Policy Framework for Autonomic Data Management

Manish Bhide, Ajay Gupta, Mukul Joshi, Mukesh Mohania, Shree Raman
IBM India Research lab,
Block-1 IIT Delhi, Hauz Khas, New Delhi - 110016
{abmanish,agupta,mukuljos,mkmukesh,shreeraman}@in.ibm.com

Abstract

The popularity of e-Business has lead to an exponential and unstructured growth in the applications space coupled with an increase in the database size. This has led to an increase in the complexity of the database management task. Moreover, organizations are increasingly concerned about the privacy of data. Thus, managing such large ever growing and privacy-preserving database is complex and time-consuming task. In this paper we describe a policy-based framework for autonomic database management using Business Objects. Our system automatically manages data based on events.

1. Introduction

In most of the organizations, the size of the distributed information repositories is increasing at the rate of 40% every year due to an increase in Internet based transactions. Managing such large distributed information repositories is a complex and time-consuming task, which is solved by hiring skilled database administrators. Adding to this complexity is the fact that these distributed information repositories are administered by multiple administrators who are responsible for maintaining different applications that are sharing the same database. Such administration of the system and database can lead to conflicting actions unless the actions are bound by constraints/ policies, which can avoid conflicts [1].

In this paper, we discuss a policy-based framework for autonomic database management that provides an efficient and easy-to-use system, which enables the policy makers to directly define and deploy the data management and operational policies of their organizations, all by themselves. The functioning of the Autonomic Data Management System (ADMS) at a high level is shown in Figure 1. Here, the user describes the policies at business object level using a GUI and then these policies are converted into XML using metadata that has information about mapping between business objects and underlying data entities. The policy XML is stored in a policy database. When an event is detected (either database, temporal or external), the relevant policies are fired and then executed on the application databases.

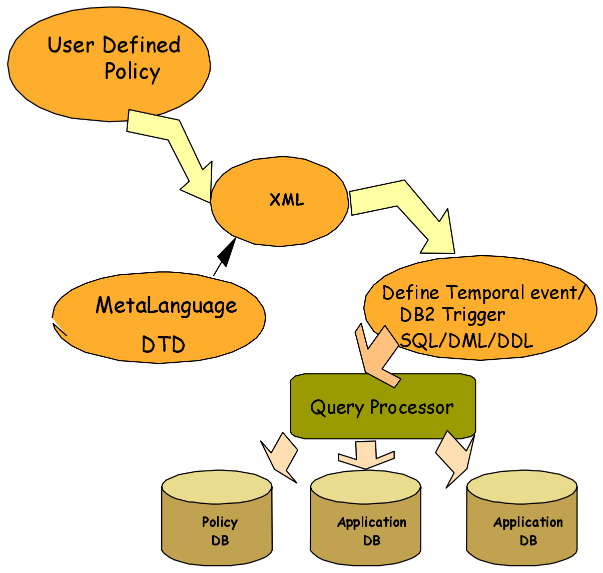

Figure 1: High level functioning of ADMS

2. Business Objects

The main impediment in the acceptance of any policy based system is its usability. The policy definition process has to be very intuitive for any policy based system to get accepted. Business objects are very good aids to visualize the real-world data. They are much easier to work with than the tables of a relational database (where enterprise data is stored). Hence there is a need for a mapping between the attributes and methods of the business object to the relational database domain. Our system has a tool that allows a domain expert to do this mapping. The domain expert has to map the attributes of the business object, which can be simple attributes, composite attributes or aggregate attributes, to the relational domain. The domain expert also has to map the methods of the business objects which can be one of event methods, condition methods or action methods, to a set of operations on the relational tables. Once the business objects have been abstracted out from a domain, the policy maker can define the policies solely in terms of the business objects. The policy maker does not need to have any knowledge about database administration practices; nor does he require understanding of the complex database schema underlying the applications.

3. ADMS Architecture

The policy maker defines the policies at the business object level using the Graphical User Interface (GUI) provided by the system. The policies defined using the GUI are passed on to the *Policy Translator*. This module converts the business object level policy into an *XML* based language, which is understood by the *Execution Engine*. In this XML language, the policies are explicitly defined in terms of the relational database tables and events, i.e. it is no more at a business object level.

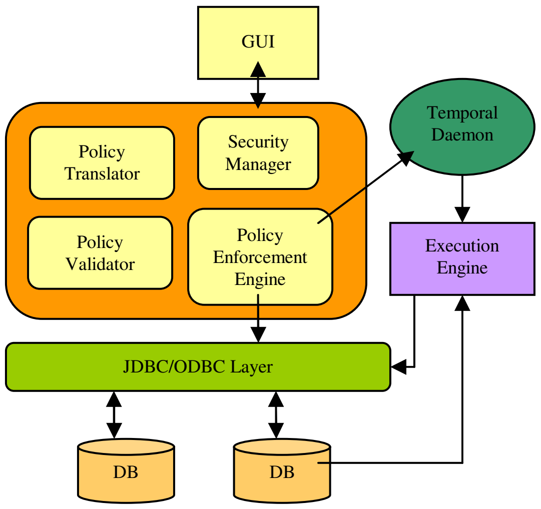

Figure 2: Architecture of ADMS

Since one business object may be constructed from many data entities, the policy translator may convert a policy defined at business objects into multiple policies defined at data entities level and creates the dependency between these policies that shows their execution order. The policy is then passed on to the *Policy Validator* module, which checks for errors and inconsistencies in the policy, and for conflicts with existing policies. Depending on the type of the policy, the Policy Enforcement Engine, either defines triggers on the database or defines alarms using the temporal daemon [2, 3]. When the trigger (either database or temporal alarm) fires, a notification is sent to the Execution Engine which does event correlation and fires the correct policy. The action is executed by the Execution engine by issuing SQL/DML/DDL statements on the underlying application databases.

Our system supports features like deferred condition/action evaluation, whereby the evaluation of the condition/action can be deferred by some time after the event/condition is checked. Some of the sample policies that our system can support are given below:

1. On every Sunday at 7:00 AM, archive all customer orders, which have been processed fully and for which payment has been received.
2. If a supplier stops supplying a part, then delete all those transaction items involving that part, if the transaction cost is less than $50.

4. Conclusion

One of the main concerns of any industry is to manage system behavior and information repositories automatically, especially when the size of information repository is increasing and there is a shortage of skilled database administrators. In this paper we have presented a policy framework based on active functionality that can help to externalize rules for autonomic data administration. Using our framework the DBA can define policies using business objects and does not have to worry about the intricacies of data management like definition of triggers, execution of periodic actions etc.

5. References

1. H.V. Jagadish, A.O. Mendelzon, and I.S. Mumick, `Managing Conflicts between Rules', 15th ACM International Conference on Principles of Database Systems, Montreal, 1996, pp 192-201.
2. A. Gupta, M. Bhide, M. Mohania, 'Towards Bringing Database Management Task in the Realm of IT non-Experts', In Proceedings of 19th International Conference on Data Engineering, India, March 2003.
3. A. Gupta, M. Bhide, S. Pandey, and M. Mohania, 'Event Based Access Control: A Demonstration', In Proceedings of 19th International Conference on Data Engineering, Bangalore, India, March 2003.