Concept of a Script MIB based Policy Management System

Frank Strauß

Computer Science Department
Technical University Braunschweig
Bültenweg 74/75
38106 Braunschweig, Germany
strauss@ibr.cs.tu-bs.de

May 2001

Abstract

Tasks 6.1 – 6.3 of the joint Jasmin Project between the Technical University of Braunschweig and NEC C&C Research Laboratories scheduled for the time from January to August 2001 are concerned with the design and implementation of a policy management system based on the Jasmin Script MIB implementation.

This report documents a number of requirements and the general concepts behind this Jasmin based approach. The architecture of a Java class package supporting the programming of policy scripts and the application to the policy based management of DiffServ routers are described.

1 Introduction

Policy based management is not a new area of research and engineering. It has been addressed in the past from the research point of view by several research groups, workshops, and conferences [1, 2] and from the vendors' point of view by some specific engineering and implementation efforts, e.g. [3].
The goals of this project are somewhere in between: Based on some research knowledge from the past few years and based on the knowledge from the work that has been done so far in the IETF Policy Framework (POLICY), Configuration Management with SNMP (SNMPCONF), and Resource Allocation Protocol (RAP) working

groups, a policy management system is going to be outlined and implemented in a prototype fashion. Its basis will be the IETF Script MIB [4] infrastructure that has been implemented in previous phases of this project [5]. The Script MIB functionality will be used to transfer and control the execution of policy 'scripts'. The execution of a policy script is realized by a new runtime engine for the Jasmin Script MIB agent [6]. This project focuses on the design and implementation of a policy definition language (or a policy definition extension to an existing language) and its implementation as a Jasmin runtime engine.
As a first step in this project, this document gives an outline of the requirements and our proposed solution of a Java based extension to the Jasmin architecture that allows to specify and execute policies in terms of Java programs. A few further ideas are presented that could lead to a more efficient policy description language.

1.1 Terminology

Various research and engineering approaches related to policy based management show some variance in terminology. To assure a common unambiguous terminology throughout this project we manifest the following terms.

Policy: A set of *rules*, usually concerned with a common *domain*, that make up a general network management goal.

Domain: An area of network management, targeted by a *policy*. Examples are IP filter, IPsec, or DiffServ configuration. The domain targeted within this project is the configuration of DiffServ routers, although the general architecture will not be limited to any domain.

Element: The actual subject on which a *policy rule* operates. Examples are IP filters or DiffServ classifiers or meters. In an object oriented fashion, elements are modeled as classes. Methods of these element classes can be used by the *conditions* and *actions* of *rules* to retrieve element instance information, e.g. a value of a single attribute, and to initiate operations on element instances, in a protocol independent way. Only the elements' interfaces to the managed devices are protocol dependent.

Role: An attribute (usually a string) administratively assigned to *elements* in order to group instances of the same class. This allows *rules* to operate on a subset of elements that are assigned to a specific role.

Rule: A construct of the form

`on` <*event(s)*> [`priority` <*priority*>] [`if` <*condition*>] `do` <*action(s)*>

A rule is being evaluated at discrete points in time given by the *event(s)*. At each occurrence of an *event*, all rules that are triggered by that *event* are processed in the order given by the *priority* values. If no priority is given, there is no preferred order for this rule.
First, the *condition*, if present, is evaluated. If the result is true or if the *condition* is absent, the *action* is executed. Attributes of the initiating *event* can be referenced by the *condition* and *action*. The *condition* can operate on a single *element* or on a set of *elements*. If it operates on a set of elements, the *action(s)* are executed for each matching element(s). Their instance identifying attribute(s) can be referenced within the *action(s)*.

Event: A formally defined event that can be bound to *rules* in order to specify the time when the evaluation of those rules is triggered. Types of events are time events (periodic, calendar based), external events (e.g. based on received SNMP notifications), and poll events (e.g. when the status of a regularly polled attribute changes).

Priority: An integer value that can be used to specify the order of the evaluation of multiple *condition*/*action* pairs triggered by a common *event*.

Condition: An expression that is evaluated to a boolean value each time the *rule* is triggered. If the *condition* contains variables that specify *elements*, it is evaluated for each combination of those elements.

Action: An operation that is executed each time the *condition* has been evaluated to true. Attributes of the *event* that triggered the execution and variables specifying the *elements* that lead to the matching *condition* can be referred within the action. Special functions that could be useful in actions are (i) raising new *events* and (ii) skipping all other actions that would otherwise be executed subsequently upon the same *event* that triggered this action.

1.2 Domain: DiffServ

The targeted policy domain of this project is the configuration of DiffServ [7] routers. However, the general architecture will not be limited to DiffServ, but most of the examples and the elements that will be modeled and implemented throughout the next months will be concerned with the configuration of DiffServ TCB elements like classifiers, meters, markers, queues, and schedulers.
Interfaces for DiffServ configuration are being developed and implemented by different groups at the time. The IETF DIFFSERV working group [8] is developing an SMI MIB [9] and an SPPI PIB [10] for managing DiffServ routers via SNMP and COPS-PR. Two free DiffServ implementations that are valuable for evaluating the implementations being developed during this project are available for the Linux kernel.

The first one has been integrated with the official Linux-2.4 kernel release. Another one has been implemented in a joint project between NEC C&C Network Product Development Laboratories and the University of Bern. For the first one, two independent DiffServ MIB implementations are underway: one as a master thesis project by Remco van de Meent at the University of Twente[1] and another one at the Pohang University of Science and Technology [11]. At NEC C&C Research Laboratories an API for their DiffServ implementation is being developed.
At least one of these DiffServ implementations and DiffServ management interfaces have to be used in this project in a way that DiffServ specific elements are designed and implemented. The 'upper' protocol independent interface of these elements will be used by DiffServ policies. The 'lower' interface will use either the DiffServ MIB SNMP interface or the NEC API. It is not expected that an implementation of the DiffServ PIB will be available in time. Ideally, multiple implementations of the designed DiffServ specific elements will be available sometime to check their exchangeability with a common policy.

2 Concepts

This section presents a number of requirements for the policy management system that is going to be developed, followed by the concept of an architecture based on the Jasmin implementation of the IETF Script MIB.

2.1 Requirements

This section describes a number of requirements that we claim to achieve for a policy management system, although they might be common to all approaches in policy based management.

- A policy rule condition must allow read access to the attributes of zero, one or multiple elements. This has to be done in a way, so that the according action can unambiguously reference those elements that matched the condition. Similarly, the attributes of the event that triggered the rule must be accessible. This means, we need a concept of free variables in the event and condition definitions that are bound by the runtime system to element instances when passed to the condition and action. These variables can be declared implicitly in the events and conditions or explicitly.

- There must be a construct to specify the value space in which the free variables of conditions are evaluated. This may span all instances of elements within a certain table or even all instances of a class among a number of managed agents.

[1] Remco van de Meent <remco@vandemeent.net>

- A class that models a certain element must support a number of accessor methods that allow a policy author to retrieve and manipulate an element in a comfortable way. E.g., counter retrieval functions should implicitly support rate computations, and SNMP RowStatus handling should be hidden by methods to construct and destruct element instances. This is not a requirement for the policy engine architecture itself, but for the design of domain specific elements.

- So far, at least three types of time events are required: Periodic events that trigger continuously at a given period. Calendar events that trigger periodically at points in time specified by calendar-type attributes (month, day, weekday, hour, minute), and one-shot events that trigger exactly once at a point in time specified by calendar-type attributes. These three types are motivated by the Schedule MIB [12].

- Another type of event is based on the reception of external notifications like SNMP traps/informs or COPS-PR state reports. These notifications should be mappable to domain specific events. Details of the initiating notifications should be accessible through accessor methods of the events.

- The policy runtime engine must support a mechanism to report errors and optional tracing/debugging information so that users can monitor the policy engine and the authors of policies can test and debug their policy code.

- The access to elements in conditions and actions may fail. The policy runtime engine must be able to handle these situations in a way that accordingly written policy code can catch the error conditions and bring the affected element to a determined state.

- It must be possible to store and execute multiple policies independently. Their code must not share any name space. However, avoiding side effects by multiple policies or policy rules acting on common elements is the responsibility of the policy author(s).

- A security mechanism is required to differentiate which users have access to which operations on which policies. This is regarded as a very sensible aspect. Building on an existing security mechanism could be helpful.

- Ideally, the policy programming interfaces of domain specific elements are independent of the underlying management interfaces. This means a policy acting on an element does not have to care about the question whether an underlying device is managed via SNMP, COPS-PR, a command line interface or an API.

- In theory, policies declare behaviors of elements dependent on conditions. However, a programmatic policy system has to work in a deterministic sequential

fashion; especially complicated actions must contain a bunch of code instead of just the goal that is about to be reached. The notation of policies should retain the declarative fashion of policies as much as possible.

- It should be possible to avoid redundancy in a way that policies or policy groups sharing rules, and rules sharing conditions or actions can be built by referring common code instead of copying code fragments. This allows to increase reusability and to avoid some errors.

- A communication mechanism between active policies based on shared memory or messages would also help to reduce redundancy. For example, one policy can determine a responsible person to which a number of other policies send reports in case of errors.

- It could be useful to pass arguments to policies when they are activated. Although all parameters of a policy behavior should be defined within the policy code, arguments could be useful to turn debugging on and off or to trace a policy in read-only mode.

2.2 A Script MIB Based Approach

In contrast to other research and development approaches, this project will build on the IETF Script MIB architecture as an infrastructure for transferring policies and managing their enforcement. The Script MIB has a number of functions that are required by policy systems in general. Furthermore, some of the requirements listed in Section 2.1 can be met at low costs in the Script MIB context.

- The Script MIB architecture supports pushing and pulling mechanisms for transferring scripts from a SNMP command generator or a script repository to Script MIB agents. Controlling the execution of scripts is also supported, including starting, suspending and resuming, terminating, controlling maximum run times, passing arguments to scripts, etc.

- SNMP security based on SNMPv3 and the user based security model and on the view based access control model is fully applicable to the Script MIB. It is reasonable to build on SNMP security and the Script MIB to achieve a homogeneous security setup.

- Logging and tracing of scripts is a general functionality that is useful for scripts as well as for policies. Although, the current Script MIB does not support access to logging data, the Jasmin implementation already supports logging via the SMX interface between runtime systems and the Script MIB core agent.

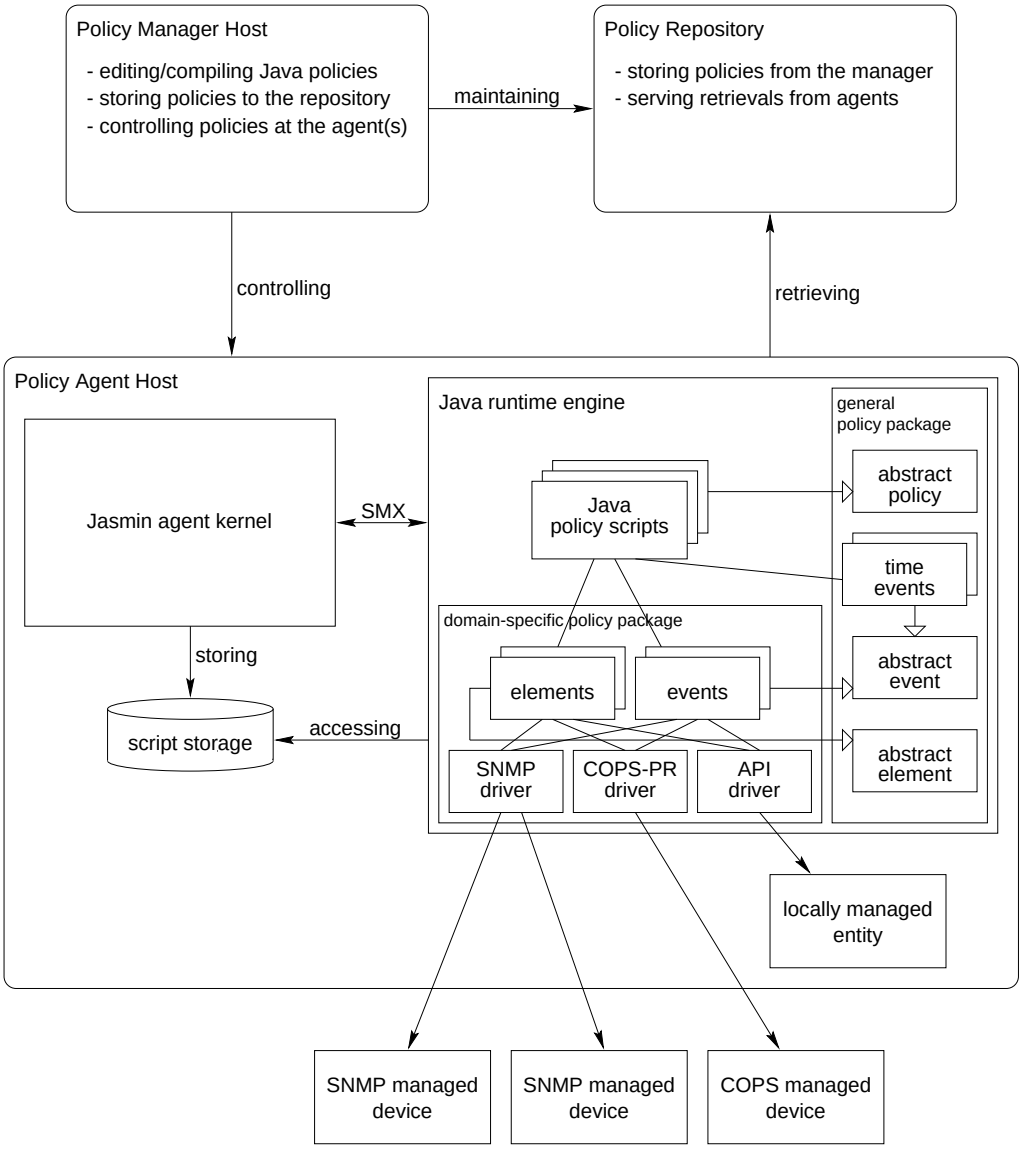

Figure 1: Architecture of a policy management system based on the JASMIN Java runtime engine.

The general architecture of the Script MIB based policy system is shown in Figure 1. This approach is based on the Jasmin Java runtime engine. Hence, policies are Java programs edited and compiled on a manager host. The created `jar`-Files are served to the Jasmin agents by a policy repository via HTTP. When the agent activates a policy, e.g. on demand of a manager, it forks a Jasmin Java runtime engine and forces it via SMX to start a given Java “script” from the local script storage.

Up to this step, the whole process is common to any other Script MIB runtime system. The policy specific components lie within the Java runtime engine, which makes use of Java class packages that support policies. A general policy package contains a number of abstract classes that represent policies (with their rules, conditions, and actions), events, and elements. Besides these abstract classes, domain-specific packages contain derived element and event classes, e.g. to represent classifiers, meters, markers, etc. in case of a DiffServ specific policy package. These domain-specific classes along with classes from the generic policy package, like time event classes, are used by policy "scripts".
The interaction between the domain-specific classes and the managed devices is based on interface drivers hidden from the package API. This allows to support different interfaces, e.g., SNMP, COPS-PR, local APIs or CLIs, without the need to adopt policy scripts to these interfaces or protocols.

2.3 Class Packages and Usage Example

The hierarchy of generic and domain-specific classes is shown in Figure 2.

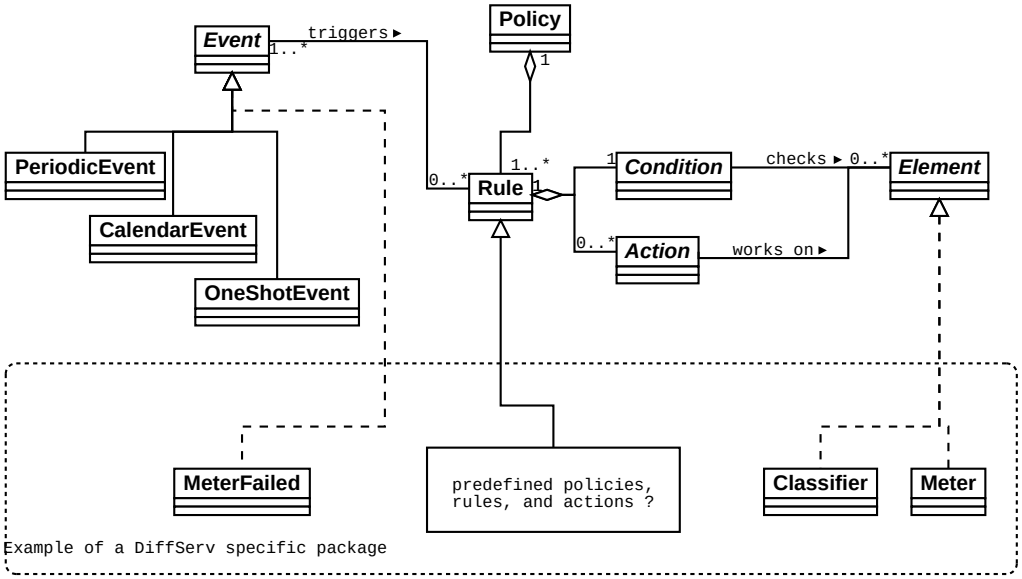

Figure 2: Class diagram of the generic and some DiffServ specific classes.

A policy script's main class has to be derived from the `Policy` class. Usually, it has to instantiate and configure a number of time events and domain specific events. `Rule`s are instantiated and registered with the policy and in turn register the triggering event instances. Each rule also registers an instance of a class implementing the `Condition` interface and as well a number of instances of classes implementing the `Action` interface.

```
public class HomeworkerPolicy extends Policy {
    // This DiffServ policy grants home workers' hosts a higher service
    // at the Remote Access Server during worktime hours. Worktime
    // starts at 8am. It ends at 6pm, but may be delayed in 5 minute
    // steps if staff members are still logged in.
    //
    // We assume the appropriate TCB is already set up and we just have
    // to toggle the diffServDscpMarkActDscp between DSCP_BEST_EFFORT
    // and DSCP_GOLD.

    private final static String RAS_HOSTNAME = "ras.company.de";
    private final static long DS_ACTIONID_HOMEWORKER = 42;
    private DiffServHost ras;
    private DiffServMarkAction homeworkerMarker;
    private SiteMonitor site;

    private static class StopCondition implements Condition {
        public boolean condition(Event event, Vector elements) {
            // triggering event and elements are not used by this condition.
            return site.numActiveUsers() == 0;
        }
    }

    private static class StartAction implements Action {
        public boolean action(Event event, Vector elements) {
            // triggering event and elements are not used by this action.
            homeworkerMarker.setDscp(DSCP_GOLD);
        }
    }

    private static class StopAction implements Action {
        public boolean action(Event event, Vector elements) {
            // triggering event and elements are not used by this action.
            homeworkerMarker.setDscp(DSCP_BEST_EFFORT);
        }
    }

    public static void main() {
        Rule startRule = new Rule();
        Rule stopRule = new Rule();

        startRule.registerEvent(new CalendarEvent("mon-fri h8 m0"));
        stopRule.registerEvent(new CalendarEvent("mon-fri h18-h21 m0/5"));

        // the startRule condition is always true.
        stopRule.registerCondition(new StopCondition());

        startRule.registerAction(new StartAction());
        stopRule.registerAction(new StopAction());

        ras = new DiffServHost(RAS_HOSTNAME);
        homeworkerMarker = new ras.getActionById(DS_ACTIONID_HOMEWORKER);
        site = new SiteMonitor();

        activate();
    }
}
```

Figure 3: A simple Java policy example.

A condition or action can refer to the triggering event by the `event` parameter. Similarly, an action can refer to the combination of elements matched by the condition by the passed vector of `Element` instances. The whole sequence of element combinations that are to be filtered by the condition has to be returned by the overwritten `iterator` method of the `Condition` class. Otherwise, the condition does not operate on any elements and in case of a condition match, the action is called once with an empty `elements` vector.
A very simple example of a Java policy containing two rules is shown in Figure 3. Its goal is to give home workers that are connected to their company through a DiffServ capable remote access server a higher service during work hours. The `main()` method instantiates two rules. The `startRule` is triggered by a `CalendarEvent` each day from Monday to Friday at 8am. This rule has no condition. Hence, its action `StartAction` unconditionally sets the DSCP value of a corresponding DiffServ marker action to the GOLD code point value. In a similar fashion, in the evening from 6pm on the `stopRule` is triggered every five minutes by another `CalendarEvent`. This rule has a condition named `StopCondition` that checks a local site monitor element for the number of active login sessions. Only if no more sessions are active, the condition returns `true` and the `stopAction` is called, which then sets the DSCP value back to BEST_EFFORT.
Note that the `DiffServHost`, `DiffServMarkAction`, and `SiteMonitor` classes used by this imaginary example have to be supported by domain-specific policy packages. Note also, that this example is not intended to represent a compilable Java policy script.

3 Outlook

The next steps within this project will be as follows:

- The classes of the generic policy package have to be refined and implemented.
- The classes of the DiffServ policy package have to be completed, refined and implemented.
- Real-world examples from the DiffServ domain have to be developed.
- DiffServ drivers for the DiffServ MIB and/or the NEC DiffServ API have to be developed.

To fulfill the requirement for a logging facility of the policy runtime system and the Script MIB system in general, an appropriate extension to the Script MIB might be developed and proposed to the DISMAN working group.

3.1 Adaption of a Policy Definition Language

One significant drawback of a policy management system purely based on the Java programming language is the extraordinary expenses that has be spent even for very simplistic or standard policies. For a single policy a minimum of three classes have to be instantiated. Usually, there are some more objects that have to be handled and even some classes have to implemented.
It is expected that a number of standard policy rules can be identified, characterized by a number of parameters. If this would be implemented (denote in the lower center box in Figure 2), this could lead to a smaller number of classes and less lines of code required for some policies. However, the frame of a Java program remains.
The evident next step is to define a policy definition language or deploy one of the languages already developed, that could be compiled to the Java policy scripts. This compilation could take place implicitly on the agent so that the maintenance of policies at the manager and repository gets easier.
A research group that is very active in the area of policy based network and distributed systems management is formed around Morris Sloman at the Imperial College, London[2]. The Ponder Policy Specification Language [13] and the Ponder compiler which is already available might be a good starting point and will be investigated.

References

[1] M. Sloman, J. Lobo, and E.C. Lupu, editors. *Policies for Distributed Systems and Networks - Policy 2001 Workshop Proceedings*, Bristol, 2001. Springer.

[2] *Proc. 6th IFIP/IEEE International Symposium on Integrated Network Management*. Boston, May 1999.

[3] J. Nicklisch. A rule language for network policies. In *Policy 1999 Workshop Proceedings*, http://www-dse.doc.ic.ac.uk/events/policy-99/pdf/26-Nicklisch.pdf, 1999.

[4] D. Levi and J. Schönwälder. Definitions of Managed Objects for the Delegation of Management Scripts. RFC 2592, Nortel Networks, TU Braunschweig, May 1999.

[5] F. Strauß, J. Schönwälder, and J. Quittek. Open Source Components for Distributed Internet Management. In *Proc. 7th IFIP/IEEE International Symposium on Integrated Network Management*, Seattle, May 2001.

[2] Morris Sloman <m.sloman@doc.ic.ac.uk>

[6] J. Quittek, J. Schönwälder, and F. Strauß. *Jasmin - A Script MIB Implementation*. TU Braunschweig, NEC C&C Research Laboratories, http://www.ibr.cs.tu-bs.de/projects/jasmin/, 2001.

[7] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475, Torrent Networking Technologies, EMC Corporation, Sun Microsystems, Nortel UK, Bell Labs Lucent Technologies, December 1998.

[8] DIFFSERV Working Group. *Differentiated Services Working Group Charter*. IETF, http://www.ietf.org/html.charters/diffserv-charter.html, 2001.

[9] F. Baker, K. Chan, and A. Smith. *Management Information Base for the Differentiated Services Architecture*. IETF DIFFSERV Working Group, http://www.ietf.org/internet-drafts/draft-ietf-diffserv-mib-09.txt, March 2001.

[10] M. Fine, K. McCloghrie, J. Seligson, K. Chan, S. Hahn, C. Bell, A. Smith, and F. Reichmeyer. *Differentiated Services Quality of Service Policy Information Base*. IETF DIFFSERV Working Group, http://www.ietf.org/internet-drafts/draft-ietf-diffserv-pib-03.txt, March 2001.

[11] Jae young Kim. *POSTECH DiffServ MIB Implementation*. http://dpnm.postech.ac.kr/research/01/ipqos/dsmib/index.html, 2001.

[12] D. Levi and J. Schönwälder. Definitions of Managed Objects for Scheduling Management Operations. RFC 2591, Nortel Networks, TU Braunschweig, May 1999.

[13] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. Technical report, http://www-dse.doc.ic.ac.uk/ mss/Papers/Ponder-summary.pdf, aug 2000.