Policy Enforcement Pattern

Yu Zhou, Qian Zhao, Mark Perry*

{yuzhou | qianzhao | markp} @csd.uwo.ca
Department of Computer Science
University of Western Ontario
London ON N6A 5B7 Canada

Abstract

Rapid development within organisations and their environments requires quick response and changes to law, business rules and other policy driven activities. Object-oriented systems, with their increasing dependence on component reuse, require an architecture that emphasises flexibility and run-time adaptability. In this paper, we abstract a high-level policy enforcement pattern that can satisfy the major requirements of such systems. The prerequisite of flexibility is addressed by using scripted policy rules. The need for adaptability is addressed by using a policy repository and plug-in modules of policy elements.

1. Intent

Enforce different kinds of policy in a uniform way. The policies and their enforcement processes can be changed easily to accommodate future changes.

2. Example

Consider a license management system of software component, as shown in Figure 1, in which we want to make sure that rights for identified clients are correctly attributed and authorised, and the usage of protected software component is properly recorded. To realise access control, and usage tracking and recording, it is necessary that there be a variety of policies that specify both the clients and their corresponding license types; that is, what they are able to do within the licensing scheme, and what they cannot do (for example make unauthorised distributions of a software component). Whether a license request is allowed or denied is based on policies that may be pre-defined by the developer of the licensing system, or a third party supplier, or may even be set up or changed by licensors at a later date, so long as this falls within the capabilities (i.e. legal restrictions) of that licensor. There may be occasions where this process is inherited by secondary licensors from primary licensors. In relation to the licensing of software, an area of high complexity and economic significance [Per2001], such a process is a commonly recurring pattern within the legal system and in business. The type of license and related licensing activities vary from client to client and over time; thus policies may need to change frequently. To maintain such licensing policies is difficult and resource intensive.

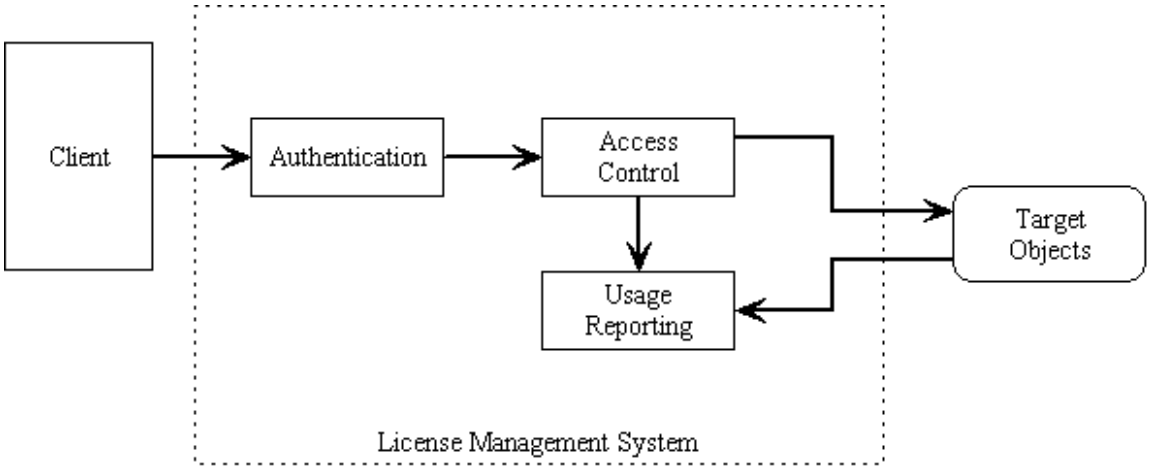

Figure 1: License Management System

A generalised and flexible architecture is required in which corresponding policies can be applied, so long as the policy elements are readily available in the rapidly changing policy environment.

3. Context

In many business environments, whether working with network management, quality of service (QoS) or within the mortgage and insurance sectors, we often have to face a policy pattern of the kind "When Who Can/Cannot Do What". As shown in Figure 2, Privileges specify "When", Subject specifies "Who", Conditions specify "Can/Cannot", and "Do What" is specified by Object and Actions.

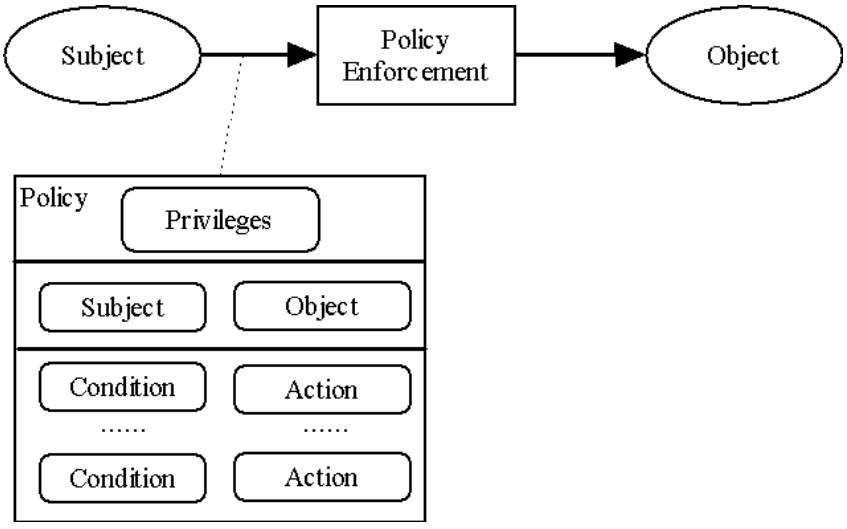

Figure 2: Policy Model

Business and management environments need flexibility and adaptability when implementing their policies. Although the target objects and their structures may be stable, the tasks, including the handling rules and procedures, are varied.

4. Problem

How to enforce policies no matter what kind of policies are required to be enforced? How to provide flexibility and adaptability to this policy enforcement, to accommodate future changes?

- System environments are in a constant state of flux, whether looked at from a high level of abstraction in business or from a lower level, such as in component management. In the face of such changeable circumstances, there have been some approaches proposed that fulfil the purpose of representing the dynamic business rules and management policies adaptively and effectively, including Command Dispatcher Pattern [Dup2001], Rule Object [Ars2000], Ponder [Dam2001], Internet Engineering Task Force (IETF) Policy Core Information Model [Moo2001].

- However, the Command Dispatcher pattern emphasises a plug-in mechanism and run-time interpretation of a sequence of command executions written in plain text, but does not consider organising those commands into policies like "if *condition* then *action* else *action*". The Rule Object Pattern, on the contrary, considers the language for policy/rule construction, but does not provide an easy way for building up policies/rules for people who are not familiar with any programming language. Ponder and PCIM take an implementation perspective and are mainly concerned with management of networks and QoS.

5. Forces

When the requirements and goals of network policies, management policies, and business policies are taken into consideration, the following forces need to be addressed in the context of policy enforcement:

1. *Integrating pre-built policy libraries:* Pre-built libraries are modules for a large number of policy elements that are used in a policy, such as privilege, condition and action. The libraries are extensible, which means that the policy elements inside can be independently developed and delivered even when the system is in runtime.

2. *Flexible configuration:* In a policy, there is the need for several collaborating elements to reach a specific purpose. Furthermore, a policy element may invoke other elements in order to complete its execution. In both situations, a number of options maybe available for a certain operation to be completed by selecting from a range of policy elements.

3. *User-defined policy rules:* The client should be able to implement and customise business rules easily.

4. *Central Policy Handling:* Policies should be stored in a repository and managed by a central instance.

5. *Efficiency:* Runtime implementation of customizable and dynamic systems often decreases the efficiency because runtime indirections, lookup mechanisms, and plain text parsing/interpretation place an overhead on the system.

6. Solution

The Policy Enforcement pattern (Figure 3) provides a uniform way to enforce a variety of business and management policies with flexibility and adaptability. With policy elements, policy processes, configuration policies and scripted policy rules, clients can customise their own policies and later register them in the repository that may be either centralised or distributed depending on the particular demands of the operation environment.

Architecture Overview

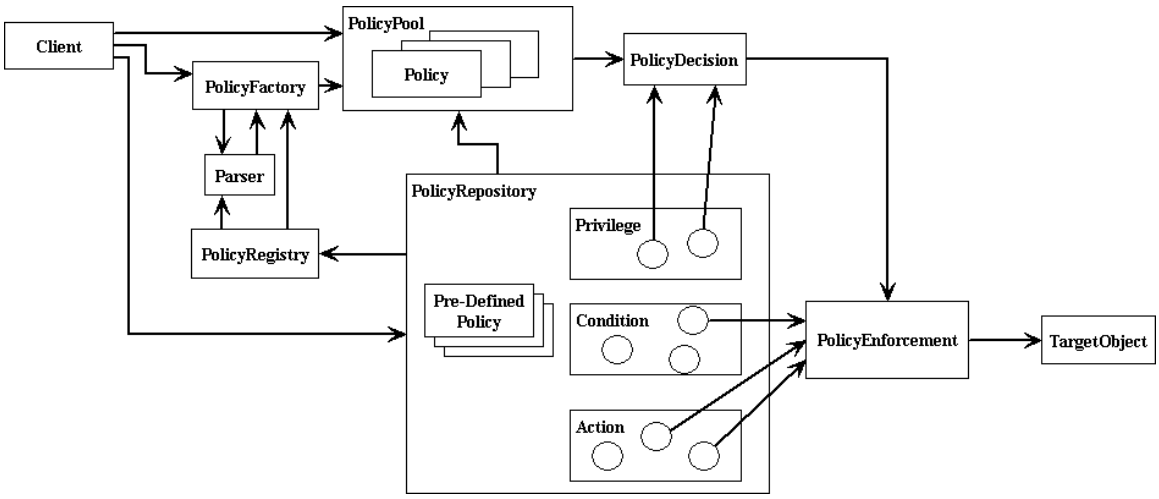

Figure 3: Policy Enforcement Pattern

As shown in Figure 3, clients request an operation on a target object. The Policy Factory creates the corresponding policy and related policy elements in the pool. Policy Decision decides whether or not the client's request is allowed. Policy Enforcement enforces the policy and executes the requested operation on the target object.

Participants

The following participants form the structure of the policy enforcement pattern:

- Common *clients* may request an execution on a targeted object. There may also be administrative clients to grant the privileges for clients, and to define and register the administrative policies. Some clients, who are developers, can define and register business rules as Policies, or they can develop and register policy modules into the Policy Repository.
- The *Policy Factory* creates runtime Policy instances and puts them into the Policy Pool.

- The *Policy Pool* provides efficient runtime storage for the policy instances. If a policy that a client request is already in the pool, it can be used directly without instantiating it again.
- A *Parser* parses the plain text format of policies, and then queries the policy element specifications from Policy Registry and in doing so helps create runtime Policy instances.
- The *Policy Registry* is a lightweight storage for specifications of policy element interface.
- *Pre-Defined Policies* are rules that can be applied to the clients' request. There are two kinds of policies, Configuration Policy and Policy Rule.
- *Policy Elements* are implemented modules used in policies, including *Privilege*, *Condition* and *Action*.
- A *Policy Decision* decides whether or not a certain request is allowed.
- The *Policy Enforcement* component evaluates the required policy conditions and invokes the corresponding policy actions. Depending on the type of the policy, the enforcement process may vary.
- The *Policy Repository* provides storage for the pre-defined policies, and policy elements. It supplies the references of Policy Element to Policy Decision and Policy Enforcement, and helps them both find the appropriate modules at runtime. The new functions and optimised elements can be integrated into it at a later date.
- The *Target Object* is the object that the client requests.

Structure

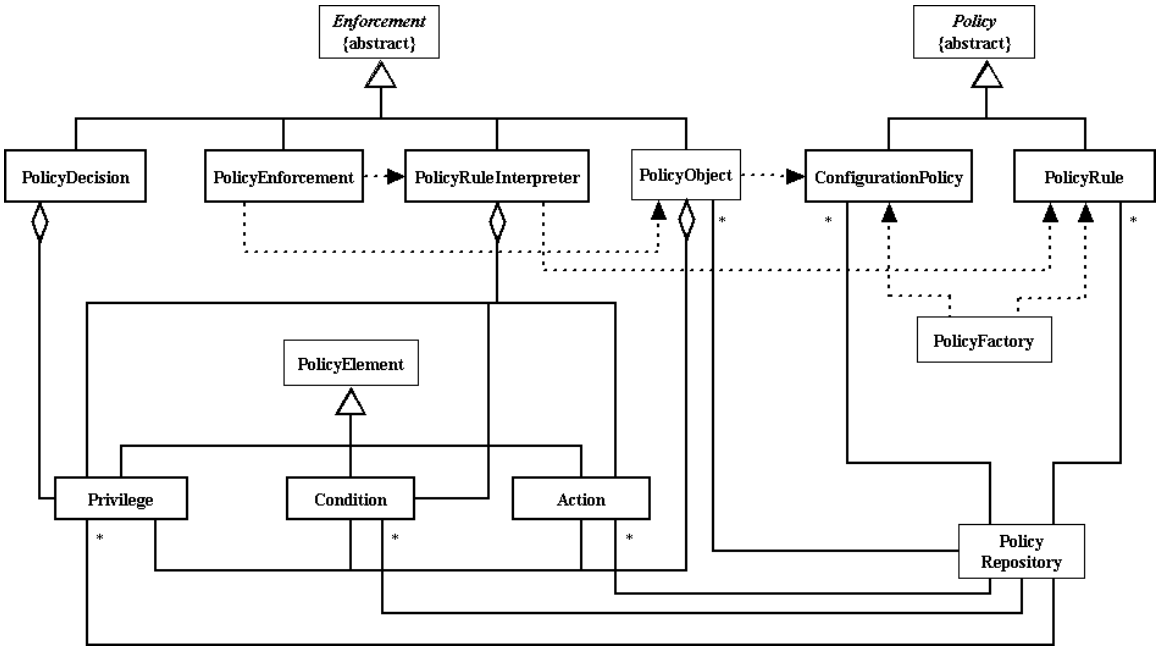

Figure 4: Class Diagram of Policy Enforcement Pattern

Figure 4 illustrates a class diagram of this Policy Enforcement pattern. There are two important interactions in this diagram: PolicyObject interacts with ConfigurationPolicy, and PolicyRuleInterpreter interacts with PolicyRule. The three kinds of PolicyElement, which are Privilege, Condition and Action, along with PolicyObject, ConfigurationPolicy and PolicyRule, can be defined, developed and delivered independently.

The PolicyDecision assesses the privileges based on the client, the target object, and corresponding policies. It queries the pre-built policy decision modules, and provides the security functions, such as authentication, access control, non-repudiation, and software license control.

PolicyEnforcement enforces the policy in two ways, through invoking either PolicyObject or PolicyRuleInterpreter, which are related to ConfigurationPolicy or PolicyRule respectively.

- PolicyObject is pre-defined policy process that interacts with ConfigurationPolicy. Some popular and well-known rules in certain target domain may be pre-defined as policy processes. These policy processes can be integrated into the system and be invoked when necessary.
- The PolicyRuleInterpreter interprets PolicyRule at runtime. Both kinds of PolicyEnforcement evaluate the required policy conditions and invoke the corresponding policy actions.

PolicyElement includes Privilege, Condition and Action:

- Privilege is module for PolicyDecision to help determine, according to its policy, whether or not the client's request on the required target object is legal.
- Condition is used to decide how a given policy will be carried out on a certain step.
- Actions are the operations that will be executed depending on different situations.

PolicyFactory creates runtime Policy instances and put them into the Policy Pool. There are two kinds of policies, ConfigurationPolicy and PolicyRule:

- *ConfigurationPolicy* contains the configuration information in a plain text format, which is used by both PolicyDecision and PolicyObject to choose proper policy elements.
- *PolicyRule* is an executable script file also containing the configuration information. In the execution of PolicyRule, the script is interpreted and the configuration information inside it provides the proper option of module choice.

The PolicyRepository provides storage for these pre-defined policies (PolicyRule and ConfigurationPolicy), policy processes (PolicyObject), and policy elements (Privilege, Condition, and Action). It supplies the

references of required policy element to `PolicyDecision` and `PolicyEnforcement`, and helps both of them find the appropriate modules at runtime.

Dynamics

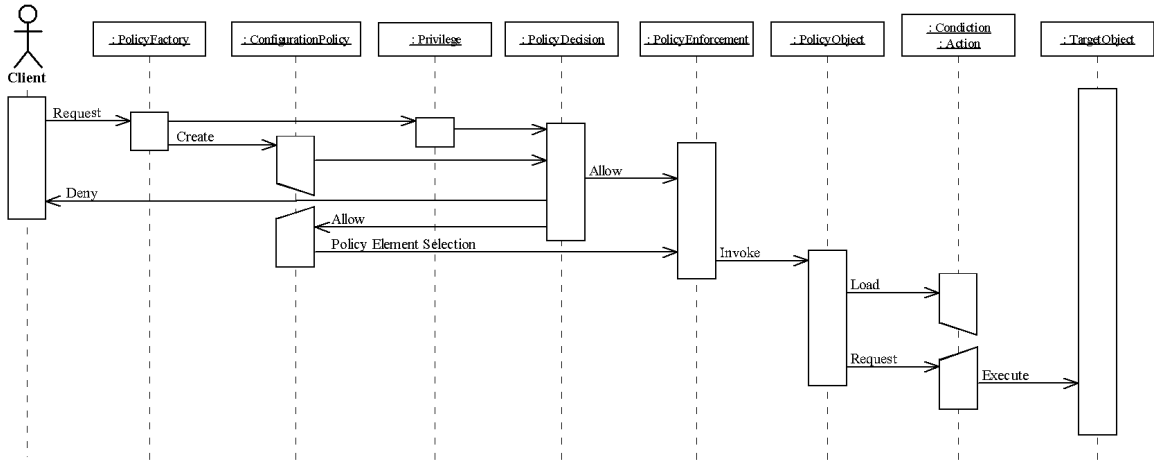

Figure 5: Collaboration for PolicyObject

Figure 5 shows the collaboration when dealing with `ConfigurationPolicy` and `PolicyObject`.

- The `PolicyFactory` checks if the policy for current client is running. If found, it will obtain the references of policy instance and use them directly.
- If not found, `PolicyFactory` will create the correct `ConfigurationPolicy` instances. In this way, the run-time costs, such as speed and storage space, are reduced.
- The `PolicyDecision` uses the `ConfigurationPolicy` instances as the configuration commands to obtain the runtime references of the pre-built Privilege modules. It can use these modules to check the authorisation and access control information, and then make the final decision.
- If the decision is positive, the `PolicyEnforcement` will invoke `PolicyObject`. Depending on the configuration policy, the `PolicyObject` gets the runtime references of the pre-built Condition modules and Action modules; then it evaluates the Conditions and invokes the Actions directly.

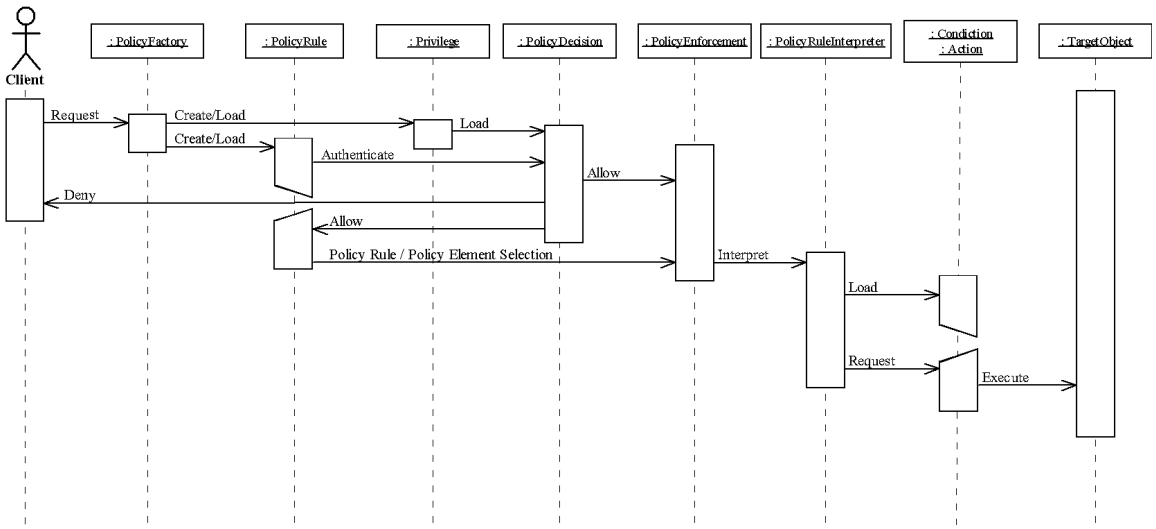

Figure 6: Collaboration for PolicyRule

When handling the scripted `PolicyRule`, as shown in Figure 6, the collaboration is slightly different from that for `ConfigurationPolicy`:

- The `PolicyFactory` will create the correct `PolicyRule` instances.
- The `PolicyDecision` uses the `PolicyRule` instances as the configuration commands to obtain the runtime references of the pre-built Privilege modules.
- If the result of Policy Decision is positive, Policy Enforcement will invoke `PolicyRuleInterpreter` that interprets and executes the policy rule. While interpreting, `PolicyRuleInterpreter` also evaluates the conditions and invokes the actions modules.

7. Example Resolved

Applying the Policy Enforcement pattern to a license management system, as shown in Figure 7, solves the problems we discussed in the example section.

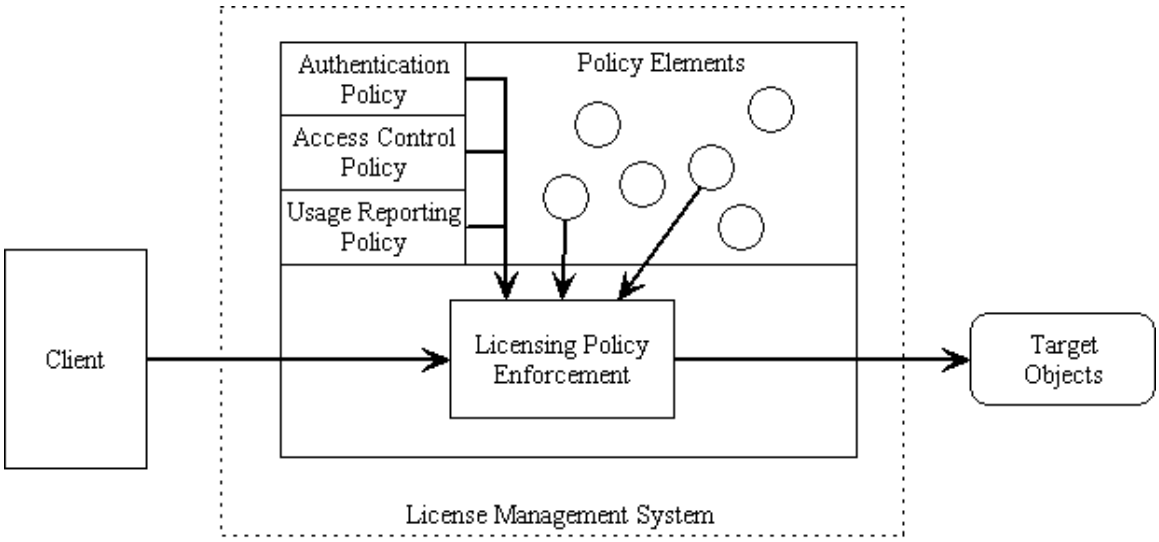

Figure 7: Example Resolved Using Policy Enforcement Pattern

If the license type of a client changes, we need only to assign another policy related to the new license type to this client and delete the old one. The Policy Decision and Policy Enforcement of the new policy will execute when the client sends a request.

If some more optimised policy elements are developed according to the published common interfaces, they can be integrated into the licensing system due to the plug-in mechanism and be used right away without affecting existing policies and other policy elements.

If a rule inside a policy does not fit into a given situation and needs to be changed, the scripted policy rule provides a convenient way to generate new policies utilising existing policy elements.

As noted above, the Policy Enforcement Pattern helps to easily maintain these varying licensing modules and policies and allows for future upgrades.

8. Consequences

Benefits

- *Flexibility:* The application of this pattern allows end users to define their own policies to meet specific situations.
- *Customizability:* This pattern facilitates customization of policy enforcement process by providing configuration information in policies.
- *Adaptability:* The policy elements can be independently developed and delivered. This ensures that new functions and more optimised elements can always be integrated into the system.

- *Universality:* This pattern can be applied in a wide variety of environments where there are clients in a policy-controlled situation.

Drawbacks

- *Performance:* This framework allows the plain text as a policy description, which needs to be parsed and interpreted at runtime. This will slow down the policy execution.

  The Policy pool and pre-built modules provide partial solutions to this performance problem. The Policy pool saves some time in policy instantiation. Pre-built modules provide time-optimised modules that can accelerate the execution time and improve performance dramatically.

- *Higher Complexity:* This pattern might increase the complexity of some target systems.

- *Understandability:* To implement the framework discussed in this study would not be trivial. It will require a relatively sound system design, which in turn will depend on detailed and thorough studies of the domain.

9. Variants

- *Access Control Policy Enforcement:* Policy Enforcement Pattern can be used to control the access to protected resources, such as in a membership-based digital library.

- *Reporting Policy Enforcement:* Policy Enforcement Pattern can be used to report the event that matches the pre-defined situation and to take corresponding actions, such as in a billing system.

- *Filter Policy Enforcement:* If we need to inspect a data flow, Policy Enforcement Pattern is able to check this flow and filter out the contents that are not allowed by policies.

- *Quality of Service (QoS) Policy Enforcement:* In QoS management, Policy Enforcement Pattern can be used to specify different types of service and to take corresponding actions.

- *Network Management Policy Enforcement:* When managing network, the rules of access control to network resource are specified as policy and are enforced using Policy Enforcement Pattern.

- *Business Policy Enforcement:* In a large enterprise, there can be many business policies involving all aspects of running that enterprise. To achieve

an efficient management, Policy Enforcement Pattern is needed in the management software.

10. Known Uses

Some examples of the use of the Policy Pattern are:

- Clients apply policies to network services, such as Quality of Service (QoS). Policies for QoS help in regulating distinct data packages, and to provide the specific services described in the service level agreements (SLAs) or service level specifications (SLSs).

- The Internet Engineering Task Force (IETF) Policy Core Information Model [Moo2001] provides a good example of model for representing policy information and classes that could use the pattern described here. It can be a QoS variant of this pattern.

- Clients apply policy to network management. Policies for network management help to control the access to network resources and to avoid illegal use.

- The Ponder [Dam2001] and Policy Framework for Management of Distributed Systems [Dam2002] address the implementation of managing network systems based on policies. They constitute Network Management Policy Enforcement.

- Clients apply policy to software license management. License policies help to simplify the management of different types of licenses and to detect the violation of these licenses. The Access Control Policy Enforcement, Reporting Policy Enforcement, and Filter Policy Enforcement are used in our software component licensing system.

- The motivation for the OASIS (Open Architecture for Securely Interworking Services) [Bac2001] derives from a study undertaken with a view to providing ubiquitous access to Electronic Health Records (EHRs) held within the National Health Service in England. It adopts the idea of the policy to realise the role based access control on the database, and employs the access control variant of this pattern to express different local policies using the same process.

- The core of Component-based Micro-Workflow [Man2002] is another known use of this pattern. The process component is similar to the pre-defined `Policy`. The execution component works as the Policy Enforcement. The synchronisation component is similar to the Policy Decision.

- Adaptive Object Model [Yod2001] extracts the business rules from the implementation code, and stores these rules separately. To achieve this, properties and behaviors (called ‘Rules’) are separated from the entity. AOM

suggests utilising the Strategy Pattern [GOF1995] or Rule Object Pattern [Ars2000] to implement Rules. This could be realised using the Policy Enforcement Pattern.

- Clients apply policies to business environments. Policies are defined according to business rules. They help to make the change from "manual" routines to "automated" systems. This helps save time and money, as well as introducing consistency. The Objectiva telephone billing system [And1998] uses the Type Object [Joh1998] that is the basis of AOM [Yod2001]. It could be implemented by using the variants of Reporting Policy Enforcement and Business Policy Enforcement.

11. Related Patterns

The Command Dispatcher Pattern [Dup2001] is implicitly part of Policy Enforcement Pattern described here. The configuration policy and the scripted policy rule are enforced as the Command Dispatcher Pattern.

The Policy Enforcement Pattern contains the `PolicyRuleInterpreter` and the `PolicyObject` , which are similar with the Rule Object [Ars2000].

The Abstract Factory Pattern [GOF1995] is used in this Policy Enforcement Pattern to create the two kinds of policies.

The Command Pattern [GOF1995] separates the request for a service from its execution. It is similar to the process of Policy Enforcement in the Policy Enforcement Pattern, which also manages clients' requests and schedules their execution.

The Strategy Pattern [GOF1995] defines a family of algorithms, encapsulates each, and makes them interchangeable. Strategy lets the algorithm vary independently from the client that uses it. It is similar to the use of flexible configuration in the Policy Enforcement Pattern, although only for simple situations and it does not offer scriptability.

Component Configurator Pattern [POSA2] allows an application to link and unlink its component implementations at run-time without having to modify, recompile, or statically relink the application. In the pattern described in this paper, the Policy Element, Enforcement and Policy are typical uses of Component Configurator Pattern.

Role Object Pattern [Bau1997] adapts an object to different clients' needs through transparently attached role objects, with each representing a role the object has to play in a particular client's context. The clients in this Policy Enforcement Pattern are also organised into different roles, such as common client, administrative client, developer and so on. Each role has its different policies, privileges, conditions and actions.

Acknowledgements

We thank Uwe Zdun for his great comments that significantly improved the presentation of this pattern.

References

[And1998] Francis Anderson and Ralph Johnson, “The Objectiva telephone billing system”, MetaData Pattern Mining Workshop, Urbana, Illinois, May 1998, http://www.joeyoder.com/Research/metadata/objectiva.pdf

[Ars2000] Ali Arsanjani, “Rule Object Pattern Language”, Proceedings of PLoP2000, Technical Report #wucs-00-29, Dept. of Computer Science, Washington University, October 2000, http://jerry.cs.uiuc.edu/~plop/plop2k/proceedings/Arsanjani/Arsanjani.pdf

[Bac2001] Jean Bacon, Michael Lloyd, and Ken Moody, "Translating Role-Based Access Control Policy within Context”, Proceedings of International Workshop in POLICY 2001, Bristol, UK, 2001, http://link.springer.de/link/service/series/0558/papers/1995/19950107.pdf

[Bau1997] Dirk Bäumer, Dirk Riehle, Wolf Siberski and Martina Wulf, "The Role Object Pattern", Proceedings of PLoP1997, Technical Report #wucs-97-34, Dept. of Computer Science, Washington University, 1997, http://jerry.cs.uiuc.edu/~plop/plop97/Proceedings/riehle.pdf

[Dam2001] N. Damianou, N. Dulay, E. Lupu and M. Sloman, “The Ponder Policy Specification Language”, Proceedings of the Policy Workshop 2001, HP Labs, Bristol, UK, Springer-Verlag, 29-31 January 2001, http://www.doc.ic.ac.uk/~mss/Papers/Ponder-Policy01V5.pdf

[Dam2002] N. Damianou, “A Policy Framework for Management of Distributed Systems”, PhD Thesis, Faculty of Engineering of the University of London, London, England, 2002, http://www-dse.doc.ic.ac.uk/Research/policies/ponder/thesis-ncd.pdf

[Dup2001] Benoit Dupire and Eduardo B Fernandez, “The Command Dispatcher Pattern”, Proceedings of Pattern Language of Pattern, 2001, http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/bdupireandebfernandez0/PLoP2001_bdupireandebfernandez0_1.pdf

[GOF1995] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, “Design Patterns – Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1995

[Joh1998] Ralph Johnson, Bobby Wolf, “The Type Object Pattern”, Pattern Languages of Program Design 3, Addisson Wesley, 1998, http://www.ksc.com/article3.htm

[Man2002] Dragos A. Manolescu**,** “An Extensible Workflow Architecture with Objects and Patterns”, TOOLS Eastern Europe 2001, Sofia, Bulgaria, 2002, http://micro-workflow.com/PDF/toolsee01.pdf

[Moo2001] B. Moore, E. Ellesson, J. Strassner and A. Westerinen, “Policy Core Information Model - Version 1 Specification (RFC 3060)” 2001, http://www.ietf.org/rfc/rfc3060.txt?number=3060

[Per2001] Mark Perry “Information Technology” Chapter 30 *Lexis/Nexis Butterworths Electronic Business Law* 2001

[POSA2] Douglas C. Schmidt, Michael Stal, Hans Rohnert and Frank Buschmann, “Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Ojbects”, John Wiley & Sons, 2000, http://www.cs.wustl.edu/~schmidt/POSA/

[Yod2001] Joseph W. Yoder, Federico Balaguer and Ralph Johnson, "Architecture and Design of Adaptive Object Models", Intriguing Technology Presentation at the 2001 Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '01), ACM SIGPLAN Notices, ACM Press, December 2001, http://www.adaptiveobjectmodel.com/OOPSLA2001/AOMIntriguingTechPaper.pdf