrictions on roles and role hierarchies. $RBAC_3$ uses constraints for separation of duty, limiting the number of users assigned to a role and enforcing various other organizational requirements [8]. Constraints can be used very effectively in role-based management to specify what rule templates can be inherited and overridden. Constraints can also be used to limit the number of entities and type of obligations assigned to a role and to specify role-based priorities for conflict resolution.

3. Challenges

The first version of the management system was implemented without roles [4, 6, 11] and is currently being extended for role-based management. Several challenges need to be addressed before the system can meet the requirements of the RBM_3 model described in section 2.3. We discuss some of the challenges and suggest possible solutions in this section.

3.1. Role Constraint Logic and Language

In order to enable role constraints, an appropriate logic and constraint language should be identified. In our work on conflict analysis [4], we used first-order predicate logic (FOPL) to specify conflicting system states as constraints that should not be violated by the policy system. Currently, we are investigating the applicability of FOPL for specifying role constraints.

3.2. Policy Analysis

Our management system uses various algorithms to detect conflicts and cycles in policy rules [4]. These algorithms are based on entity references. Extending these algorithms for rule templates is an interesting problem. Rule templates specify rules with reference to roles and role-based analysis techniques have to be developed. Further, rules are inherited from other roles in the role hierarchy and therefore the analysis should scan the hierarchy to determine conflicts and cycles.

3.3. Rule Template Overriding

When rule templates are inherited from other roles certain rules may need to be overridden to meet role-specific requirements. For example, the *MobileDevice* role inherits the heartbeat rule template from the *ActiveSpaceEntity* role. Therefore, the heartbeat rate is dictated by the inherited template. If mobile devices need to send heartbeats at a lower rate to conserve battery power a new rule template should be designed that overrides the inherited template. This problem is analogous to method overriding of object-oriented programming. We need to define conditions when rule of a super-role can be overridden by a rule of a sub-role.

3.4. Scalability and Rule Consolidation

Our management system currently consists of a central policy store and evaluator. When an entity is assigned to a role, rules are instantiated from the rule templates and added to the policy store. Multiple entities assigned to a common role generate multiple instances of the same rule template. This increases the policy evaluation time when an event occurs. In addition, policy enforcement becomes complex as multiple instances of the same management action need to be executed for different entities. The overall response time of the management system increases significantly with the number of entities resulting in poor scalability.

One possible approach to reduce the number of rule instances generated is by rule consolidation. Rule consolidation combines multiple instances of a rule template into a single rule. For example, if two applications are assigned to the *Application* role, two rules are created to send heartbeats periodically from the applications. Since both rules are triggered by the same event and conditions they can be combined into a single rule whose action triggers heartbeats from both applications. Rule actions should be designed to accommodate multiple entities but this approach greatly reduces policy evaluation time and rule enforcement complexity.

3.5. Rules with Composed Events

Currently, the ECPAP rule framework only supports single events in each policy rule. Some management actions may be based on occurrence of multiple events. The ECPAP framework should be extended with an event definition language that supports different event-based operations. Some issues need to be addressed before extending the system for multiple events:

- Since an active space uses asynchronous communication, what should be the event reception model? In particular, after an event is received how long should the event reception system wait for the next event before evaluating the policy?
- If rule r_1 uses event e_1 and rule r_2 uses event e_1+e_2, does this imply that whenever r_2 is triggered r_1 should also be triggered? What are the event evaluation semantics?

4. Architecture and Middleware Support

The management system is implemented as an active space service. The system architecture is shown in figure 5. The management service contains a coordinator component that coordinates the interactions among various components of the service. The role manager generates a policy and the policy compiler compiles it to generate an object file. The action library contains a library of actions that can be invoked from the action part of the policy rule. The policy loader loads the generated object file into the management service. The service stores the policy rules in a policy store implemented as a simple database. The event receiver is responsible for subscribing to events and receiving them when they occur. The event receiver verifies the types of the parameters in the events and notifies the management coordinator of the event occurrence along with the parameters. The management coordinator determines the triggered rules, and resolves dynamic conflicts. The workflow generator constructs a workflow of actions from triggered rules, which is executed by a workflow execution engine.

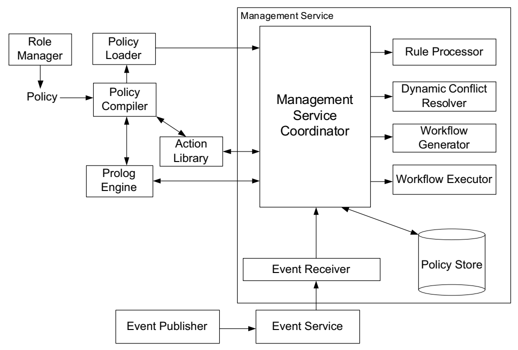


Figure 5. Management system architecture

4.1 Parameterized Events

ECPAP rules support events containing typed data. Parameterized events are useful to convey additional information about events and enable more expressive rules. Delivering parameterized events requires additional middleware support to marshal and unmarshal event parameters, extend events with data and enable entities to send events based on well-defined event signatures.

4.2. Role manager

Role manager is responsible for managing role hierarchies and is implemented as an active space service. It stores the active space roles in a role store and provides interfaces to add and remove roles and rule templates and assign entities to roles. Role manager also instantiates rules from rule templates when any entity is added to the system. We are currently developing a static analysis tool as a component of the role manager to perform static conflict and cycle analysis on the rule templates in the role store.

4.3. Action Interfaces

Management action interfaces should be standardized for a space and managed entities should support these interfaces. Currently, we have designed an interface library that lists the management actions supported by the active space. Rule designers can use this library to determine the appropriate action to invoke in a management rule.

5. Conclusion

The dynamic nature of active spaces makes policy design and maintenance a laborious process. Policies have to be modified whenever active space entities are added or removed from the space. Our experience with active space design and usage revealed that though the entities of a space change frequently the types of entities present in a space generally remain unchanged for significant periods of time. This motivated us to extend our policy framework to a role-based approach.

In this paper, we presented a framework and preliminary implementation details of a role-based management system for active spaces. Our initial design demonstrated that policies based on roles reduce maintenance complexity of policies and is well suited for managing active spaces. We are currently extending roles to support constraints and investigating the discussed challenges.

Reference

[1] R. Bhatia, J. Lobo and M. Kohli, “Policy Evaluation for Network Management”, *INFOCOM 2000*, pp. 1107-1116.
[2] M. Sloman, “Policy Driven Management For Distributed Systems”, *Plenum Press Journal of Network and Systems Management*, Vol 2, No. 4, Dec. 1994, pp. 333-360.
[3] K. Amiri et al., “Policy based management of content distribution networks”, *IEEE Network Magazine*, 2002.
[4] C.Shankar and R.Campbell, “A Policy-based Management Framework for Pervasive Systems using Axiomatized Rule Actions”, *Fourth IEEE International Symposium on Network Computing and Applications (IEEE NCA05)*, MA, July 2005.
[5] L. Kagal et al., “A Policy Language for a Pervasive Computing Environment”, *Policy Workshop*, June 2003.
[6] C.Shankar, A.Ranganathan and R.Campbell, “An ECA-P Policy-based Framework for Managing Ubiquitous Computing Environments”, *Mobiquitous 2005,* San Diego July 2005.
[7] E.Lupu, “A Role-Based Framework for Distributed Systems Management”, PhD Thesis, Imperial College, London, 1998.
[8] R. S. Sandhu et al., “Role-Based Access Control Models”, *IEEE Computer 29(2): 38-47*, IEEE Press, 1996.
[9] D.F. Ferraiolo et al., “Role Based Access Control: Features and Motivations”, In *Proceedings of the 11th Annual Computer Security Applications Conference (CSAC '95)*, 1995.
[10] M.Roman et al., “Gaia: A Middleware Infrastructure to Enable Active Spaces”, In *IEEE Pervasive Computing*, pp. 74-83, Oct-Dec 2002.
[11] C.Shankar and R.Campbell, “Ordering Management Actions in Pervasive Systems using Specification-enhanced Policies”, *PerCom 2006: Fourth IEEE International Conference on Pervasive Computing and Communications*, Italy, March 2006.