한국시뮬레이션학회 논문지
Vol. 17, No. 4, pp. 209-218 (2008. 12)

xPMP : UML-based High-Level Modeling of Policy-Driven Management Applications

Tran, Doan Thanh[1] · Eunmi Choi[1†]

xPMP – 정책 기반 관리 어플리케이션의 상위 단계 모델링

쩐도안타인 · 최은미

ABSTRACT

The Unified Modeling Language becomes popular to specify, visualize, construct, and document software-intensive systems, especially in supporting the design phase of software engineering. Most of designs in UML have focused on firm designing of software system structure. Recently, some researches have raised additional demands in many emerging complex software systems, such as aspect-oriented design. In this paper, we work on the dynamic aspect of policy-driven architecture. We present a UML-based high-level modeling of policy-driven management which is applicable in various application domains. In order to manage a number of activities of applications, dynamics and flexibility should be supported with policies consistently on different resources in the same context. Thus, a methodology of meta-modeling to represent dynamic aspects of policy-driven architecture is studied. Based on our methodology, we could achieve meta-modeling to develop a number of policy-driven management applications.

Key words : Policy-driven management, Modeling language, Model integrated computing

요 약

UML(Unified Modeling Language)은 소프트웨어 설계 단계에서 디자인을 구체화하고, 가시화하여 구성하고, 문서화하는데 일반적으로 사용되고 있다. UML 대부분의 설계는 소프트웨어 시스템 구조의 설계에 초점을 맞추고 있다. 최근에는 aspect -oriented 설계와 같이 복잡한 소프트웨어 시스템을 설계하기 위한 부가적인 설계의 요구 사항들이 등장하고 있다. 본 논문에서는 policy-driven 아키텍처의 동적인 양상을 연구한다. 다양한 어플리케이션 도메인에서 적용할 수 있는 policy-driven 관리 어플리케이션의 상단계에서의 모델링을 제안한다. 어플리케이션들의 많은 활동들을 관리하기 위해서, 같은 상황에서의 다른 자원들에게 정책(policies)과 함께 활동성과 유연성이 지원되어야 확장이 가능하다. 이러한 흐름에서, policy-driven 아키텍처의 동적인 양상을 보여주기 위해 메타 모델링 방법론을 연구하였다. 본 방법론에 기반으로 하여 본 논문에서는 policy-driven 관리 어플리케이션 영역에서의 메타모델링을 설립하였다.

주요어 : Policy-driven 관리, 모델링 언어, 모델 통합 컴퓨팅

* This research was supported by the research program in Kookmin University and the MKE(Ministry of Knowledge and Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA (IITA-2008-C1090-0804-0015).

2008년 11월 13일 접수, 2008년 12월 6일 채택

[1)] 국민대학교 비즈니스IT학부
주 저 자 : Tran, Doan Thanh
교신저자 : Eunmi Choi (최은미)
E-mail; emchoi@kookmin.ac.kr

1. Introduction

As large number of services provides collaborative combination in service-oriented era, software development requires service-oriented architecture (SOA) and a development process supporting SOA. Information technology is scaled out to the global scope in scale and complexity as well as in IT organization. Newly created applications fit into pervasive ubiquitous computing environment over world-wide geographical spread. This leads to the needs of emerging complex applications which are well-supported by applying SOA and the development process supporting it. These kinds of applications require a large amount of complicated services that needs to be managed consistently towards number of aspects by keeping the benefits of dynamics and flexibility of the systems. Policy-driven management is considered as a desirable management solution in these requirements. In order to support the dynamic aspect of SOA and the service-oriented development process, dynamic requirements need to be applied to model management systems even from the design level. Thus, we study a high-level modeling methodology for the dynamic aspect of policy-driven architecture.

In this paper, we introduce Extensible Policy Management Profile (xPMP) which is a UML-based high-level modeling of policy -driven management applications. The profile can be used to model variety of applications from distributed systems, enterprise scale systems to mobile environments. For prototype, we apply xPMP in a case study in which we create a policy-driven management system in ubiquitous service domain. By defining stereotypes in UML form and using meta-modeling of Generic Modeling Environment (GME) [2] to create the meta-model, the xPMP provides meta-modeling paradigms to establish high-level models for policy-driven management applications including application service management and policy modeling. Its high-level modeling is able to manage a number of activities of u-services by using dynamic policy management so that dynamics and flexibility can be consistently achieved on different resources in the same context with policy-driven management. Based on the meta-modeling paradigms, we also provide modeling methodology to develop from system interaction models to policy management model. This methodology gives a development process to provide the dynamic aspect of policy-driven architecture to a model for general service and application. By using the xPMP and its modeling methodology, we can achieve meta-modeling design and development of policy-driven management applications. In this paper, we show an example of meta-modeling development in ubiquitous service application.

The rest of the paper is organized as follow. We first summarize the related work in Section 2. We then show our policy-driven modeling methodology overview in Section 3. In Section 4, we describe The Policy-Driven Management Architecture for Ubiquitous Services Domain. Finally, we conclude this paper in section 5.

2. Related Work

In order to develop a meta-modeling, recently, the UML has been used to model aspect-oriented design [4]. However, the approach is just limited at describing the behavioral features of the operation or method in behavior diagram at the design phase. If the system need to change its behaviors during run-time, UML 2.0 could not provide the solution. CUP 2.0 [3] presents a UML 2.0 profile of context-sensitive user interfaces, which provides stereotypes and corresponding tagged values to increase support for the expression of the models. The limitation of Cup 2.0 are: first, it cannot describe the system constraints, the only thing it support is the context-aware design for the user interface; second, the automation process is till in a progress work. Compositional MDA [7] is a compositional subset of UML 2.0 activity diagrams that support modeling the coordination aspect. The authors develop a language that allows defining the transformation from platform-independent models (PIMs) into platform-specific models (PSMs). The language focuses on the behavioral part of coordination. The major focus of Compositional MDA is to restrict the system modeling process in a compositional subset of UML notations. Thus, it is straight forward to transform from PIMs to PSMs. However, it limits the system to describing complex activities which are normally non-compositional. Multimedia Modeling Language (MML) [8] is a visual modeling language supporting the design process in multimedia application development. It integrates the results of two different research lines: application-oriented multimedia modeling and model-based user interface development. The authors show how MML aims to integrate the different developer roles in multimedia application design. Because MML is designated to multimedia applications, it can not support modeling a general system. As a configurable tool suite, GME [9] is a domain-specific modeling environment that can be configured and adapted from meta -level specifications. It has the same concept with Meta-Object Facility (MOF) [11] which is use to create meta-model for any specific domains. Instead of focusing only on modeling for software engineering like MOF, GME provide a general visual modeling environment to create a meta-model of any domain in any context and even its model instance. Besides, we can use GME tool suit to transform between different model dynamically by providing the rule mapping through GReAT tool. Because of the completeness of GME in Model Integrated Computing, we choose GME as the base to build up xPMP.

There are a number of researches to model a policy-driven management. GPM [10] provide a management solution of model-based, policy-driven, and agent-oriented for Applications and Services. However, their model is created by their own specification instead of a standardized language like UML and thus it is difficult to integrate with different applications and services in different platforms. Besides, their approach is just adequate for a class of policies which are result-driven rather than other aspects such as event-driven or behavior-driven. xPMP is designed with the goal is to model any type of policy for software systems.

There are different researches focusing on policy driven management. Chisel [1] presents an open framework for dynamic adaptation of services using reflection in a policy-driven, context-aware manner. The system is based on decomposing the particular aspects of a service object that do not provide its core functionality into multiple possible behaviors. As the execution environment, user context and application context change, the service object will be adapted to use different behaviors, driven by a human-readable declarative adaptation policy script. SPECSA [5] presents an optimized, policy-driven security architecture for wireless enterprise application. It is scalable, extensible, flexible, and customizable. It supports end-to-end client authentication, data integrity and confidentiality between wireless clients and enterprise servers. The security services are customized and controlled by security policy that specifies several security related attributes, classifies, network data based on sensitivity and content and provides an abstraction for the communication and messaging between the client and the server. In [6], Flegkas et. al discuss the policy management for IP Differentiated Services networks and they focus on the functionality of the network dimensioning component. They view policies as a mean of extending the functionality of management systems dynamically, in conjunction with pre-existing hard-wired management logic. None of these policy management approaches provides a general modeling method for applications and services management design. With xPMP, we can provide a methodology to model the Policy and the Policy-driven Management System so that software developers can easily design, implement, and maintain their Management systems in Policy-driven manner. Through GME tool suit, the software developer can generate automatically the policies for their systems.

3. Model Overview

In UML 2.0, there are three prominent parts of a system model: functional model, object model, and dynamic model. *Functional Model* shows the functionality of the system from the user's point of view. It includes Use case diagrams. *Object Model* shows the structure and substructure of the system using objects, attributes, operations, and relationships. It includes Class Diagrams. *Dynamic Model* shows the internal behavior of the system. It includes sequence diagrams, activity diagrams, and state machine diagrams. In design

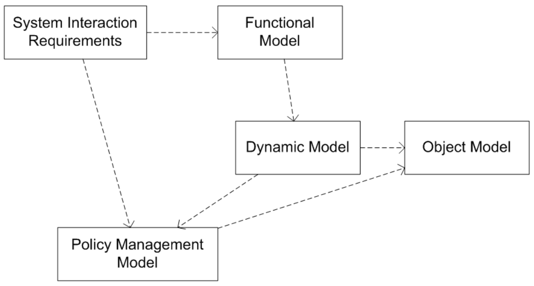


Fig. 1. System Modeling with xPMP in Software Design Phase

phase of software engineering, the *System Interaction Requirement* is collected at the first step. Based on this information, the designer will design the *Functional Model* of the system. From the Functional Model, the Dynamic Model is created through a comprehensive analysis and design process. Finally, the *Object Model* is built based on Dynamic Model.

The xPMP focuses on providing paradigms graphics interface (through GME tool suit) that support constructing *Policy Management Model* instance for policy-driven management systems. The *Policy Management Model* is created by a methodology that converts the non-functional constraints from *System Interaction Requirements* and functional-constraints from *Dynamic Models* into policies and rules in *Policy Management Model*. In addition, the *Object Model* is designed based on the *Policy Management* Model to support policy-driven management. Figure 1 shows the visualization view of relationships of *Policy Management Model* with other models.

In order to provide a framework for designing and implementing Policy-driven Management System with support of Model-driven development, we introduce the eXtensible Policy Management Profile. The xPMP provides two meta-models for designing policy-driven management system: The Meta-model for Policy Model and the Meta-model for Object Model of Policy Management systems. We used Generic Modeling Environment (GME) [9] to develop our meta-models.

3.1 Model Stereotypes

The Policy Management Model is composed of two

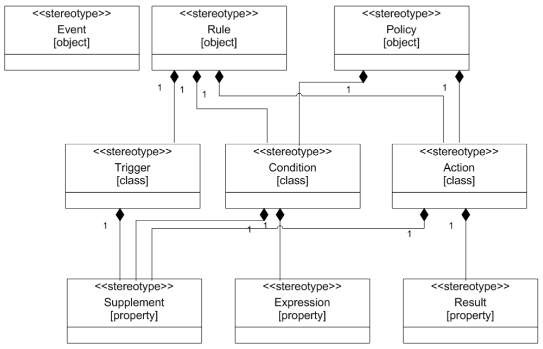

<<stereotype>> Event [object]
<<stereotype>> Rule [object]
<<stereotype>> Policy [object]
<<stereotype>> Trigger [class]
<<stereotype>> Condition [class]
<<stereotype>> Action [class]
<<stereotype>> Supplement [property]
<<stereotype>> Expression [property]
<<stereotype>> Result [property]
1
1
1
1
1
1
1
1
1
1
1

Fig. 2. Stereotypes of xPMP

types of system instructions: *policies* and *rules* to manage the system and the events that invoke these instructions of the system.

A *policy* is a definite plan or a course of actions selected among alternatives and in the given condition to guide and determine present and future decisions of a specific object in a managed system. The decisions are either to change status of objects or execute appropriate actions in the system. The policies govern the behaviors of the managed system. A *policy* refers to the set of externalized properties of a target system. Policy can be applied to all levels of modeling, e.g., business-level policy and platform-specific policy. In a specific domain, policy describes the system behavior and overall control aspects.

A *rule* is a control expression of process, operation, and action. A *rule* carries out an action systemically when it is triggered and meets the condition. *Rules* define, constrain or validate some aspect of a system through the evaluation of conditions and context of the rule invoker. A rule shows an action-oriented behavior, while a policy represents a requirement of user's plan. A rule reflects a specific functionality, while a policy reflects a nonfunctional requirement.

Figure 2 shows the stereotypes provided by xPMP for Policy Management Model. The stereotypes <<event>>, <<policy>>, and <<rule>> are used to describe the interaction between the application system and the policy system. An event causes the execution process on different policies and rules in the system. Policy as mentioned above, governs the behaviors of the managed system. Therefore, it is composed of <<condition>> and <<action>> classes. There is no specific trigger for these system instructions. The system will automatically check if a behavior in the system is governed by any policy. Rule is more specific than policy and it requires a trigger to be executed. It is composed of <<trigger>, <<condition>>, and <<action>> classes. These classes require <<supplement>> properties to be executed. Besides <<supplement>> properties, a <<condition>> class can accept <<expression>> properties in order to support logical expression on <<condition>> class returning value. Likewise, a <<action>> class can provide <<result>> properties to show the results of execution for further management or logging purposes.

3.2 The Policy Meta-model for Policy Modeling

The xPMP supports modeling policies and rules of policy-driven management system by providing the meta-model for *Policy Management Model*. We used GME to specify the objects, classes and properties of *Policy Management Model* and the relationships between them by using UML specification.

Figure 3 shows the meta-model of The Policy Management model. The *Policy Diagram* is the base holder that contains all the policies, rules, and events and the relationships among them. *Event*, *Policy*, and *Rule* are basic object of the system. Because policy and rule are composed of more detailed components, they must be models. As events provide the needed stimulus for policies and rules to be executed, we generalized Policy and Rule by an *Instruction* FCO object and created the

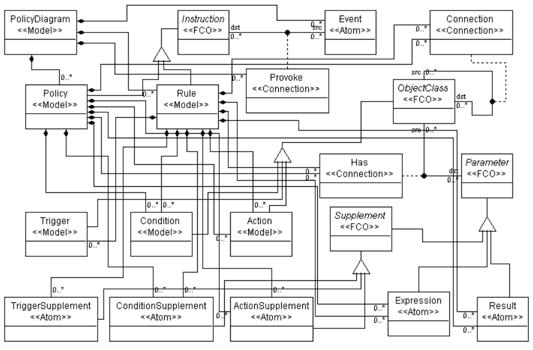


Fig. 3. The Policy Meta-model

Provoke connection between Event and Instruction. *Trigger*, *Condition*, and *Action* are the object class required to build the *Policy* and *Rules*. They use *Supplement*, Expression, and *Results* as their parameters. Therefore, we created *ObjectClass* object as generalization of *Trigger*, *Condition*, and *Action* and *Parameter* object as generalization of *Supplement*, *Expression*, and *Result*. The relationship between *ObjectClass* and *Parameter* is described as *Has* connection. The relationship among *ObjectClass* is described as *Connection connection.*

In this Meta-model, we created three aspects *EventAspect*, *PolicyAspect*, and *RuleAspect*. The *EventAspect* is used to present the Policy Management Model at Event interaction level. It contains *Policy*, *Rule*, *Event*, and the *Provoke* connection. The *PolicyAspect* is used to present the structure of a specific policy with *Condition*, *Action*, their *Parameters*, and relationship among them. *RuleAspect* is used to present the structure of a specific rule with *Trigger*, *Condition*, *Action*, their *Parameters*, and relationship among them.

3.3 The Policy Management Model

The Policy Management Meta-model is provided by xPMP to model the required component object for a policy-driven management system in the Object Model of UML.

The Policy Management Meta-model has seven major components

- *User Service*: provides the user interface to interact with users of the system. It sends User event to the Event Service.
- *Event Service*: listens to all events of the system and sends the signal events to the Rule manager if that event is registered by the Rule manager.
- *Rule Manager*: has two major roles: one is to register to Event Service to listen to all expected events of the system (including general system events and user generating events), the other role is to request Rule Engine for the Rule object based on the received events and then execute appropriate actions that the events trigger the rules.
- *Rule Engine*: receives requests for a rule from Rule

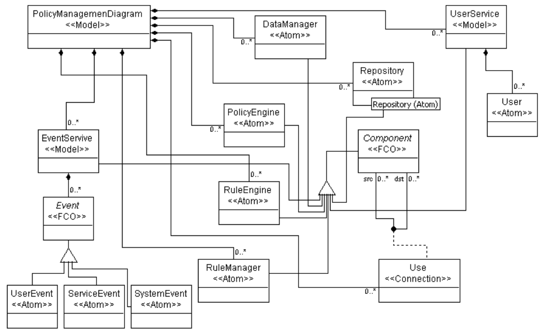


Fig. 4. The Policy Management Meta-model

Manager, retrieves the suitable rules from Database, merges rules with related policies to create a Rule object, and then provides them to Rule Manager.

- *Policy Engine*: retrieves the Policy from Database, converts it into a Policy object, and provide it to Rule Engine when requested.
- *Database Manager*: maintains the Policy data of the system including Rule maintenance, Policy maintenance.
- *Repository*: stores User data, Policies, and Rules.

Figure 4 shows the meta-model for the *Object Model* of Policy Management Systems. The *Policy Management Diagram* is the base holder that contains all the required components to support modeling policy-driven management systems. *UserService* and *EventService* have more details inside. Therefore, we use models for these components. *DataManager*, *RuleManager*, *RuleEngine*, *PolicyEngine*, and *Repository* are the required components and are modeled by atom objects. The *UserService* is composed of different *User* objects which are modeled by the *User* atom. Likewise, *Event* service is composed of different events which are modeled by *Event* atoms. The relationship between these components is described as *Use* connection.

For this Meta-model, we created one aspect called *ManagementSystem Aspect*. This aspect is used to model the policy-driven management system including the required components mentioned above and the relationship among them.

3.4 Methodology of Constructing Policy Management Model

Based on the System Requirements and Dynamic Model, we can create the Policy Management Model. This model includes Event interaction diagram, Policy structure, and Rule structure.

The Event interaction diagram can be constructed based on the System Requirement and Dynamic Model as follows: all the system nonfunctional requirements that have events can be described as an event-policy relationship; all events which appear in Dynamic Model can be described as an event-rule relationship.

The Policy structures are constructed as follows: system nonfunctional requirements are described as policies; functional requirements in Dynamic Model that applied for a group of system components or system objects are described as policies.

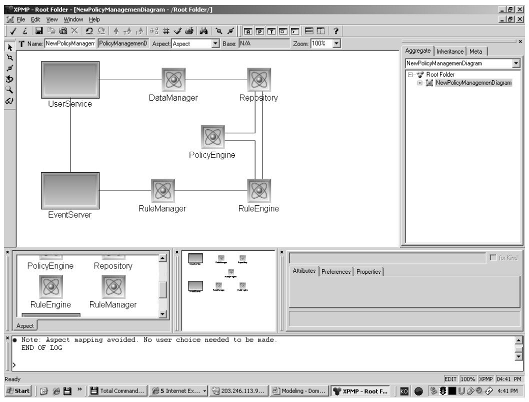
The Rule structures are constructed from all the functional requirements in Dynamic Model that applied for one specific system object.

4. The Policy-Driven Management Architecture for Ubiquitous Services Domain

In this section, we apply the meta-model of policy management system to develop a policy-driven management architecture in ubiquitous service domains - the Ubiquitous Service Center (USC) project - by using xPMP that we proposed in the previous section. The policy-driven management system dynamically adapts to the change of the environment by analyzing the context of the system in ubiquitous environment and generate the consequence actions to react to it. The Figure 5 shows the Policy Server Model design using xPMP as the modeling tool.

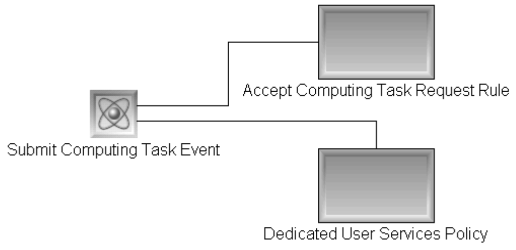
4.1 Modeling Event in EventServer Using xPMP

The EventServer is in charge of listening to system environment change, and service and user behaviors. When it recognizes a registered event, it will send the signal event to inform the situation to the Rule Manager.



```
Note: Aspect mapping avoided. No user choice needed to be made.
END OF LOG
```

Fig. 5. The Policy-Driven Management Architecture for Ubiquitous Services Domain


Accept Computing Task Request Rule
Submit Computing Task Event
Dedicated User Services Policy

```
- <model id="id-0065-00000001" kind="PolicyDiagram" childrelidcntr="0xf">
  <name>NewPolicyDiagram</name>
- <model id="id-0065-00000002" kind="Policy" role="Policy" childrelidcntr="0x13">
    <name>Dedicated User Services Policy</name>
  </model>
- <model id="id-0065-00000003" kind="Rule" role="Rule" childrelidcntr="0x18">
    <name>Accept Computing Task Request Rule</name>
  </model>
- <atom id="id-0066-00000001" kind="Event" role="Event" >
  <name>Submitting Computing Task Event</name>
- <connection id="id-0068-00000016" kind="Provoke" role="Provoke" >
    <name>Provoke</name>
    <connpoint role="dst" target="id-0065-00000003" />
    <connpoint role="src" target="id-0066-00000001" />
  </connection>
- <connection id="id-0068-00000017" kind="Provoke" role="Provoke" >
  <name>Provoke</name>
  <connpoint role="dst" target="id-0065-00000002" />
  <connpoint role="src" target="id-0066-00000001" />
  </connection>
  </model>
```

Fig. 6. An example of Event interaction diagram and its XML format file

The EventServer has two subcomponents:

- Context Manager: listens to the system, service, and user behaviors and generates the correlative events to the Event Listener.
- Event Listener: receives event registration from the Rule Manager or receives events generated by Context manager and sends them to Rule Manager.

Figure 6 shows an example of Event interaction diagram in USC project. Our USC provides computing services as we register the service center computer as a member in a Business Grid. The event "Submit Computing Task" to the USC affects "Accept Computing Task Request" rule and "Dedicated User Services" policy.

4.2. Modeling Rule Manager in xPMP

The Rule Manager is in charge of receiving signaling events from Event Service, requesting Rule Objects through Rule Engine and executing appropriate actions for the events. In Rule Manager, there are three subcomponents:

- Trigger: receives the events sent by Event Service.
- Condition: check the event with the rule condition
- Action: execute correlative action based on the rule condition.

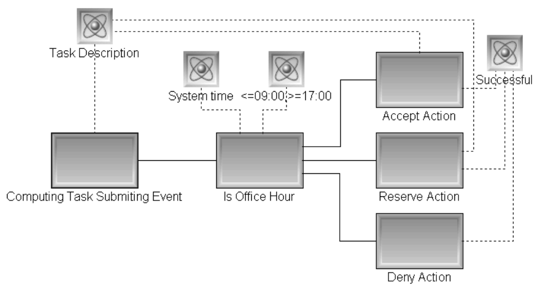
4.3. Modeling Rule in Rule Engine using xPMP

Rule Engine is in charge of requesting rules from Repository such as Database, merging with related policies, and providing them to Rule Manager. There are three subcomponents in Rule Engine:

- Rule Checker: checks the requested rule existing in database
- Rule Parser: converts Rule data to Rule Object
- Rule Extractor: extract appropriate rule structure and merge with related policies and send it to Rule Manager.

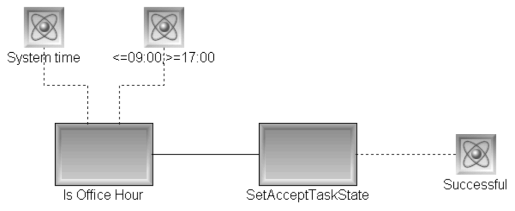
Figure 7. A rule example in USC modeled by xPMP and its XML format file.

Figure 7 shows an example of Rule modeled by xPMP. The rule is "If a job is submitted to computing center of USC, it is only accepted besides office hours 9am-5pm. If it is submitted during office hour, it can be denied or reserved for process. If it is submitted out of office hours, it will be accepted or reserved". The Target Class "Computing Task Submitting Event" has "Task Description" TriggerSupplement. The Condition class "Is Office Hour" requires ConditionSupplement System time, and Expression "<=09:00;>=17:00". The



```
<model id="id-0065-00000003" kind="Rule" role="Rule" childrelidcntr="0x18">
<name>Accept Computing Task Request Rule</name>
<model id="id-0065-00000006" kind="Trigger" role="Trigger" >
<name>Computing Task Submission Event</name>
  </model>
...
<atom id="id-0066-00000006" kind="ConditionSupplement"
  role="ConditionSupplement" >
<name>System time</name>
  </atom>
...
<atom id="id-0066-00000009" kind="Expression" role="Expression" >
<name><=09:00;>=17:00</name>
  </atom>
...
<connection id="id-0068-00000018" kind="Has" role="Has" >
<name>Has</name>
<connpoint role="src" target="id-0065-00000005" />
<connpoint role="dst" target="id-0066-00000006" />
  </connection>
...
  </model>
```

Fig. 7. A rule example in USC modeled by xPMP and its XML format file



```
<model id="id-0065-00000002" kind="Policy" role="Policy" >
<name>Dedicated User Services Policy</name>
<model id="id-0065-00000007" kind="Action" role="Action" >
<name>SetAcceptTaskState</name>
  </model>
<model id="id-0065-00000008" kind="Condition" role="Condition" >
<name>Is Office Hour</name>
  </model>
<atom id="id-0066-00000003" kind="ConditionSupplement" role="ConditionSupplement" >
<name>System time</name>
  </atom>
<atom id="id-0066-00000004" kind="Expression" role="Expression" >
<name><=09:00;>=17:00</name>
  </atom>
<atom id="id-0066-00000005" kind="Result" role="Result" >
<name>Successful</name>
  </atom>
..
<connection id="id-0068-00000012" kind="Has" role="Has" >
<name>Has</name>
<connpoint role="src" target="id-0065-00000007" />
<connpoint role="dst" target="id-0066-00000005" />
  </connection>
  </model>
```

Fig. 8. A policy example in USC modeled by xPMP and its XML format file

action classes provide one Result parameter "Successful". Two "Accept Action" and "Reserve Action" classes require "Task Description" Supplement.

4.4. Modeling Policy in Policy Engine using xPMP

Policy Engine is in charge of retrieving Policies from Database and converting to Policy Objects for sending it to Rule Engine when requested.

Figure 8. A policy example in USC modeled by xPMP and its XML format file.

Figure 8 shows an example of Rule modeled by xPMP. The policy is "During the official hours from 9am to 5pm, the Ubiquitous Service Center dedicates to serving the clients." In the figure, the Condition class "Is Office Hour" requires ConditionSupplement System time, and Expression "<=09:00;>=17:00". The action class SetAcceptTaskState provides one Result parameter "Successful".

4.5. Modeling DataManager in xPMP

The DataManager is in charge of manipulating all data of the system including Rule data, Policy data, and User data. There are following sub-components in DataManager:

- Rule Registrar: add, edit, update Rules
- Policy Registrar: add, edit, update Policies
- User manager: add, edit, update Users

The Database stores all information of system including User information, Policies, and Rules. In order to organize hierarchical structure of users, we use Roles and Users Architecture.

- Roles: describes the permission of accomplishing some actions over some resources.
- Users: a user can be assigned different roles.
- User groups: a user group can be assigned roles. A user belongs to one group can have all the roles of

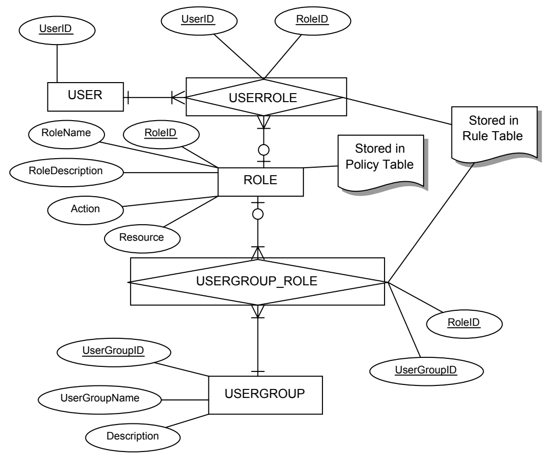


Fig. 9. Entity Relationship Diagram for User Schema. The Policy and Rule are used to manage the User-Role relationship in User system.

the group.

- We based on Policy system to realize this Roles and Users Architecture as follow:
- Roles: are described by Policies
- Assignment of Roles to users or user groups is expressed by Rules.

We use Policy and Rule to describe the User system. Therefore, we have an Entity relationship Diagram for User Schema as in figure 9.

Role Entity is described by Policy and has following attributes:

- RoleID: this is the Role Identifier number associated with each role and it is PolicyID which is automatically generated by system. (Primary key)
- RoleName: this is the Name of the Role.
- RoleDescription: This is the Description of the Role
- Action: This is the action that the user can execute if he is assigned this role.
- Resource: this is the resource over which the actions were executed.

The Role entity has relation with both the User entity and the UserGroup entity. We have two associate entities named UserRole and UserGroupRole express these relationships. Both these two entities are described by Rule.

5. Conclusion

As showed in the USC project, xPMP has provided a modeling profile that supports modeling of a dynamic and complex system in policy-driven management. The profile allows a detailed description of both Policy Management Model and the Policy Management Architecture of Object Model in the policy-driven management system. By using GME to create the meta-model, xPMP is designed in UML notation and is easily extend to further details of Policy management. We also propose in this paper the methodology to construct Policy Management Model from the basic models in UML design. Therefore, xPMP help to integrate easily policy-driven management in to any current software system design using UML. Finally, the working USC system and the models design of USC proved that xPMP is a reasonable tool supporting UML-based high-level modeling of policy-driven management applications.

Reference

1. J. Keeney, and V. Cahill “Chisel: A policy-driven, context-aware, dynamic adaptation framework.” Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks, 2003.
2. A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason IV, G. Nordstrom, J. Sprinkle, P. Volgyesi, “The Generic Modeling Environment.” Workshop on Intelligent Signal Processing, Budapest, Hungary, 2001.
3. J.V.D. Berg, and K. Coninx, “CUP 2.0: High-Level Modeling of Context-Sensitive Interactive Applications.” Model Driven Engineering Languages and Systems, 9th International Conference, 2006.
4. UML 2.0 Superstructure Specification. OMG documents ptc/03-08-02.
5. W. Itani, and A.I. Kayssi “SPECSA: a Scalable, Policy driven, Extensible, and Customizable Security Architecture for Wireless Enterprise Application.” Proceeding of the 2006 international conference on Communications and mobile computing, 2006.
6. P. Flegkas, P. Trimintzios, G. Pavlou, A. Liotta, “Design and Implementation of a Policy-Based Resource Management Architecture”, Proceedings of IEEE/IFIP Integrated Management Symposium (IM’2003), 2003.

7. L.V. Gool, T. Punter, M. Hamilton, R.V. Engelen, “Compositional MDA. In: Model Driven Engineering Languages and Systems”, 9th International Conference, MoDELS 2006, 2006.
8. A. Pleuss: “MML: A Language for Modeling Interactive Multimedia Applications.” Proceedings of Symposium on Multimedia, 2005.
9. GME 5 User’s Manual, http://www.isis.vanderbilt.edu/Projects/gme/GMEUMan.pdf.
10. M. Dekhil, V. Machiraju, K. Wurster, M. Griss, “Generalized Policy Model for Application and Service Management”, Policy Workshop 1999, Bristol, U.K, 1999.
11. R. Sobek, MOF Specifications 2.0 white paper.

Tran, Doan Thanh (thanhtd@kookmin.ac.kr)

2002 Hochiminh National University of Science, Telecommunication and Networking, B.S.
2006 Kookmin University, School of Business IT, M.S.
2006～Now Kookmin University, School of Business IT, Ph.D. Candidate

Areas of Interest : Grid computing, Ubiquitous Computing, Ubiquitous Sensor Network, Web Service Orchestration

최 은 미 (Eunmi Choi) (emchoi@kookmin.ac.kr)

1988 고려대학교 컴퓨터학과 학사
1991 Michigan State University, Computer Science, M.S.
1997 Michigan State University, Computer Science, Ph.D.
1998～2004 한동대학교 전산전자공학부 조교수
2004～현재 국민대학교 비즈니스IT학부 부교수

관심분야 : 분산시스템, 미들웨어, 유비쿼터스 컴퓨팅, 소프트웨어 메타 모델링, 대용량 검색 시스템, 그리드 컴퓨팅