A Toolkit for Policy Enablement in Autonomic Computing

Dinesh C. Verma
IBM Research
dverma@us.ibm.com

Seraphin B. Calo
IBM Research
scalo@us.ibm.com

1. Introduction

Developers of policy enabled systems need a common set of basic functionality regardless of their areas of application. This is most conveniently provided in terms of a toolkit from which they can choose those components that are needed for their particular implementations. IBM T J Watson Research Center is developing such a policy toolkit in Java. The toolkit's goal is to accelerate the adoption of policy based technologies in different products and services. The Policy Toolkit (PTK) is written in Java and consists of a core module plus a set of modules that perform specific functions as shown in Figure 1.

A key challenge in building a policy toolkit is that policy functions need integration with existing management consoles. A policy language compatible with multiple management consoles is difficult to define. To address the absence of a common policy language, PTK has been designed for policies conforming to an information model, as opposed to a specific policy language. A policy information model specifies constraints on the structure of the policy rule, without specifying the syntax for expressing the policies. An extensible parser module provides the support needed for different types of policy languages.

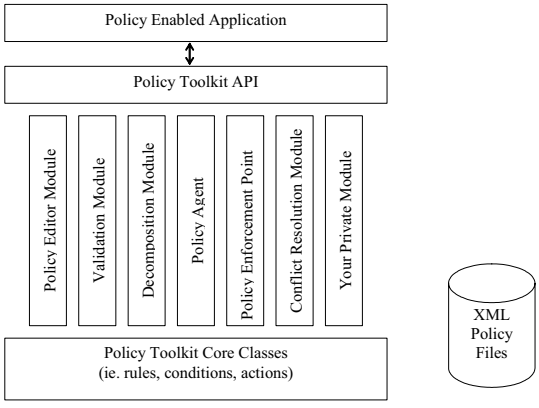

Figure 1: Policy toolkit design

Policies specified within a specific language are parsed into Java objects, and the Java objects representing policies are used by the other modules. To build a system based on the PTK, the developer needs to decide on the syntax of the policy language, select the modules for his system, and the pattern (if any) for the overall system. The developer can then customize the operation of specific modules by providing configuration information for each module (in XML files) and possibly extending the interfaces provided by each of the modules.

The modules that make up the PTK include: a ***Core module*** that provides a set of basic Java classes representing policies; a ***Parser module*** that provides a way to convert policies from the syntax of a given policy language to the core model; an ***Editor Module*** that provides a generic GUI for policy manipulation which can be customized to a specific policy syntax by using the common parser interface; a ***Policy agent module*** that provides a way to cache and receive policies from a repository; a ***Policy federator*** that provides a repository and distribution support for policies; a ***Validation module*** that provides methods for checking consistency among different policies; a ***Transformation module*** that provides the ability to change policies from one format to another; an ***Enforcement point*** that provides an efficient mechanism for finding policies matching an event and finding the resulting actions; an ***auto-update module*** that provides a way for systems to periodically update local copies of policies from a remote repository; and, a ***patterns module*** that provides a set of ready-to-use templates for building policy-enabled systems.

The design of many of these modules is relatively straight-forward. Some of the modules that are more complex are described in subsequent sections.

2. Validation and conflict resolution

For any system within the scope of more than one policy, ambiguous, anomalous, or merely undesirable situations may arise. The validation module provides a set of APIs that can be used to check for conflicts

Proceedings of the International Conference on Autonomic Computing (ICAC'04)
0-7695-2114-2/04 $20.00 © 2004 **IEEE**

IEEE COMPUTER SOCIETY

among different policies. These functions are used when a new policy is added to the repository or existing policies are modified. System specific knowledge needed for validation checks can be provided by means of an XML file to the system.

Generic validation functions supported in PTK include: ***Range checking*** that determines if the condition and action attributes of the rule are permissible; ***Consistency checking*** to verify that the actions specified in a set of policies do not conflict; ***Dominance checking*** to find redundant policy rules within the system; and, ***Runtime conflict resolution,*** to resolve conflicts during policy evaluation. Priority based schemes and meta-policies for validation are supported in PTK.

3. Transformation

Many policy-based systems provide for high-level business-oriented policies that need to be transformed into lower level technology-oriented policies for execution. The toolkit includes a set of generic transformation functions which can be customized in an extensible manner by a system developer. The transformations supported by PTK include static rule-based transformations, and adaptive case-based learning transformations.

Static rules allow expert users to provide meta-policies for transforming policies from one format to another. A common use is to map different classes of services (gold, silver, bronze) to a set of more detailed parameters. Other static rules are used to change actions into more efficient (albeit complex) formats.

Adaptive case based transformation within the toolkit provides a case-based reasoning technique for mapping measurable states (or goals) within the system into configuration settings. The case-database can be learnt by online monitoring, or be pre-populated. Various interpolation and clustering techniques are supported to enhance the effectiveness of the scheme. A developer needs only identify the configuration parameters and goal components within the system, and PTK module determines the right mechanism to map goals to configuration settings.

4. Design patterns

In order to ease the task of builders of policy based systems, the PTK provides patterns for building a variety of policy enabled systems. Each pattern consists of a set of interfaces that are linked together and provide the skeletal framework for developing a policy enabled system. The toolkit provides some standard implementations of these interfaces. Some patterns included within the toolkit are: **configuration pattern** used to build a system that lets an administrator define a set of high-level business policies, and have them transformed into a set of lower-level policies that are distributed to different devices within the network for enforcement; **auditor pattern** used to build a system that checks compliance with a specific set of policies; and the **planner pattern** can be used to build an application that uses policies to design or configure a new system (e.g., how many system resources should be provided to meet an incoming request in a Grid environment).

Patterns provide a way to present toolkit users with solutions that are already pre-built to a large extent. They can then be reused to build policy enabled systems rapidly.

5. Example Toolkit Usage

A sample usage of the policy toolkit has been to build a configuration checker for storage area networks (SAN). The system is based on the auditor pattern, and checks that the configuration of a SAN is consistent with the interconnection policies specified by the administrator. The auditor pattern consists of four interfaces connected together; a data-scanner interface that reads the data to be audited from the system; a data-analyzer interface used to break the data into policy terms; an enforcement point interface; and an action invoker interface. The SAN system provided extensions to the different interfaces, providing a data-scanner implementation that reads a SAN configuration database is invoked to check policies, selects from a menu of available data analyzers, and provides customized action invokers to generate policy violation reports.

6. Conclusions

The Policy Toolkit consists of a set of components for supporting the policy enablement of computer applications. As well as providing the necessary basic functionality, it also includes advanced capabilities like policy validation, transformation, and conflict resolution. It also provides patterns for building a variety of policy enabled systems. A number of engagements have been undertaken in order to assess the usefulness of the components provided, and determining what additional capabilities might be needed. We have found that the Policy Toolkit has made it easier and more convenient for software developers to incorporate policy-based technologies into their applications.