Approach To Policy Execution in Autonomic Manager Toolkit

Hoi Chan, Dinesh Verma, Alla Segal, Bill Arnold, James Giles, Dashi Agrawal, David Olshefski

IBM T.J. Watson Research Center

19 Skyline Drive

Hawthrone, NY 10532

(914)-784-7741

hychan@us.ibm.com

ABSTRACT

IBM Research is developing an Autonomic Manager Toolkit [1] (AMTK) which supplies components and infrastructure to simplify the incorporation of autonomic functions [2] into applications. In this report, we describe the Autonomic Computing (AC) requirements for policy tooling, policy management, the execution and data models for policy driven applications, and how the design and implementation of the AMTK policy evaluation system provides a policy execution infrastructure with the flexibility and extensibility to meet the needs that autonomic managers have for policy tooling.

General Terms

Algorithms, management, design

Keywords

Autonomic, toolkit, policy, rule engines

1. INTRODUCTION

The main purpose of using policies is to guide the behavior of entities in applications by extracting and externalizing business logic from applications into sets of rules. In general, these rules are human readable and in some cases, they are expressed as XML or other programming languages. Modification of policies does not affect the underlying data and applications, resulting in a higher level of maintainability, variability and manageability. Most current policy systems are domain and environment specific, and therefore require domain and environment specific information to execute the policies. There is a need for an execution system which is flexible and usable across a variety of domains. AMTK provides such an execution system by utilizing component technologies and separates the process of executing a policy into various sub-processes or components. Each sub-process or component can be developed independently, configured independently, and reused.

2. OVERVIEW OF AMTK

AMTK is structured with interfaces, not inheritance and is highly pluggable. Interfaces and concrete implementations are provided for each major component of the toolkit. It also contains a skeleton with a complete implementation of a null autonomic manager which can be extended through inheritance. The AMTK provides the following classes of tools for component self management:

Knowledge base – provide uniform access to knowledge

Monitoring – provides active tracking of current state via sensors

Analysis – prediction algorithms, modeling

Planning, Execution **–** policy evaluation and effectors to change external state

3. POLICY IN AUTONOMIC COMPUTING

Monitor: Policies could be defined controlling the specific type of monitoring that needs to be performed under various conditions.

Analyze: Policies dictate the type of reports generated, and determine whether the current state of the system is in violation (or potential violation) of any specified policies in the system.

Plan**:** Policies dictate the type of actions that will need to be taken, including actions that can modify the configuration of the system. A policy may call for allocation of new resources, initiation of new tasks, or termination of existing tasks.

Execute: Policies may define specific details for steps to take in order to carry out any actions identified in the planning phase.

The policy execution system used in autonomic computing should be sufficiently flexible to be able to evaluate a wide range of different policies as described above and extensible to meet future requirements. AMTK provides such an infrastructure.

4. POLICY MODEL

The AMTK uses a draft OGSA policy standard as its default policy model. This draft proposed a close mapping of Policy Core Information Model (PCIM) [3] to a policy XML schema. An Autonomic policy effort is defining a set of common policy definitions to be used in the context of Autonomic Computing within IBM. It is a refinement of the PCIM information model, where each policy rule is viewed as being composed from four types of constructs: pre-condition, measurable intent, scope, and business value. The autonomic policy workgroup will be defining some common types of pre-conditions, measurable intent, scope, and business value expressions. AMTK will track this emerging standard and will support it as it emerges.

5. POLICY EXECUTION

Policies are generally not represented as executable code, and must be converted to an executable representation such as a rule language. The translation process involves understanding of the semantics of the policy, the mapping of logic contained in the policy to the constructs of the underlying code, and the mapping of terms used in policy to the underlying data such as database

entries, application objects or resource models which describe the devices to which the policy is applied. A policy executed in a different environment may require different mappings; while its logic expressed as policy may be the same, the data, class objects or device resource model may be different. Generic rule engines and production rule systems such as OPS5, CLIPS, flex [4,5], Java Script engines, backward and/or forward chaining inference rule engines[9], fuzzy rule engines, and SQL engines can be used as the underlying execution system for policy, depending on the data requirements of the policy and the level of reasoning required. The AMTK provides interfaces for, and default implementations of, a rule based policy execution mechanism, that support the direct invocation of class methods as sensors and effectors through the use of a pluggable rule engine. In some policy applications, policy evaluation consists simply of finding a set of policies that match some conditions and taking the actions upon them. In these cases, developers may want to use a more efficient evaluation mechanism. The hypercube evaluation system[6], which will be part of the AMTK policy evaluation infrastructure provides such an evaluator, and may be appropriate for developers who do not need inference capabilities provided by rule-engines. In a hypercube system, each policy rule defines an n-dimensional box in n-dimensional space, where n is the total number of variables used in the policy definition. The hypercube policy evaluator performs an efficient matching of policies by determining the region of the space which corresponds to the conditions triggering policy evaluations. The different regions in the hyper-space are mapped to a search tree, and efficient tree-traversal techniques are used to find out the set of hypercube spaces, and the policies that match them.

Policy execution begins by input of a policy document (which may have been retrieved from a Policy Repository). The policy is then translated to an executable format by using an appropriate translator from the Translator Repository, using a a set of resource mapping definitions expressed in XML. The output of the translator is an executable ruleset for the selected rule engine or execution mechanism. The ruleset can then be published to the execution engine, or simply stored in the Executable Ruleset Repository for subsequent use. In addition, executable rulesets can be pre-processed or cached, and fetched at runtime to achieve better performance and variability.

6. AUONOMIC ELEMENT AND AMTK POLICY STRUCTURE

Figure 1. shows a basic conceptual view of how policy based functions are incorporated into Autonomic Elements (AE). AMTK provides authoring and policy services at design time to construct policies and a run time infrastructure which allows pluggable components to be dynamically loaded into the. An AE can switch to another ruleset and/or mapping and/or execution mechanism depending on the environment or/and the logic of the policy. This architecture provides AE with a much higher level of adaptive behaviors, and allows AEs to be well situated in a heterogeneous environment, and maximize the variability and flexibility provided by the use of policies[7].

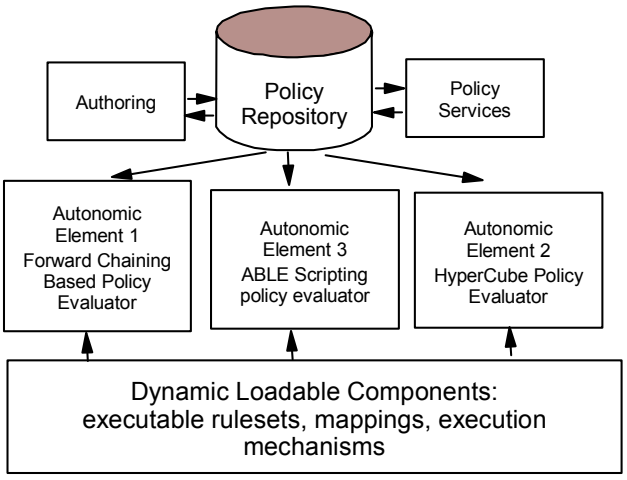

Figure 1

7. CONCLUSION

In this paper, we have described the approaches for policy execution in AMTK.. The AMTK policy execution infrastructure allows components to be selected based on context, environment, data and rule engine requirement. It covers a variety of rules based applications, from simple conditional scripting language to complex reasoning with specialized reasoning rule engines. AMTK is available for free download (30 days IBM Alphaworks license) 3Q, 2003.

***** The full paper from which this abstract is derived is available from the authors .

8. REFERENCES

[1] Autonomic Manager Toolkit (available 3Q, 03 for free download : http://www.alphaworks.ibm.com)

[2] Jeff O. Kephart, David M. Chess, "The Vision of Autonomic Computing", Computer Journal, IEEE Computer Society, January 2003 issue.

[3] Policy Core Information Model (PCIM), Version 1.0 ,RFC 3060, Feb,2001 (http://www.ietf.org/rfc.rfc.3060.txt)

[4] Rete: "A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem", Charles L. Forgy, Artificial Intelligence (1982), 17-37

[5] Giarrantano and Riley, "Expert Systems: Principles and Programming", Second Edition, P6WS Publishing (Boston, 1993)

[6] Policy Toolkit Description Version 1. Available from Dinesh Verma (dverma@us.ibm.com)

[7] Murthy Devarakonda, Alla Segal, and David Chess, "A Toolkit-Based Approach to Policy-Managed Storage", in press. To be presented at Policy 2003 (Intl Workshop on Policies for Distributed Systems and Networks)