SPCL: Structured Policy Command Language

Michael Locasto Matthew Burnside Chun Li
Aron Wahl

May 13, 2003

Contents

1 Introduction

The Structured Policy Command Language (SPCL) is designed to address a number of issues found in policy and permission systems. The notion of a policy is central to the design and operation of most complex systems. Because the concept of policy is a nebulous one, efforts to formalize it have suffered from both specialized domain requirements and the inherent complexity of policy decisions. The SPCL is an effort to provide a domain-independent language for specifying policy requirements, principals, rules, and actions in a lightweight syntax with semantics as close to natural expression as possible, in order to avoid esoteric and complex programming constructs. If need be, SPCL can be compiled to another form, such as that supported by the Keynote trust management system.

Current policy systems are either built into a system (making them hard to adjust or replace) or too course grained to be effective and easy to use. There is a clear need for a policy language that greatly simplifies the task of policy specification and separates policy specification from policy implementation. In addition, policy is often specified by non-programmers, so any language should be sufficiently easy to learn while retaining powerful specification abilities for systems programmers or application architects.

It is important to note that policy is not merely access control. Policy is essentially about expressing a want or desire. Therefore, we can conclude that policy is about *specifying*, *enforcing*, and *revising* requirements. Any system built to handle policy should provide specification, enforcement, and revision mechanisms.

1.1 Background

1.1.1 What is Policy?

Policy is generally considered to be a plan or course of action intended to influence decisions or outcomes. A policy is a set of guiding principles whereby some outside agency can match its current environment or task with the guidelines in the policy and proceed with a course of action.

Policy decisions range from very simple (accept or reject access request N) to very complex (never let the dogs outside unless someone is home and their collars are on and the electric fence is on and it is not raining) to extremely complex (allow half of this customer's address space to advertise their prefix with us, but do not let their traffic use us as a transit network, unless the lawyers say otherwise).

Policy is ubiquitous in computing systems, and is often implicitly coded into a system's structure by functional requirements, language features, and design decisions. Policy is not easily managed because it is so high-level and rarely translated to formal program semantics.

1.1.2 What are some examples of current policy mechanisms?

A language-specific attempt to provide a policy mechanism is the Java Policy and Permissions objects. While the Java approach is a fairly good attempt, it suffers from two

major weaknesses: difficulty of use and lack of a fine-grained approach. The file used to control the policy rules in the currently running JVM is often ignored because it is an additional layer of complexity in the design, construction, and testing of a system. The default file controlling Java policy (if a SecurityManager is invoked in the current JVM) is located in the JAVA_HOME/jre/lib/security/java.policy file, and has a series of ”grant blocks” that allow a set of classes to have a predefined set of permissions (essentially an arbitrary collection of statements permitting the set of classes to invoke certain language functions). The Java policy file syntax is roughly the following:

```
grant SOME_CLASSES {
      permission SOME_PERMISSION “args”;
      permission ANOTHER_PERMISSION “args”;
}
```

In addition, the default policy mechanism is very course-grained and although it is easily extensible, does not lend itself to quick analysis of any given software. Even though there are a number of permissions to legislate, this set of permissions is not provably complete or precise enough for many complex applications.

Another effort to specify a policy language is the Web Services Security Policy Language by IBM, Microsoft, Verisign, and RSA. However, this policy language is domain-specific and meant to provide an XML-grammar based mechanism for policy assertions for web services.

Yet another system that involves policy specification is KeyNote, a trust management and policy system. While KeyNote is a powerful mechanism, its policy specification syntax is not for the faint of heart.

1.2 SPCL Features

1.2.1 SPCL Highlights

SPCL’s primary two features are an object-oriented-like command language (and compiler) and a PolicyEngine interpreter acting as an oracle to a number of other systems. SPCL’s syntax is closest in appearance to a mixture of Java and SQL. The PolicyEngine should be thought of as a VM for policy.

A common pattern for specification is of the form:

{command} {principal} performs {action} on {object} when {conditions}

A policy is specified in the SPCL syntax, parsed, and then translated to a form suitable for input to the PolicyEngine. The specification for the PolicyEngine is platform-independent, so it can be implemented for any target platform. After parsing and construction, it then loads and interprets the policy. The PolicyEngine has a simple protocol for asking questions about the running policy (in SPCL) as well as loading new policies in, updating current policy, and retiring outdated policy.

1.2.2 SPCL Goals

The SPCL has three primary goals that correspond to the essential components of a policy mechanism. SPCL must support policy specification, enforcement, and revision.

First, in order to support **policy specification**, the SPCL should provide a syntax that is close to the natural language method of specifying policy rules and permission constraints. SPCL is necessarily a high-level language. Ease of use and ease of specification is expected to increase because of this goal.

In the absence of natural language processing support for policy (which may be rectified by research conducted by the Navy), this pattern of principals, actions, and objects very naturally suggests itself. SPCL, like SQL, is a language that seeks to be easy to use while remaining focused, formal, and powerful.

The important point here is the focus on the user of the system. The user is actually writing the policy as they express their desires. However, they should have to do a very small amount of work to express their policy formally, and this work should not involve strange syntax or traditional source code structures a 'programmer' would know how to manipulate. This requirement is aided by the simple command-oriented nature of policy. Most policies have little hierarchical structure; they specify a set of actions under given conditions.

Second, in order to support **policy enforcement**, the SPCL should have semantics that are not costly to interpret, have little or no side effect other than the intended goal of the policy, and are domain independent.

Third, to support **policy revision** SPCL should have some idea of system lifecycle. Most policy is implemented implicitly or statically in program code and is difficult and expensive to change. Policy needs to be dynamically updateable, and should respond to a lifecycle model. Knowledge of the lifecycles for both the policy and the systems the policy governs will allow the PolicyEngine to make informed and powerful decisions. Users must be able to create, alter, and retire their policy easily, without a costly impact on the rest of the system.

1.3 Example Policy Domains

1.3.1 Authentication & Authorization

Authentication and authorization is the most simple and common sort of policy implementation in systems today. SPCL is useful for this base case of binary accept or reject decisions, such as those faced during a login procedure or access request. An example policy for authentication is:

Reject any user who provides the wrong password.

Reject any user who has three successive bad login attempts.

1.3.2 Firewall Rules & Packet Filtering Filtering

IP packet traffic is another application of policy. The network administrator desires traffic into and out of her network to have certain characteristics. Packet filters are often simple tables of rules. Traffic is pattern matched against the table, and the specified

action is performed. A common tool for packet filtering is the Linux tool ipchains and has syntax like the following:

ipchains -A input -j ALLOW -p tcp -s 0.0.0.0/0 -d www.mycompany.com 80

to express a policy of:

Allow anybody to access my webserver over TCP on port 80.

Note the natural structure of the command, however:

Add an 'ALLOW' rule to my 'input' policy for a principal '0.0.0.0/0' on my object 'www.mycompany.com' with the condition that the protocol is TCP.

1.3.3 BGP

BGP, or Border Gateway Protocol (v4), is the protocol used on the Internet to advertise routes between computers. BGP does not calculate the optimal path (shortest path) for traffic; rather, BGP calculates the best path for traffic, where best is defined by an arbitrary policy: legal and business agreements.

1.3.4 House Monitoring System

The application that SPCL focuses on for motivating examples is a home monitoring system. This system is a policy-aware control application for home appliances. Homes and families have many arbitrary rules that govern use of various appliances and tasks that need to be done in order for the home to run smoothly.

1.4 SPCL Language Primitives

The language is a declarative-style specification language; it is not an imperative one. The language does allow for traditional branching control flow (if..then..else) even though its model of computation is primarily event based. It has the following primitives (some of which directly translate to reserved words):

- zone - an area containing many policy definitions generally under 1 administrative domain.
- policy - a policy definition
- principal - an object in the system that acts autonomously
- object - an object in the system that is acted upon
- group - a group of principals
- action - a flexible but concrete notion of what a principal attempts to get permission for
- rule - a regular expression for a class or group of actions

- condition - an expression to be satisfied in order for the policy engine to respond to the user.
- meta-action - the action taken by the system in response to a principal's request to perform an action on an object
- constraint - reserved for future use
- requirement - reserved for future use
- assertion - reserved for future use
- demand - reserved for future use
- consequence - reserved for future use

The language operates with a data model that focuses on events that affect a three-tiered map of principals, groups, and objects. Events can query the state of the map (a relationship between map objects) or alter the state of the map (change the current policy), and three precedence levels: user, group, and default group. The policy engine will match rules beginning with the specific user, then against groups the user belongs to, and finally against the default group.

1.5 Conclusions

SPCL is a language meant to simplify the expression of arbitrary policy. The primary goals of SPCL is to formalize the expression of policy, separate the specification of policy from the implementation of the system the policy governs, and provide the system with a lifecycle-aware policy mechanism. Meeting these goals provides a policy mechanism that supports the specification, enforcement, and revision of policy.

2 Language Tutorial

2.1 Introduction

SPCL is a domain-independent language for specifying policy requirements. Generally speaking, a policy is a set of principles that defines what is permissible in a specific domain. SPCL allows you to define a domain, the agents that exist in the domain, the objects upon which the agents may act, and the actions that apply to said objects. This tutorial will introduce you to the basic elements of SPCL and help you write and test your first SPCL program.

2.2 A Simple SPCL Example

Let's begin by looking at a simple SPCL program. This program serves as a hello world type program:

```
zone house;

policy MyPolicy{

  default{
        deny *;
  }

}
```

This policy defines a zone *house* where every action is denied.

Your second helping of SPCL defines a policy for the four members of the Smith family – parents George and Glenda, and children Timmy and Sarah – who live together in an ulra-modern computerized home.

George is on a diet so he should only be able to access the refrigerator during certain hours. Neither of the children should watch late night televsion, and Timmy is not allowed to watch the Playboy channel. Sarah has been known to sneak out of the house after her eleven o'clock curfew. Glenda is perfect. She's the policy administrator and she would like to know if Sarah breaks her curfew. The following SPCL file, HouseRules.spcl, will keep the family members' vices in check. If you don't understand the following code right away, don't worry:

```
zone smithhouse;

policy HouseRules {

 default{
   deny *;
 }


 object front_door {
   actions {
```

```
     action unlock = "unlock";
     // tell mom if somebody tries to unlock the door
     open_door.meta_action = notify Mom;
  }
}

object refrigerator {
  actions {
      action open = "open";
      open_frige.meta_action = notify Mom;
  }
}

object tv {
  string channel = "Fox5";
  actions {
      action watch = "watch";
  }
}

group Parents {
  allow *;
}

group Children {
  allow open on refrigerator;
  allow unlock on front_door;
  // 'system.time' returns the current time
  allow watch on tv
   when ( system.time <= "9:00 pm");
}

principal Mom {
 alias Glenda, supermom, ADMIN;
 member Parents;
}

principal Dad {
 alias George;
 member Parents;
 deny open on refrigerator
   when ( system.time >= "8:00 pm");
}

principal Son {
 alias Timmy;
 member Children;
 deny watch on tv when ( tv.channel == "Playboy");
}
```

```
principal Daughter {
 alias Sarah;
 groups Children;
 deny unlock on front_door
   when ( system.time >= "11:00 pm" );
}
```

2.3 Elements of the SPCL Language

In order to understand the above program, you must first familiarize yourself with some of the SPCL basics. The following are fundamental elements of SPCL:

2.3.1 Zone

A zone the area to which one or many policies apply. A zone is generally under one administrative domain. In the above program, the zone is the Smith household, called *smithhouse*.

2.3.2 Policy

A policy is a policy definition as defined in an SPCL program. A policy definition may contain the elements described below.

2.3.3 Principal

A principal is an object in the system that acts autonomously. Principals attempt to act upon objects. *Mom*, *Dad*, *Son*, and *Daughter* are principals in the above program.

2.3.4 Group

A group is a collection of pricipals. The above program defined two groups – *Parents* and *Children*.

2.3.5 Object

An object is something that can be acted upon by principals. HouseRules.spcl defines three objects – *tv*, *frontdoor*, and *refrigerator*.

2.3.6 Action

An action is a definition of what a principal may do to an object. A principal must have permission to perform an action on an object. For example, the principals in the above program may *open* the refrigerator object.

2.3.7 Condition

A condition is a expression that must be satisfied in order for a principle to act upon an object. In the above program, for example, the *Son* principal cannot perform the *watch* action on the *tv* object if the tv's channel is Playboy.

2.3.8 Meta Action

A meta action is the action a system takes in response to a principal's request to perform an action on an object. In the above example, Mom is notified if her daughter opens the front door after 11 pm.

2.3.9 Rule

A rule is a statement that defines, for a given principal or group, what actions may be performed on what objects. In the previous program, rules are the statements that begin with *allow* and *deny*.

2.3.10 Comments

Comments are sequences of characters included in SPCL code that describe the code itself. Comments are ignored by the SPCL compiler. Comments are designated in one of two ways; either by lines that begin with '//' and end with a newline character or by a block of text that begins with '/*' and ends with '*/'.

2.3.11 Star Glob Operator

The '*' is used in SPCL as you would use it in Perl. The command 'deny *;' means deny everything.

That should be enough to get you started. For more detailed explaination of the SPCL syntax, consult the SPCL language reference manual.

2.4 Quick Analysis

Let's run through HouseRules.spcl. After declaring the zone and policy name, the program sets the default values. SPCL contains the notion of precedence for rules. Essentially, rules with a the highest precedence overide rules with lower precence. Rules defined in the *default* block of code have the lowest precedence, followed by group rules, followed by principal rules. For the default case, the program denies everyone permission to do anything.

After the default case, the program declares two groups – 'parents' and 'children'. Let's look at one of them:

```
group Children {
  allow open on refrigerator;
  allow unlock on front_door;
  allow watch on tv when (system.time <= "9:00 pm");
}
```

The above block of code overrides the default case. The children are now allowed to open the refrigerator, unlock the front door, and watch tv if it is before 9 pm. The default case makes certain that the children cannot watch tv after 9 pm, because the more specifc allow rule will fail to match on the conditions and defer to the default. Explicitly providing a deny rule for after 9 pm would accomplish the same thing, but is redundant.

The program defines four principals – Mom, Dad, Son and Daughter. Let's take a closer look at one of those definitions:

```
principal Son {
 alias Timmy;
 members Children;
 deny watch on tv when ( tv.channel == "Playboy");
}
```

Since a rule defined within a principle code block has higher precedence than a rule defined within a group code block, the Son principal cannot watch the Playboy channel regardless of time.

The variable channel is defined in the 'tv' object declaration:

```
object tv {
  string channel = "Fox5";
    actions {
      action watch = "watch";
    }
}
```

Note that the channel is set to "Fox5". Objects in SPCL have a state. When a request is made about an object, the Policy Engine uses the object's state to determine whether it should accept or deny the request. The *channel* variable is an aspect of the *tv* object's state. Assume that some interface exists between the tv object and a tv. That is to say, the *channel* variable would change when the channel changes on the actual tv to which the object refers.

2.5 Installation

If you did not receive a copy of the Compiler and PolicyEngine with this tutorial, you can download it from our website or access it via CVS, or ask your friendly neighborhood group member to provide you with a copy.

Now that you have a basic idea of how to write an SPCL program, it is time to install SPCL on your system. First, make sure that you have a Java Development Kit (JDK version v1.4.x is required) installed. If you do not have a JDK, you may download it from the Sun website at (http://java.sun.com).

Next, download either 'spcl-0.x.tar.gz' or 'spcl-0.x.zip'. Now copy the file into the directory from which you would like to work. You must now decompress the file. NOTE: The dollar sign is the command-line prompt so do not include it in your commands.

If you downloaded 'spcl-0.x.tar.gz' type:

```
$ gzip -d spcl-0.x.tar.gz ; tar -xvf spcl-0.x.tar
```

If you downloaded 'spcl-0.x.zip', type:

```
$ unzip  spcl-0.x.zip
```

Now the file should be decompressed. Type 'ls' to make sure. In the bin/ directory, there is a Makefile and some shell scripts. You must to edit both the JAVA_HOME and the SPCL_HOME variables at the top and set them to the appropriate directories. For example, if you installed to

```
/home/professor/opt/spcl
```

then

```
SPCL_HOME=/home/professor/opt/spcl
```

(NOTE: there is no trailing '/'). In addition, if Java is in /usr/java/sdk/jdk1.4.1 then JAVA_HOME=/usr/java/sdk/jdk1.4.1 (NOTE: again, no trailing '/').

Currently, there are two different makefiles but they will be merged. The Makefile will set up your CLASSPATH properly (to include SPCL_HOME/classes/).

In addition, you should edit the pole.properties file provided in SPCL_HOME/conf/ to reflect the available ports and shutdown message for the PolicyEngine.

2.6 Running A Program

Now that you have SPCL installed on your system, try running a program. If you're feeling lucky, try writing your own program. Otherwise, using a text editor of your choice either cut and paste or recopy the above program, HouseRules.spcl, and save it as 'HouseRules.spcl'.

The SPCL compiler transforms a given policy into java code, which it then compiled into appropriate the Java ".class" files. The policy can then be loaded into the SPCL policy engine, the mechanism which handles queries regarding the policy. To compile your policy program, execute

```
SPCL_HOME/bin/polc xxx.spcl
```

at the command-line, while in the directory from which you have decided to run SPCL. For example, you might type:

```
$ /home/professor/bin/polc.sh HouseRules.spcl
```

If your program is correct, this will invoke the SPCL compiler to produce xxx.java. Then it will invoke the java compiler to produce xxx.class and to place xxx.class in the directory

```
'SPCL_HOME/conf/policy'.
```

Now, you can invoke the PolicyEngine on this policy by typing:

```
$ SPCL_HOME/bin/pole.sh <policy>
```

Notice that there is no file extension. This will run the policyengine and load in the given policy class. You can stop the policy engine by typing:

```
$ SPCL_HOME/bin/pole.sh -stop
```

Otherwise, you can use the client to connect to the policy engine and query it:

```
$  SPCL_HOME/bin/pec.sh
```

Keep in mind that you are playing the role of an object asking the policy something on behalf of the user. Now, try querying your policy via the policy client. Start by typing:

```
$ list zones
```

If you used the above example, this should return the value 'smithhouse'.

You can also switch zones. Typing the following command will take you into the built in default zone.

```
$ switch zone zone:zone
```

Next, try typing:

```
$ list principals
```

This should return a list of all the principals you declared in your policy program. You may also use the *list* command to display groups, objects, and policies. The *current* command allows you to quickly determine your current scope with a zone,policy,object triple.

Now you are ready to make requests. In SPCL, a request applies to a specific object. A request has the following form:

```
$ can [principal] do [action] with ([conditions]) ?
```

It is important to realize that the conditions specify the current state of the object. They are an AND'ed list of variable to value associations of the form:

```
with ( y = 2 AND channel = 'playboy' )
```

To change the object to which your request applies, type:

```
$ switch object [object]
```

Of course, the principal must be something you have declared and the action and condition must apply to the object about which you are making the request. You may leave the condition empty (i.e. just use empty parenthesis) if you wish to make a more general query. If you used the SPCL program from the beginning of this tutorial, try the following commands. First, change the object in question to tv, if you have not done so already:

```
$ switch object tv
```

Then check if timmy can watch television:

```
$ can son do watch with channel = 'PlayBoy' ?
```

Note that even if the system time is earlier than 9 pm – the children group to which timmy belongs cannot watch tv after 9 – the policy engine will deny the request. Why? The channel variable in the tv object declaration is set to 'Playboy'. A rule in the Son principal declaration specifically forbids the principal from watching the Playboy channel. Continue making requests until you feel comfortable with the syntax.

2.7 Writing An Advanced Program

Congratulations on having run your first SPCL program. Now you're ready to write a more advanced program. Perhaps you could try adding neighbors and relatives and the police to the above program. First try adding new principals and overlapping groups to your program. This will help you get familiar with SPCL's precedence rules.

Next, try adding conditional side effects to your program to rules in your program. Consult the Language Reference Manual for proper syntax and rules. Conditional side effects are triggered whenever their rule is being applied to a request. Condition side effects may either trigger meta-actions (ie 'log' an event or 'notify' the policy administrator) or update a rule, or execute arbitrary Java code.

Now, try experiment with rule updates. Rule updates have the highest precedence, overriding rules declared in the principal blocks to which they apply.

2.8 Tutorial Conclusion

Hopefully, you found this tutorial to be helpful. You should now be an expert in SPCL programming. Consult the Language Reference Manual and the SPCL Whitepaper for a more detailed exploration of the topics discussed above. Or purchase the O'Reilly SPCL book with the half–platypus half–monkey on the cover. Have fun!

3 Model of Computation

The model of computation for SPCL's runtime environment is very similar to Prolog. The runtime environment is represented by the PolicyEngine and defines various namespaces called 'zones'. The PolicyEngine bootstraps each zone with a policy as defined by an SPCL source file. Then, the runtime environment loads and resolves the objects and references in the zone's policy. Any system is now free to talk the SPCL protocol (make SPCL 'CAN' queries) to the PolicyEngine.

The PolicyEngine runs the following algorithm to answer a query:

```
query( principal, groups, objects, action_string, conditions, zone) {
        switchTo[ zone ];
        conditions = conditions ⋃ get[ current_conditions ];
        find[ principals and groups ];
        ∀( principal ){
                find[ rules ]; //find relevant rules
                resolve[ action_string, conditions ] with [ object.actions ]
                if( match ) rule.fire() AND return;
        }
        ∀( group ){
                find[ rules ]; //find relevant rules
                resolve[ action_string, conditions ] with [ object.actions ]
                if( match ) rule.fire() AND return;
        }
}
```

The unification of the current conditions with the required variable relationships and current state information is a critical step of the algorithm. In addition, each rule must be tested to make sure it is currently active. These implementation details are discussed throughout the PolicyEngine.java file, the Rule.java file, and the Condition.java file. Note also that there is an ordering between principals and groups that reflects the precedence levels of these entities.

4 Language Reference Manual

4.1 Introduction

This section is a description of the syntax of Structured Policy Command Language (SPCL). Before proceeding to the details of this language, it is highly recommended to familiarize yourself with the overview and the goal of SPCL, which could be found in prior sections.

The overall structure of SPCL is as follow:

1. Each SPCL file is associated with a distinct policy under a finite administrative domain, known as zone.
2. Each policy is composed of objects, principals, and groups.
3. Objects could either be some representation of the environment or some resources of the system to be accessed by users.

4. Principals are the entities or users of the system. Associated with each principal, among with others, are a set of rules describing the parts of the resources that could be accessed or the parts that are restricted. There is a default principal describing the privileges of any unidentifiable principals.

5. Groups exist solely for the convenience of the administrator for organizing principals with the same privileges under the same zone.

Again, the above discussion is just a brief abstraction of SPCL. For further details, please refer to the corresponding sections below.

4.2 Lexical Convention

There are five kinds of tokens: keywords, identifiers, numbers, strings and operators. In general, as in most languages, white spaces are ignored except as they serve to separate tokens. When parsing tokens, the longest matched string of characters is used, unless otherwise specified.

4.2.1 Comments

A comment is a sequence of characters served as notes for human beings, and is ignored by the compiler. Two forms of comment are available.

1. Line comment, which starts with // and end with the first occurrence of a line terminator.

2. Block comment, which starts with /* and end with the first occurrence of */.

4.2.2 Identifiers (Names)

An identifier is a sequence of letters, underscores and digits, with the first character being a letter. Upper case and lower case letters are considered different.

4.2.3 Keywords

All keywords are case sensitive. The following identifiers are reserved for use as keywords, and may not be used otherwise:

```
zone                allow
policy              deny
default             on
group               by
principal           when
object              if
alias               else
url                 log
email               notify
phone               meta_action
member              system
actions             constraint
action              requirement
number              assertion
```

```
string              demand
boolean             consequence
true
false
```

Some keywords, including constraint, requirement, assertion, demand and consequence, are not used in the current design of SPCL. They are reserved for future uses.

4.2.4 Strings (Values)

A string is a sequence of characters surrounded by double quotes ‘ ” ’. The meaning of this string value is system dependant. The language itself merely treats it as some mysterious sequence that describes the condition of certain object, or a representation of certain actions to be performed. Escape character is not permitted in the current design.

4.2.5 Numbers (Values)

A number is any real number with an optional sign in front of it. E-notation is not supported in this language. A real number is a sequence of digits with at most one optional decimal point some where in this sequence.

4.2.6 Operators

Operators are further divided into five different sub-types, known as relations, assignment operator, star sign, dot operator and structural operators.

4.2.7 Relations

A relation must be one of the following: “>”, “<”, “<=”, “>=”, “==”, “!=”. A relation, along with variables and values, are used for expressing conditions of the environment.

4.2.8 Assignment Operator

The assignment operator is a single equal sign ‘=’. It is used to initialize or change the value of certain variables.

4.2.9 Star Sign

The star sign is a single star ‘*’ character that is not surrounded by any double quotes. It is used to express the idea of “everything”. Depending on the context of the statement, “everything” might have slightly different meanings. When this token is permitted in a statement, its actual meaning would be further defined in the corresponding section.

4.2.10 Dot Operator

The dot operator is a single period ‘.’. This is used to refer variables of a specific object. Please refer to section section 4.6 for further details.

4.2.11 Structural Operators

Structural operators serve as the boundary of certain structural blocks and have virtually no meaning in the context of SPCL. These operators include: open parenthesis '(', close parenthesis ')', open brace '{', close brace '}', comma ',', semi-colon ';'. Although no semantic is bound to any of these operators, the exact location on which any of them could be placed is strictly defined by each structural block.

4.3 Syntax notation

The syntax notation used in this manual follows the guideline of Antlr's tool. In general, it is in the form

syntax : $rule_1|rule_2|rule_3 \ldots$;

where "syntax" is a name that represents that particular syntax and all the rules, $rule_1$, $rule_2 \ldots rule_n$, are the possible choices of this syntax. A few other notations are also used in this manual:

syntax : $(rule_1)*$; // zero or more occurrence
syntax : $(rule_1)?$; // zero or one occurrence
syntax : $rule_1 rule_2 rule_3 \ldots$; // concatenation of rules

The author believes that the above notations shall suffice in the scope of this manual. If reader is not satisfied with the above description, please refer to Antlr's homepage, "http://www.antlr.org/", for further details.

4.4 Zone

A zone is defined to be a finite administrative domain whose name must be agreed upon in between both the administrator and the system. In other words, when the administrator defines a policy on certain resources R under a specific zone Z, the system or the resources of the system, R has to know that it is under that zone named Z. This should not be a problem since the administrator is supposed to be the person who has absolute knowledge of everything under his/her administration.

The syntax of zone declaration is as follow:

zone : "zone" ID ';' ;

At the beginning of each SPCL file, the above declaration has to be present in order to tell which administrative domain the policy belongs to. Zone is the keyword, while ID is the name of the zone that is bound to the specific policy.

4.5 Policy

Conceptually, a policy is a set of rules governing the behavior of a system under a given zone.

In the context of SPCL, policy is the main body of the whole file. For the current design of SPCL, only one policy is allowed per file, i.e. each SPCL file consists of exactly one zone declaration and exactly one policy declaration.

Each policy, in turn, is composed of various sub-components, namely object definition, group definition, principal definition, and default principal definition. These

sub-components have to be in the given order. As a result, the syntax of policy becomes:

```
policy : “policy” ID ‘{’
                default_definition
                (object_definition)*
                (group_definition)*
                (principal_definition)*
        ‘}’;
```

Policy is the keyword, while ID is the name of this policy. Each sub-component is further defined in subsequent sections. Note that default_definition is a required component.

4.6 Object Declaration

An SPCL object is an abstract data that represents a resource that is available on the system. With the exception of predefined object “system”, all objects must be explicitly declared in the document. Defining an object “system” is subject to compiler error.

4.6.1 User Defined Objects

The exact syntax of object declaration is as follow:

```
object_definition : “object” ID ‘{’
                            ( “url” ‘=’ string ‘;’ )?
                            ( variable_declaration )*
                            ( “actions” ‘{’ single_action (single_action)* ‘}’)?
                    ’}’;
variable_declaration : (“number” ID (‘=’ number)? ‘;’)
                     | (“string” ID (‘=’ string)? ‘;’)
                     | (“boolean” ID (‘=’ (“true”|“false”) )? ‘;’)
                     ;
single_action : “action” ID ‘=’ string (‘,’ string)* ‘;’
              ( ID “.meta_action” ‘=’ ( “log” | “notify” ) pID ‘;’)?
            ;
```

Object is a keyword for declaring object while ID is the name that binds to it.

Within the braces, the first option, url, is meant to be the identification of the object in the real world.

The second option is an enumeration of all variables that belong to this object. There are three possible types for a variable: “boolean”, “number” or “string”. The value after equal sign, when present, is the initial value of a variable. When referring to variable A of object O, use the ID of O followed by a single dot operator ‘.’ followed by the ID of A. This is exactly the same convention used in many modern object-oriented languages, such as C++/Java.

The third option enumerates all possible actions on this object, using the keyword actions. All of the string above complies with the definition stated in section 4.2.4. Furthermore, they are treated as regular expressions, for matching actions requested to

the policy engine. If the optional meta_action exists for a given action, whenever that action is requested, the meta_action would be executed. See section 4.11.6 for details about meta action.

4.6.2 Predefined Object

There is a predefined object called "system". In the current design of SPCL, there are two predefined variables for object "system", namely $state$ and $time$. They are both of type string.

The first one $state$ is a string description of the current status of the environment. The meaning of this string is system dependent.

The second one $time$ is a string representation of current system time in local time zone. The smallest distinction of any two given time is one minute. The time is of the form "HH:MM AP", where HH represents the hour and MM represents the minute. AP could either be "am" or "pm" and must be in lower case. A day starts at time 12:00 am and ends at 11:59 pm.

To refer to one of these predefined variables, use system.variable_name, where variable_name is the variable that one is trying to refer to.

4.6.3 Predefined Variable

In addition to predefined objects, there are also predefined variables, that are applicable to every single defined objects. Currently, only state is available. The idea behind this variable is to encapsulate the current status of certain resources. In order to examine this information, one may use objectID.state, where objectID is the object to be examined. There is no need to define this variable prior to usages. Furthermore, if any one is trying to declare a variable with name, state, compiler error shall be generated.

4.7 A Quick Glance at Rule

The complete definition of rule is rather involved. Please refer to section 4.11 for a detailed discussion about the syntax of rule.

However, in order for reader to understand the structure of group, principal and default declaration, the author foresee a need to explain briefly the meaning and structure of rules. To present the general idea of rules, the simplest form is provided as follow:

rule : ("allow"|"deny") (actionID) "on" oID ';' ;

The above definition says that each rule states whether the administrator ("allow") allows or ("deny") denies certain form of actions (actionID) on certain objects (oID). Each rule is applicable only to the entity that embraces it, unless overloaded by other optional factors.

All objects must be explicitly declared in the document. Also, all actions must be referring to one of the actions defined under the target object. If any of the above assertion fails, a compiler-error shall be generated.

Again, though the above discussion is perfectly compatible with SPCL's syntax, it merely exhibits the basic functionality of rules. For further details, please refer to the discussion below in section 4.11.

4.8 Group Declaration

Conceptually, a group binds principals with similar privileges together. It helps the administrator to organize regulations among many users.

In the current definition of SPCL, each principal could belong to any number of groups. Once a group is given a certain amount of privileges or restrictions, all members of that group would be given the privileges.

A group declaration assures the existence of a group and provides it with a unique name. The associated rules for the group are also defined. But, the member that belongs to it is not declared here. For binding members to groups, see section 4.9.

The syntax of a group definition is as follow:

```
group_definition : "group" ID '{'
                        (rule)*
                   '}';
```

group is the keyword for declaring a new group. The ID following group is the name used to refer to this group. Within the body of a group definition, there could be zero or more rules that are applicable to every member of this group. Rule is defined in section 4.11.

4.9 Principal Declaration

A principal is a representation of a user in a system. Each principal could belong to any number of groups discussed in the previous section (4.8). The syntax of principal is as follow:

```
principal_definition : "principal" ID '{'
                                ( "alias" '=' ID (',' ID)* ';' )?
                                ( "url" '=' string ';' )?
                                ( "email" '=' string ';' )?
                                ( "phone" '=' string ';' )?
                                ( "member" '=' gID ( ',' gID )* ';' )?
                                ( rule ) *
                       '}' ;
```

The declaration starts with the keyword principal, followed by an identifier that uniquely determines the principal from other entities.

The first line is a list of all optional aliases used in the SPCL. i.e., the administrator could refer to a particular principal using its unique principal ID as well as these aliases. The names used by aliases are also unique identifiers in an SPCL document. As a result, in the context of SPCL, "the name of a principal" implies either the actual ID of the principal or one of the aliases of that particular principal.

The three optional lines followed are meant to be the real world's identity of the principal. For the current version of SPCL, only the keyword email and phone are permissible. These are used when a meta-action is defined on this principal. See section 4.11.6 for further details.

The next option, member, is the binding in between groups and principals. Each gID is the identifier for a group that is defined in the document. If gID is not found, a compiler error shall be generated.

For the definition of rule, see section 4.11.

4.10 Default Declaration

The default declaration describes the behavior of an unidentified user. It also serves as the final consult whenever related rules cannot be found in both the principal's declaration and all of the related groups' declaration. This is the only required declaration in an SPCL document. The syntax of default declaration is as follow:

```
default_definition : "default" '{'
                          ( rule ) *
                     '}';
```

The declaration of default block starts with keyword default followed by any number of rules surrounded by braces.

4.11 Rule Declaration

A rule in SPCL is a statement that binds principal with certain privileges or restrictions. In other words, a rule says what can be done or can't be done by a person.

4.11.1 Basic Rule

basic rule is the simplest form of rule that is used by an SPCL document. The syntax is as follow:

```
basic_rule      : ("allow"|"deny") (action_list) "on" oID ;
action_list     : action (',' action)* ;
action          : string ;
```

In the current design of SPCL, allow and deny are the only keywords for granting access or restricting access on any resources in the system.

Following one of these keywords, is a list of actions separated by comma. Each of these actions is a string, which follows the definition in section 4.2.4, that matches with one of the many actions defined in object oID. See section 4.6 for action definitions.

Actions stated in a rule are always treated as explicit strings that represent specific actions as opposed to a regular expression in object declaration. All actions must be surrounded by double quotes.

The last part starts with keyword on, followed by oID, where oID is the name of an object declared some where in the document. This object tells where the action is going to be performed on.

(Note: If you compare the definition state here with the definition stated in section 4.7, you may noticed that a semi-colon is missing. This is because the semi-colon is the terminator of a complete rule while the basic rule is just the beginning of the whole story. Other options discussed in subsequent sections shall be placed before the final terminator. Also note that semi-colon is just one possible way to terminate a rule)

4.11.2 Usage of *

In the context of rule definition, a star sign '*' is used as a shorthand for the notion of every possible action. When this operator is included in the rule definition, the syntax is as follow:

```
basic_rule          : ("allow"|"deny") action_and_target;
action_and_target   : ( '*' ( "on" oID )? )
                    | (action_list) "on" oID ;
```

When the optional keyword "on" exists and is followed by an object ID (oID), the star sign is a short hand for all possible actions defined in object oID. If the target object is not stated, the star sign means all possible actions on all defined objects.

4.11.3 Optional By-Clause

The first option that follows the basic rule is a by-clause. The syntax is :

```
by_clause : ("by" pID (',' pID)* )? ;
```

A by_clause starts with the keyword by, followed by a list of names separated by commas. Each of these names refers to a principal or group defined some where in the document. If the principal or group is not defined, a compiler-error shall be generated.

Normally, when a rule is defined, it is being imposed onto the entity that embraces the declaration of the rule. If this optional by-clause exists in the rule, the rule is being imposed onto the entity/entities defined in the list instead.

This functionality is only available in the rules used in rule updates. See the details in section 4.11.5.

If the by-clause exists in any rule other than a rule updates, a compiler-error shall be generated.

4.11.4 Optional When-Clause

Following the optional by-clause is the when-clause. The syntax is as follow:

```
when_cause : "when" '('
                    oID '.' vID relation value
                    ( ',' oID '.' vID relation value ) *
             ')' ;
value      : string | number | "true" | "false" ;
```

A when-clause begins with keyword when, followed by an open parenthesis, followed by a list of conditions separated by commas, and is terminated by a close parenthesis. The definition of relation, string and number are stated in section 4.2. oID is the name of an object that is declared in the document, while vID is the name of a variable that belongs to oID. All conditions are ANDed together to determine the final result. Boolean OR is currently not available in SPCL.

4.11.5 Optional Side-Effects

The last option of a rule is side-effects. There are three possible structures of side-effect, namely meta-action, rule-update, and conditional-side-effect. When side-effect exists, semicolon is made optional. In addition, a semicolon is optional when it is followed by a closing brace '}'.

The reason for making semicolon optional is that a semicolon is merely served as a statement terminator and possesses no meaning. Thus when the structure itself is clear that the statement is terminating, a semicolon is made optional.

More concretely, the terminator of a rule is as follow:

```
terminator      : ';'
                | '{' ( side_effect ) * '}' (';')?
                | (next_token is '}') (';')?
                ;
side_effect     : meta_action
                | rule_update
                | conditional_side_effect
                ;
```

The side effects get triggered whenever the rule is being applied to a request. Each of the sub-components is to be discussed immediately in subsequent sections.

4.11.6 Meta Action

The idea of meta-action is to provide some kind of mechanism to inform certain party when unexpected events occurs. Currently, only log and notify are supported. The syntax is as follow:

meta_action : ("log" | "notify") pID ';' ;

where pID is the name of a principal that has been defined. In order for this function to operate appropriately, the information has to be provided in the principal's definition.

4.11.7 Rule Update

In the context of SPCL, updating a rule means introducing a new rule dynamically. These new rules would be placed at the principal precedence level (see section 4.12). The syntax of rule_update is just rule, i.e.:

rule_update : rule;

The right hand side, rule, could includes everything that has been discussed in section 4.11, or, to be more precise, the definition that is going to be presented at section 4.11.9.

4.11.8 Conditional Side Effects

A conditional side effect is just another flexible feature of SPCL. The administrator might decide to have certain side effects based on some conditions other than those that are already matched by the rule. The syntax is as follow:

```
conditional_side_effect : "if" condition_list '{'
                                    ( side_effect ) *
                          '}'
                          ( "else" '{' ( side_effect ) * '}' )?
                        ;
```

Condition_list has exactly the same format as in when-clause (section 4.11.4) except that the keyword "when" is missing. Side_effect has already been defined in section 4.11.5.

The syntax says a side effect could be composed of nested if-else blocks to determine the exact side effect that is applicable to a given situation.

4.11.9 Gathering All Elements Together

For readers' convenience, the complete definition of rule is duplicated here :

```
rule :    ("allow"|"deny")
          ( ( '*' ( "on" oID )? )
          | action_list "on" oID
          )
          ( by_clause ) ?
          ( "when" condition_list )?
          ( ';'
          | '{' ( side_effect )* '}' (';')?
          | (next token is '}') (';')? )
    ;

action_list : action ( ',' action )* ;
by_clause : ("by" pID(',' pID)* )? ;
condition_list :
              '('
                oID '.' vID relation value
                ( ',' oID '.' vID relation value ) *
              ')'
             ;


side_effect : ( ( "log" | "notify" ) pID ';' )
            | ( rule )
            | ( "if" condition_list '{'
                     ( side_effect ) *
              '}'
              ( "else" '{' ( side_effect ) * '}' )?
              )
            ;
```

4.12 Rule Conflicts and Precedence Levels

Rules defined in different blocks have different precedence levels. The following precedence are defined:

rule level	precedence
principal	20
group	10
default	0

Each rule is associated with a rule level, and its corresponding precedence value. When there is a conflict between two rules, the one with higher precedence value dominates.

If conflicts happens within the same level. The action requested would be rejected and a run-time error would be generated. However, if the conflict is foreseeable during compile time, a compiler-error shall be generated to stop ambiguity instead.

4.12.1 Principal Level

A rule is defined to be at principal level when the rule exists as a direct rule under a principal_definition discussed in section 4.9. In addition, if a rule is generated on demand (see section 4.11.7) and the target is a principal, it is also considered to be at principal level.

4.12.2 Group Level

A rule is defined to be at group level when the rule exists as a direct rule under a group_definition discussed in section 4.8. In addition, if a rule is generated on demand (see section 4.11.7) and the target is a group, it is also considered to be at group level.

4.12.3 Default Level

A rule is defined to be at default level when the rule exists in default_definition discussed in section 4.10.

4.13 Implicit declarations

Implicit declaration is not allowed in all aspects of SPCL. For example, whenever the administrator wants to define a rule on the usage of object A, A must be declared some where in the document. The same restriction applies to action, principal and group.

4.14 Regular Expressions

The regular expressions used in SPCL conform to JAVA's conventions. Please refer to JAVA API for further details on the exact syntax of regular expressions. They follow closely the Perl 5 regular expression syntax.

4.15 Syntax Summary

Please see the Grammar appendix.

5 Project Plan

This section deals with our project roadmap. We were fortunate enough to have a clear idea of our language from the beginning, so we were able to plan well.

5.1 Project Timeline

We formulated the idea of a policy language in late January and had a stable version of the whitepaper in early February.

In general, we followed a rough time of weekly meetings on Tuesdays at two-thirty in the afternoon. Depending on the subject matter, the meetings would last from fifteen minutes to an hour. We were able to complete the whitepaper fairly early. This accomplishment enabled use to complete the Language Reference Manual early enough to begin tackling the actual design and implemenation of the componenets in early to mid-March.

Most of the interface and policy engine were stable by late March.

We had one last push at the beginning of May to bring everyone's work together for a successful project. The Project Log section contains a detailed list of meetings and milestones.

5.2 Planning

Planning accounted for a large portion of our time. Our overall planning process and strategy was to:

1. secure the proposal of a policy language
2. perform research on policy languages
3. identify the major primitive types in our language
4. design and write our grammar early, so as to iterate over it and refine it
5. complete our LRM and interface so we had a stable template to begin construction
6. incorporate the design of an automated suite of tests from the beginning
7. identifying appropriate code freeze and feature freeze points
8. address material for the final paper in a timely fashion

5.3 Specification

Once we were confident that our planning had identified the appropriate areas of the project to work on, we divided the work according to the volunteered talents of the group members.

A requirements specification and design were the next tasks to tackle.

Our requirements specification is essentially detailed by the project whitepaper. We wanted a policy language that made easy the expression of a variety of common policy rules.

At this stage, we were also able to concentrate on specification of the language grammar and the interface between the compiler and the runtime environment. This

step was crucial, because it allowed us to proceed with a stable grammar in parallel implementation paths.

5.4 Development

Implementation of the project was a non-trivial task. In this section, we detail our approach to the development, including source code styling conventions and our development environment.

5.4.1 Programming Style Guide

Our overall programming style was fairly consistent. Surprisingly, all of our developers write Java in much the same way. Following are general requirements for Java and SPCL source code. Most of the requirements are simply to promote readability of the source and support maintainence operations.

1. All Java source code should be appropriately commented according to the JavaDoc specification. It is not necessary to JavaDoc simple get() or set() methods. Java source files should indicate their authors and date of last modification.
2. All code should be heavily internally documented and all non–obvious code snippits summerized somewhere within the method body or header.
3. Braces are allowed to appear on the same line as method declarations, control flow statements, and other such places.
4. For the initial non–production version of the application, liberal use of debugging statements is encouraged.
5. Classes should be cleanly and clearly defined. Functionality that is non–essential or relative should be moved to a new object.
6. The package architecture should be cleanly defined.
7. The test suite should rely on the diff operation, thus, tests should follow a consistent output metric.
8. Files should *NOT* be placed in the CVS if they do not compile or make without errors.
9. There is no requirement on OS or choice of editor.
10. The code must compile with the J2SDK v 1.4.1

5.4.2 Software Development Environment

Our development environments varied widely in terms of OS and editors. We standardized on Java 1.4 and the latest release of Antlr. We also employed a source code repository hosted by one of the team members to control all documents associated with the project. Operating systems include at least Linux, Solaris, and Windows2000, and team members employed each, even switching between them at various points in the process. Editors include at least Notepad, Textpad, JBuilder, Forte, x/emacs, and pico. Our source code control system was the standard Unix CVS, and compilation and test suites were controlled via Makefiles.

5.5 Testing

The Testing section provides detailed descriptions of our testing procedures. Our general test plan processes are described there.

The goal of the testing was to provide a very modular and automated approach towards unit testing as well as a standard suite of tests for the compiler and policy engine, as per the professor's suggestion.

5.6 Team Roles

Team implementation roles are described in the Architecture section, and not duplicated here to keep the reader's eyes open.

We were able to divide the project fairly evenly as well as enable everyone to have an influence on every part of the project. Every team member contributed to the design of the grammar and overall feel of the language.

Michael Locasto served as the project coordinator, but his job was considerably eased by the talent and determination of his team members. He wrote the initial whitepaper and redrafted it according to suggestions of the team and professor. He wrote an initial version of the grammar as well as implementing the PolicyEngine. In the latter stages of the project, he was responsible for coordinating with Matthew Burnside on the final system integration of the compiler, interface, and policy engine. Michael was responsible for generating the final report.

Chun Li revised the grammar and constructed most of the compiler proper, including the Lexer, Parser, TreeWalkers, and SymbolTable. Chun was responsible for writing the LRM. Chun also formatted the whitepaper and LRM into Latex for the final report. Chun also generated a number of test cases and contributed to the design of the language grammar, as well as writing the Compiler Architecture subsection of the final report.

Aron Wahl was responsible for recording meeting minutes, generating test cases, writing the language tutorial section and parts of the project plan section of the final project report, and assisting in the interface design and implementation.

Matthew Burnside took the initial version of the grammar and revised it. Matt was responsible for creating and maintaining the test suite system and the CVS. He worked closely with Michael to integrate the system components at the project's conclusion, revising the compiler, grammar, and code generator as needed to produce a valid version of our Java intermediate format. Matt was responsible for writing the Testing section of the final report.

In all, everyone contributed heavily to all parts of the project, and the coordinator would like to express his heartfelt thanks for everyone's hard work.

5.7 Project Log

Our project milestones follow:

System integration included making minor changes to the grammar, as well as the interface, compiler, and policy engine to make them interoperable. Because of our detailed planning, the changes were minimal and largly implementation-oriented, not conceptual.

Our CVS provides a detailed list of check-ins of the source code and related documentation. The highlights of this log are included after the milestones list.

5.7.1 Milestones

Language Concept:	January 24, 2003
Draft of whitepaper:	January 28, 2003
First meeting:	January 28th, 2003
First draft of grammar:	January 30th, 2003
Second draft of grammar:	February 3rd, 2003
Second meeting:	February 4th, 2003
Outline of Tutorial:	February 4th, 2003
Approved Draft of whitepaper:	February 7th, 2003
Logical design of PolicyEngine:	February 8th, 2003
Third meeting:	February 11th, 2003
Initial version of grammar:	February 14, 2003
Grammar revision:	February 18th, 2003
Whitepaper submitted:	February 18th, 2003
Fourth meeting:	February 18th, 2003
LRM discussed, grammar haggled over:	February 18th, 2003
Fifth meeting:	February 25, 2003
Began compiler construction:	March 25th, 2003
Final LRM version submitted:	March 25th, 2003
Runtime Architecture Diagram Completed:	March 24th, 2003
Runtime Code version 1:	March 18th–April 11th, 2003
Interface Features frozen:	March 30th, 2003
Portions of the Compiler finished:	April 10th, 2003
Began Final Report:	May 5th, 2003
Integrated system:	May 5th–May 9th, 2003
Final Report and Project Presentation finished:	May 11th, 2003

5.7.2 CVS Project Log Highlights

```
February 11th, 2003:
Update of /home/mb/dat/cvs/w4115/final

Modified Files:
grammar.g
Log Message:
Added the new rule syntax.

February 26th, 2003:
CVS moved locations, testing functionality added

March 10th, 2003:
Update of /home/mb/cvs//class/w4115/final/doc

Modified Files:
Langauge Reference Manual.html
Added Files:
Language Reference Manual.pdf Language Reference Manual.ps
Log Message:
Lauguage Reference Manual converted to pdf format.
```

March 22, 2003:
Update of /home/mb/cvs//class/w4115/final

Modified Files:
Makefile TestGrammar.java grammar.g tree.g
Added Files:
CodeGenerator.g SymbolTable.java SymbolTableBuilder.g
TypeChecker.g VariableConflictException.java
VariableNotFoundException.java
Log Message:
.) Added a few more test cases.
.) non-determinism in lexer is elimiated.
.) Initial Design of Compiler is ready. Currently, the design devides the compiler into the following stages:

LexerSPCL --> Tokens
ParserTokens --> AST
Building SymbolTable AST --> Symbol Table
Type Checking(AST,Symbol Table) --> (no)error
Code Generation(AST,Symbol Table) --> Java Code

The first two part was done previously.

The following files are added for creating Symbol Table:
SymbolTable.java
VariableNotFoundException.java
VariableConflictException.java
The following files are added for building the last 3 stages:
SymbolTableBuilder.g
TypeChecker.g
CodeGenerator.g

March 22, 2003:
Update of /home/mb/cvs//class/w4115/final/tests

Added Files:
example10.txt example11.txt example12.txt example13.txt
example14.txt example9.txt
Log Message:
.) Added a few more test cases.

March 23, 2003:
Update of /home/mb/cvs//class/w4115/final/doc

Modified Files:
Language Reference Manual.pdf
Log Message:
LRM updated

April 10th, 2003:
Update of /home/mb/cvs//class/w4115/final/java_file

```
Update of /home/mb/cvs//class/w4115/final/tests
Update of /home/mb/cvs//class/w4115/final/g_file

Modified Files:
ErrorHandler.java SymbolTable.java
Added Files:
InvalidActionException.java NoRegexDefinedException.java

Added Files:
example23.txt example24.txt example25.txt example26.txt
example27.txt example28.txt

Modified Files:
CodeGenerator.g SymbolTableBuilder.g TypeChecker.g

Update of /home/mb/cvs/class/w4115/final/java_file

Modified Files:
ErrorHandler.java SymbolTable.java TestGrammar.java
TypeMisMatchException.java
Log Message:
Wooo, getting close!

May 10, 2003:
Update of /home/mb/cvs/class/w4115/final/spcl/docs/paper

Added Files:
architecture.tex computationModel.tex contents.tex grammar.tex
intro.tex lessons.tex main.tex projectplan.tex refmanual.tex
source.tex testing.tex tutorial.tex
Log Message:
the final report files

Update of /home/mb/cvs/class/w4115/final/spcl/src/spcl/data

Modified Files:
Action.java DefaultPolicyInstaller.java Group.java Policy.java
PolicyException.java PolicyLoader.java PolicyRequest.java
Principal.java Rule.java SystemObject.java Variable.java
Zone.java
Added Files:
Condition.java PolicyContext.java Test1PolicyInstaller.java
thisZonemyPolicyInstaller.java
Log Message:
The interface API files: Rule.java is interesting, so is Condition


Update of /home/mb/cvs/class/w4115/final/spcl/src/spcl/policyengine

Modified Files:
PolicyEngine.java
```

```
Log Message:
The PolicyEngine proper, with working resolution, unification and blah
algorithms


Update of /home/mb/cvs/class/w4115/final/spcl/docs/paper

Modified Files:
pe_arch.eps
Added Files:
pc_arch.eps
Log Message:
updated arch pictures

Update of /home/mb/cvs/class/w4115/final/tests

Modified Files:
DuplicateNamedSLO2.result WithObject.result
Added Files:
TutorialProblem.result TutorialProblem.spcl
Removed Files:
example1.txt example10.txt example11.txt example12.txt
example13.txt example14.txt example15.txt example16.txt
example17.txt example18.txt example19.txt example2.txt
example20.txt example21.txt example22.txt example23.txt
example24.txt example25.txt example26.txt example27.txt
example28.txt example3.txt example4.txt example5.txt
example6.txt example7.txt example8.txt example9.txt
parserError.bat symBuilderError.bat typeError.bat
Log Message:
Moved all the example files into a tests subdirectory
```

6 Application Architecture

This section details the compiler and runtime environment architectures.

6.1 Implementation Roles

The entire team participated in the development of the system design and language grammar.

Michael Locasto was primarily responsible for constructing the PolicyEngine, PolicyClient, and core interface objects. Aron Wahl was responsible for defining some of the core interface objects and providing some test cases. Matthew Burnside was responsible for creating the initial grammar file, portions of the interface objects, most of the test suite functionality, and various operations on the compiler. Chun Li was primarily responsible for constructing the compiler components, as well as some test cases. Every team member worked extremely hard, and the talents of each should be recognized.

6.2 SPCL API

There is a set of core objects that define the interface for the compiler and the SPCL interpreter. This object interface is currently implemented in Java, but it is language-independent.

This section is a detailed discussion of the purpose and functions of the objects. Specific implementation detail is available in the attached source files.

The Java package `spcl.data` represents the interface between the Compiler and the PolicyEngine. This core set of objects essentially define the data types for SPCL's model of computation, much like registers define the basic data type for a microprocessor.

Each major object in the SPCL grammar syntax has a reflection in this package that provides a semantic context for the object: e.g., the operations that can be performed on the object and its relationships to other objects.

The objects in `spcl.data` can be grouped into three categories:

1. Basic Objects - direct reflections of the language primitive types
   (a) Zone
   (b) Policy
   (c) Principal
   (d) Group
   (e) SystemObject
   (f) Rule
   (g) Variable
   (h) Action
2. Context Objects - more abstract implied objects that do not contain a direct reflection in the language grammar, but are implied by the language functionality
   (a) PolicyContext
   (b) Condition

(c) Firing

3. Functional Objects - service objects for the system

   (a) PolicyException

   (b) PolicyInstaller

   (c) PolicyLoader

   (d) PolicyRequest

   (e) PolicyResponse

6.2.1 Basic Objects

Of the Basic Objects, the Rule and Variable objects are the most interesting. The other objects more or less contain functionality for maintaining the appropriate relationship data structures.

The Variable object is a flexible container for a triple-typed object. It can represent a string, a signed integer, or a boolean value, or all three at once. This capability is a bonus for providing a flexible type system for Rule conditions.

The Rule object, in addition to basic data structure operations, contains powerful functionality for evaluating itself. Part of this functionality is the ability to fire side effects. This side effect ability is discussed later. Rules contain an `evaluate()` method that returns a decision given the current object, the state of that object, and an action string. The Rule and the PolicyEngine have a close relationship during the evaluation and matching of rules. The `evaluate()` method is called from the PolicyEngine's query evaluator with the appropriate meta-information, and the Rule proceeds to check that the provided parameters unify with its predefined Condition variables. All conditions must unify and then evaluate to true. If the logical AND of all the conditions is not satisfied, then the Rule does not match. After condition unification, the Rule then checks if the provided action string is a legal incarnation of the Rule's regular expression. Finally, the Rule returns an *allow*, *deny*, or *wishy–washy* decision.

The *allow* and *deny* decisions recommend the obvious result, but the *wishy-washy* decision allows lower precedence levels a chance to evaluate the request.

6.2.2 Context Objects

The Context Objects are implied functionality that do not clearly map from a grammar rule to a Basic Object.

The simplest Context Object to understand is also the most useful. Much like the "`current`" macro in the Linux kernel, which keeps track of the process that currently owns the CPU, the PolicyContext keeps track of a triple: the current zone, the current policy in that zone, and the current client object, if any.

The Firing object is also simple in definition, but it is subtle in execution. The Firing object is essentially a placeholder. It defines the `doFire()` method, which enables Rules to have side effects. The generated code creates a number of anonymous inner classes that correspond to Firings of Rules. Each Rule, when matched, regardless of its result, will invoke all its Firings by calling their `doFire()` method, which is generated by the compiler and overrides the default definition of `Firing.doFire()`

The powerful ability to have side effects makes for some interesting relationships between Rules, Principals, Groups, and SystemObjects. For example, two matched

Rules could constantly set each other active or inactive, dynamically changing the installed Policy based on runtime conditions.

The Condition object is not terribly complicated, but is worthy of note. A Condition represents a boolean valued comparison. It holds a Variable, a relation operator, and an rvalue, which is a value on the right side of the boolean expression. Its `eval` method will evaluate its data fields when necessary and return a boolean decision. The key idea here is that a Condition is unified during every search, so its Variable value, and thus its overall evaluation, may change depending on runtime conditions.

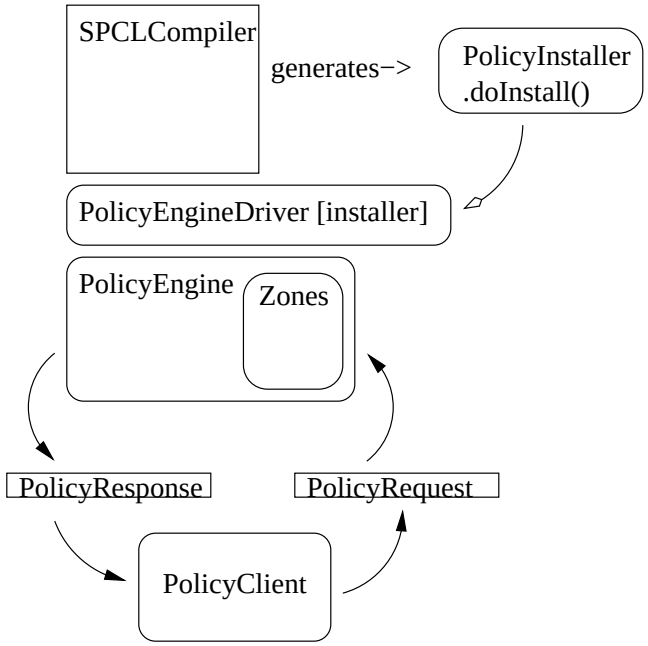
6.2.3 Functional Objects

The Functional Objects in this package provide high level interface functionality to simplify the job of the compiler. The PolicyException class encapsulates some higher level exception and error information and the PolicyRequest and PolicyResponse objects define an interface for a PolicyClient to talk to the PolicyEngine over a network socket (or a telegraph machine, for that matter).

The PolicyLoader and PolicyInstaller interface and class, respectively, combine to enable the PolicyEngine an easy way to load the class definitions produced by the compiler. The compiler turns each SPCL source file into a subclass of PolicyInstaller, whose `doInstall()` method is filled with the necessary information to create the relationships between all the Policy objects. In addition, the PolicyInstaller has a reference to a PolicyLoader, which is an interface that the PolicyEngine implements. The PolicyLoader interface contains methods for managing the policy lifecycle as well as evaluating some conditions dealing with the system itself (such as the system time and date).

6.2.4 Policy Engine and Runtime Architecture

The PolicyEngine is the core of SPCL's runtime environment.



The PolicyEngine is the most complicated piece of software in the system besides the compiler, and is assisted by the methods of the Rule object.

The current implementation of the PolicyEngine creates a server application that listens on two sockets. One socket is for a command stream for the PolicyEngine software itself. The other socket is dedicated to servicing PolicyRequests from system objects.

The PolicyEngine is bootstrapped with the definition of a PolicyInstaller class that represents the source code specified in a SPCL source file.

The PolicyEngine then uses its own extension to the Java ClassLoader functionality to load in and define the supplied PolicyInstaller. The PolicyInstaller then loads into a Zone in the PolicyEngine.

The PolicyEngine then waits for requests against the policy and answers the queries. The PolicyEngine itself must perform some mini–lexing, mini–parsing, and mini–type checking of its own to make sure that queries are of the appropriate form. The `doQuery()` method provides the implementation details.

The PolicyEngine then proceeds to run the model of computation algorithm, identifying the appropriate principals, objects, groups, and information to satisfy the query, unifying the conditions, and proceeding to use the Rules to evaluate through the precedence levels.

6.3 Compiler Architecture

The Compiler could be viewed as a black box that generates JAVA code from SPCL program, as depicted from the figure. Within this black box, there are roughly 5 different stages.

1. Lexer
2. Parser
3. Symbol Table Builder
4. Type Checker
5. Code Generator

6.3.1 Lexer

The lexer is responsible for stripping inputs into tokens of predefined format. This component as well as the Parser (discussed next), uses Antlr's default error handling routine.

6.3.2 Parser

The parser is responsible for generating an internal representation (AST) of the complete SPCL program.

6.3.3 Symbol Table Builder

This component uses Antlr's tree walking facility to generate a tree walker. This tree walker is responsible for building a symbol table for the corresponding AST.

A symbol table stores information, such as ID, type, and scope about all variables. Each table entry uses a homogenous infrastructure. The type is the only way to tell whether certain piece of information is relevant to a given variable.

If something is wrong during this stage (e.g. variable is redeclared) it will report the error to an error handling routine. At the end of the tree walking, it will consult the error handling routine and see if any error has been reported. If that is the case, the compiler would refrain from going to the next stage.

6.3.4 Type Checker

This is another tree walker that walks the AST the same way as in the previous stage. The only difference is that it doesn't do anything but generate Exceptions. To be more precise, for every place a reference to any variable is found, the actual variable type is checked against the expected type. If something is wrong during this checking, an error would be reported to the error handling routine.

This component is separated from the previous stage for 2 reasons:

1. make the code more modular
2. SPCL is a language that inherently permits forward reference, thus type checking cannot be done in one pass of the AST.

Similar to the previous stage, if anything is wrong, the compiler would refrain from going to the next stage.

6.3.5 Code Generator

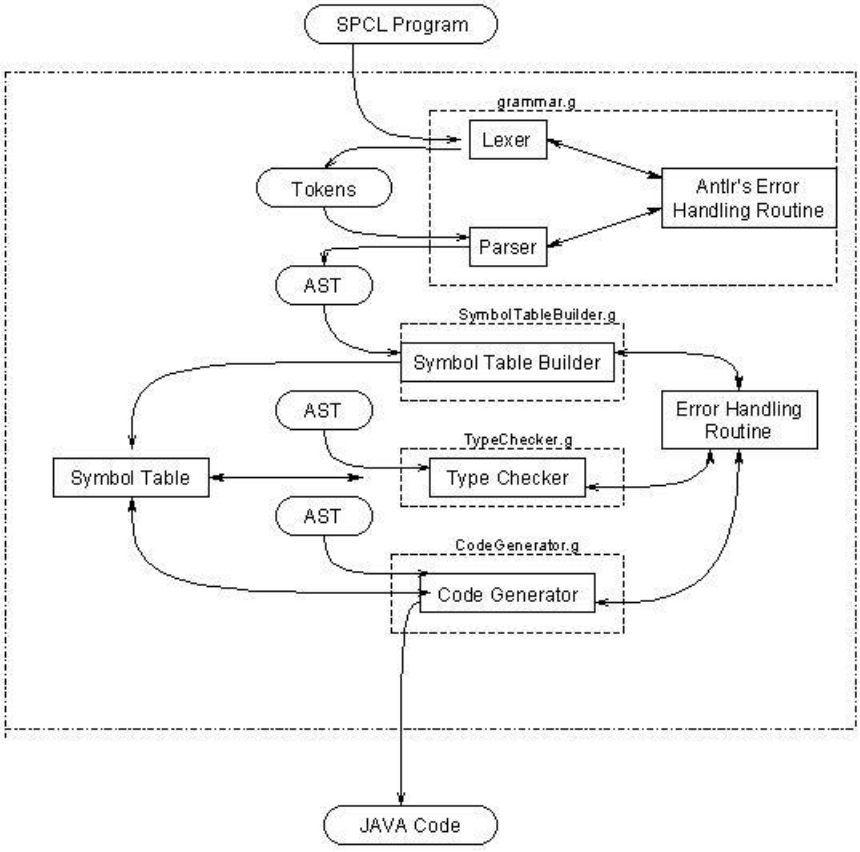
The role of this component is to generate JAVA codes that would create and install the policy specified by the SPCL source program. This is the last stage of the compiler. When this point is reached, the code is assumed to contain no error. The connection with the error handling routine is just a redundant safety feature that ensures all previous stages are doing the right jobs.

It may seem possible to combine this stage with the previous one. However, they are separated from each other for 2 reasons:

1. make the code more modular
2. If something is wrong in the source program, it really doesn't make sence to generate partial JAVA codes. Thus, before generating anything, the code has to be checked.

If something does go wrong in this stage, the compiler should stop immediately.

The following figure depicts the major architectural highlights and workflow of the compiler itself.


SPCL Program
grammar.g
Lexer
Tokens
Antlr's Error Handling Routine
Parser
AST
SymbolTableBuilder.g
Symbol Table Builder
AST
Error Handling Routine
TypeChecker.g
Symbol Table
Type Checker
AST
CodeGenerator.g
Code Generator
JAVA Code

```
$(test_out)/%Installer.java : clean_test
       $(java) $(jflags) SPCLCompiler \
            $(addsuffix .spcl, $(patsubst %, \
            $(test_dir)/%, $*)) >& $(addsuffix .error, \
            $(patsubst %, $(test_out)/%, $*))
        mv $(test_dir)/*.java $(test_out)
        diff  $(addsuffix Installer.java, \
              $(patsubst %, $(test_out)/%, $*))  \
              $(addsuffix .result, $(patsubst %, \
                                  $(test_dir)/%, $*))
```

Figure 1: Makefile rule for applying the test suite

7 Test Plan

Testing the SPCL compiler is in some ways unique, because we are designed a language, a compiler for that language, and the platform on which the compiled language will run. We broke testing down into a hierarchy; starting with testing each stage of the compiler and policy engine individually, followed by testing the full compiler and full policy engine individually, and then testing the compiler and policy engine together.

Testing the policy engine is performed primarily by hand, since requests must be generated dynamically at runtime. Testing on the compiler is performed through use of a Makefile script that runs the compiler on suite of test inputs and compares each test output to a predetermined output. As you can see if Figure 7, the Makefile runs the compiler, then does a simple `diff` to compare the compiler output to the predetermined output. If the `diff` fails, the Makefile will halt. Figure 7 contains a selection of tests that we've run on the compiler.

In addition, below you can find the SPCL source code and intermediate format Java code for several test cases.

One test case, EmptyAllow is a simple test on the simplest possible rule base. The SPCL source is:

```
zone Empty;

policy Allow {
        default {
                allow *;
        }
}
```

EmptyAllow compiles to the following:

```
import spcl.data.*;
import java.util.*;
import java.io.*;

public class EmptyAllowInstaller extends PolicyInstaller {

        public EmptyAllowInstaller( ) {
        }

        public void doInstall() throws PolicyException {
                if (null==policyLoader)
                        throw new PolicyException("No policy loader defined.",
                          PolicyException.NULL_LOADER);
                final Zone zone = new Zone( "Empty" );
                final Policy policy = new Policy( "Empty.Allow" );

                final Principal default_p = new Principal( "default_p" );
                policy.setDefaultPrincipal( default_p );

                final Rule Empty_Allow_default_p_1 = new Rule( "Empty.Allow.default_p.1" );
                Empty_Allow_default_p_1.setActive( true );
                Empty_Allow_default_p_1.setAccess( spcl.data.Rule.ALLOW );
```

```
                Empty_Allow_default_p_1.setAction( spcl.data.Rule.STAR );
                Empty_Allow_default_p_1.setObject( spcl.data.Rule.STAR_OBJECT );
                Empty_Allow_default_p_1.setOwner( default_p );
                default_p.addRule( Empty_Allow_default_p_1 );

                zone.attach( policy );
                policyLoader.load( zone );
        }
}
```

A more interesting test case is the CyclicReference test case. In this test, we have two rules, and when either one fires, it activates a rule referencing the principal of the other rule.

```
zone Cyclic;

policy Reference {

        default {
                deny *;
        }

        object o {
               boolean t = true;
               boolean x = true;
               actions {
                             action z = "blah";
               }
        }

        principal A {
                allow z on o when (o.x == true) {
                       if (o.t == true) {
                          deny z on o by B;
                       }
                }
        }

        principal B {
                allow z on o when (o.x == true) {
                       if (o.t == true) {
                          deny z on o by A;
                       }
                }
        }

}
```

Note that when CyclicReference is compiled, the setOwner() calls are delayed to allow the cyclic references to succeed.

```
import spcl.data.*;
import java.util.*;
import java.io.*;

public class CyclicReferenceInstaller extends PolicyInstaller {

        public CyclicReferenceInstaller( ) {
        }

        public void doInstall() throws PolicyException {
                if (null==policyLoader)
                        throw new PolicyException("No policy loader defined.",
                          PolicyException.NULL_LOADER);
                final Zone zone = new Zone( "Cyclic" );
                final Policy policy = new Policy( "Cyclic.Reference" );

                final Principal default_p = new Principal( "default_p" );
                policy.setDefaultPrincipal( default_p );

                final Rule Cyclic_Reference_default_p_1 = new Rule( "Cyclic.Reference.default_p.1" );
                Cyclic_Reference_default_p_1.setActive( true );
                Cyclic_Reference_default_p_1.setAccess( spcl.data.Rule.DENY );
                Cyclic_Reference_default_p_1.setAction( spcl.data.Rule.STAR );
                Cyclic_Reference_default_p_1.setObject( spcl.data.Rule.STAR_OBJECT );
                Cyclic_Reference_default_p_1.setOwner( default_p );
                default_p.addRule( Cyclic_Reference_default_p_1 );

                final SystemObject o = new SystemObject( "o" );
                policy.addObject( o);
                final Variable Cyclic_Reference_o_t = new Variable("t", Variable.BOOLEAN, "true");
                o.addVariable(Cyclic_Reference_o_t);
                final Variable Cyclic_Reference_o_x = new Variable("x", Variable.BOOLEAN, "true");
                o.addVariable(Cyclic_Reference_o_x);
                final Action Cyclic_Reference_o_z = new Action( "z" );
                Cyclic_Reference_o_z.setActionPattern( "blah");
                o.addAction(Cyclic_Reference_o_z);

                final Principal A = new Principal( "A" );
                policy.addPrincipal( A );
```

```
                final Rule Cyclic_Reference_A_3 = new Rule( "Cyclic.Reference.A.3" );
                Cyclic_Reference_A_3.setActive( true );
                Cyclic_Reference_A_3.setAccess( spcl.data.Rule.ALLOW );
                Cyclic_Reference_A_3.setAction( Cyclic_Reference_o_z );
                Cyclic_Reference_A_3.setObject( o );
                Cyclic_Reference_A_3.setOwner( A );
                A.addRule( Cyclic_Reference_A_3 );
                Cyclic_Reference_A_3.addCondition(Cyclic_Reference_o_x,"==", true);

                final Rule Cyclic_Reference_A_3_2 = new Rule( "Cyclic.Reference.A.3.2" );
                Cyclic_Reference_A_3_2.setActive( false );
                Cyclic_Reference_A_3_2.setAccess( spcl.data.Rule.DENY );
                Cyclic_Reference_A_3_2.setAction( Cyclic_Reference_o_z );
                Cyclic_Reference_A_3_2.setObject( o );

                Cyclic_Reference_A_3.addFiring(
                        new Firing(){
                                public void doFire() {
                                        if (policyLoader.resolveCondition( Cyclic_Reference_o_t,"==", true)) {
                                                Cyclic_Reference_A_3_2.setActive( true );
                                        }
                                }
                        }
                );

                final Principal B = new Principal( "B" );
                policy.addPrincipal( B );
                Cyclic_Reference_A_3_2.setOwner( B );
                B.addRule( Cyclic_Reference_A_3_2 );

                final Rule Cyclic_Reference_B_5 = new Rule( "Cyclic.Reference.B.5" );
                Cyclic_Reference_B_5.setActive( true );
                Cyclic_Reference_B_5.setAccess( spcl.data.Rule.ALLOW );
                Cyclic_Reference_B_5.setAction( Cyclic_Reference_o_z );
                Cyclic_Reference_B_5.setObject( o );
                Cyclic_Reference_B_5.setOwner( B );
                B.addRule( Cyclic_Reference_B_5 );
                Cyclic_Reference_B_5.addCondition(Cyclic_Reference_o_x,"==", true);

                final Rule Cyclic_Reference_B_5_4 = new Rule( "Cyclic.Reference.B.5.4" );
                Cyclic_Reference_B_5_4.setActive( false );
                Cyclic_Reference_B_5_4.setAccess( spcl.data.Rule.DENY );
                Cyclic_Reference_B_5_4.setAction( Cyclic_Reference_o_z );
                Cyclic_Reference_B_5_4.setObject( o );
                Cyclic_Reference_B_5_4.setOwner( A );
                A.addRule( Cyclic_Reference_B_5_4 );

                Cyclic_Reference_B_5.addFiring(
                        new Firing(){
                                public void doFire() {
                                        if (policyLoader.resolveCondition( Cyclic_Reference_o_t,"==", true)) {
                                                Cyclic_Reference_B_5_4.setActive( true );
                                        }
                                }
                        }
                );

                zone.attach( policy );
                policyLoader.load( zone );
        }
}
```

Finally, a test which hits on a number of different points at once; ManyTests.

```
zone Many;

policy Tests{

        default{
                deny *;
                allow rule1
                on fridge
                by Ken
                when (system.time=="5:00pm")
                { /* notify Mom; */ }
        }

        object fridge{
                url = "appliance://FF::AB:34:22/kenmore1/";
                actions {
                        action rule1 = "ls [a..z]*";
                }

        }

        object webserver{
                url = "http://www.thesmiths.home:80/";
                actions{
                        action rule1 = "GET [a..z]*.html HTTP/1.[1|0]";
                }
        }

        group parents{
                allow * on *;
        }

        group children{
```

```
                deny * on *;
        }

        principal Mom{
                alias = wonderWoman, ADMIN;
                url = "ldap://192.168.4.240/mom/keys/pub.key";
                member = parents,children;
        }

        principal Dad{
                alias = father;
                member = parents;
                allow *;
        }

        principal Ken{
                member = children;
                allow rule1
                        on webserver
                        when (system.time > "16:00")
                        { /* notify Dad; */  }
                allow rule1 on fridge
                        { if( system.time > "14:00" ) {
                                /* notify Dad; */
                          } else {
                                /* notify Mom; */
                          }
                        };

        }


}
```

Note the seemless handling of the regular expressions.

```
import spcl.data.*;
import java.util.*;
import java.io.*;

public class ManyTestsInstaller extends PolicyInstaller {

        public ManyTestsInstaller( ) {
        }

        public void doInstall() throws PolicyException {
                if (null==policyLoader)
                        throw new PolicyException("No policy loader defined.",
                          PolicyException.NULL_LOADER);
                final Zone zone = new Zone( "Many" );
                final Policy policy = new Policy( "Many.Tests" );

                final Principal default_p = new Principal( "default_p" );
                policy.setDefaultPrincipal( default_p );

                final Rule Many_Tests_default_p_1 = new Rule( "Many.Tests.default_p.1" );
                Many_Tests_default_p_1.setActive( true );
                Many_Tests_default_p_1.setAccess( spcl.data.Rule.DENY );
                Many_Tests_default_p_1.setAction( spcl.data.Rule.STAR );
                Many_Tests_default_p_1.setObject( spcl.data.Rule.STAR_OBJECT );
                Many_Tests_default_p_1.setOwner( default_p );
                default_p.addRule( Many_Tests_default_p_1 );

                final Rule Many_Tests_default_p_2 = new Rule( "Many.Tests.default_p.2" );
                Many_Tests_default_p_2.setActive( true );
                Many_Tests_default_p_2.setAccess( spcl.data.Rule.ALLOW );
                Many_Tests_default_p_2.addCondition(spcl.data.PolicyLoader.SYSTEM_TIME, "==", "5:00pm");

                final SystemObject fridge = new SystemObject( "fridge" );
                policy.addObject( fridge);
                final Action Many_Tests_fridge_rule1 = new Action( "rule1" );
                Many_Tests_fridge_rule1.setActionPattern( "ls [a..z]*");
                fridge.addAction(Many_Tests_fridge_rule1);
                Many_Tests_default_p_2.setAction( Many_Tests_fridge_rule1 );
                Many_Tests_default_p_2.setObject( fridge );

                final SystemObject webserver = new SystemObject( "webserver" );
                policy.addObject( webserver);
                final Action Many_Tests_webserver_rule1 = new Action( "rule1" );
                Many_Tests_webserver_rule1.setActionPattern( "GET [a..z]*.html HTTP/1.[1|0]");
                webserver.addAction(Many_Tests_webserver_rule1);

                final Group parents = new Group( "parents" );
                policy.addGroup( parents );

                final Rule Many_Tests_parents_3 = new Rule( "Many.Tests.parents.3" );
                Many_Tests_parents_3.setActive( true );
                Many_Tests_parents_3.setAccess( spcl.data.Rule.ALLOW );
                Many_Tests_parents_3.setAction( spcl.data.Rule.STAR );
                Many_Tests_parents_3.setObject( spcl.data.Rule.STAR_OBJECT );
                Many_Tests_parents_3.setOwner( parents );
                parents.addRule( Many_Tests_parents_3 );

                final Group children = new Group( "children" );
                policy.addGroup( children );

                final Rule Many_Tests_children_4 = new Rule( "Many.Tests.children.4" );
```

```
                Many_Tests_children_4.setActive( true );
                Many_Tests_children_4.setAccess( spcl.data.Rule.DENY );
                Many_Tests_children_4.setAction( spcl.data.Rule.STAR );
                Many_Tests_children_4.setObject( spcl.data.Rule.STAR_OBJECT );
                Many_Tests_children_4.setOwner( children );
                children.addRule( Many_Tests_children_4 );

                final Principal Mom = new Principal( "Mom" );
                policy.addPrincipal( Mom );
                parents.addMember(Mom);
                children.addMember(Mom);

                final Principal Dad = new Principal( "Dad" );
                policy.addPrincipal( Dad );
                parents.addMember(Dad);

                final Rule Many_Tests_Dad_5 = new Rule( "Many.Tests.Dad.5" );
                Many_Tests_Dad_5.setActive( true );
                Many_Tests_Dad_5.setAccess( spcl.data.Rule.ALLOW );
                Many_Tests_Dad_5.setAction( spcl.data.Rule.STAR );
                Many_Tests_Dad_5.setObject( spcl.data.Rule.STAR_OBJECT );
                Many_Tests_Dad_5.setOwner( Dad );
                Dad.addRule( Many_Tests_Dad_5 );

                final Principal Ken = new Principal( "Ken" );
                policy.addPrincipal( Ken );
                Many_Tests_default_p_2.setOwner( Ken );
                Ken.addRule( Many_Tests_default_p_2 );
                children.addMember(Ken);

                final Rule Many_Tests_Ken_6 = new Rule( "Many.Tests.Ken.6" );
                Many_Tests_Ken_6.setActive( true );
                Many_Tests_Ken_6.setAccess( spcl.data.Rule.ALLOW );
                Many_Tests_Ken_6.setAction( Many_Tests_webserver_rule1 );
                Many_Tests_Ken_6.setObject( webserver );
                Many_Tests_Ken_6.setOwner( Ken );
                Ken.addRule( Many_Tests_Ken_6 );
                Many_Tests_Ken_6.addCondition(spcl.data.PolicyLoader.SYSTEM_TIME, ">", "16:00");

                final Rule Many_Tests_Ken_7 = new Rule( "Many.Tests.Ken.7" );
                Many_Tests_Ken_7.setActive( true );
                Many_Tests_Ken_7.setAccess( spcl.data.Rule.ALLOW );
                Many_Tests_Ken_7.setAction( Many_Tests_fridge_rule1 );
                Many_Tests_Ken_7.setObject( fridge );
                Many_Tests_Ken_7.setOwner( Ken );
                Ken.addRule( Many_Tests_Ken_7 );

                zone.attach( policy );
                policyLoader.load( zone );
        }
}
```

8 Grammar

The SPCL Grammar.

```
policydef
: "zone"^ ID SEMICOLON! policy ;

policy
: "policy"^ ID LCURLY!
  default_def
  (object_def)*
  (group_def)*
  (principal_def)*
RCURLY! ;

default_def:
"default"^ LCURLY!
  (rule)*
RCURLY!
;

group_def:
"group"^ ID LCURLY!
   (rule)*
RCURLY!
;

principal_def:
"principal"^ ID LCURLY!
( alias )?
( url )?
( email )?
( phone )?
( member )?
( rule )*
RCURLY!
;

alias: "alias"^ ASSIGN! ID (COMMA! ID)* SEMICOLON! ;
email: "email"^ ASSIGN! STRING SEMICOLON! ;
phone: "phone"^ ASSIGN! STRING SEMICOLON! ;
member: "member"^ ASSIGN! ID ( COMMA! ID )* SEMICOLON! ;
url: "url"^ ASSIGN! STRING SEMICOLON! ;

object_def:
"object"^ ID LCURLY!
( url )?
( variable_declaration )*
( actions )?
```

```
RCURLY!
;

variable_declaration
: ( num_def )
| ( string_def )
| ( bool_def )
;
num_def: "number"^ ID (ASSIGN! NUMBER)? SEMICOLON!;
string_def: "string"^ ID (ASSIGN! STRING)? SEMICOLON! ;
bool_def: "boolean"^ ID (ASSIGN! ("true"|"false") )? SEMICOLON! ;

actions: "actions"! LCURLY! (single_action)+ RCURLY! ;
single_action
: "action"^ ID ASSIGN! STRING (COMMA! STRING)* SEMICOLON!
  ( default_meta_action )?
;
default_meta_action
: ID DOT! "meta_action"^ ASSIGN! ( "log" | "notify" ) ID SEMICOLON!
;

rule: ("allow"|"deny")
( ( STAR ( "on"! (ID|STAR) ) ? )
| (action_list "on"! ID )
)
( by_clause )?
( when_clause )?
( {LA(1)==RCURLY}? /*last ';' optional */
| SEMICOLON!
| (LCURLY! (side_effect)* RCURLY! (SEMICOLON!)? ))
{ String id=this.newID(); #rule = #([RULE_ID, id], #rule); }
;

side_effect:
( meta_action
| rule
| ( "if"^ condition_list LCURLY!
      (side_effect)*
    RCURLY! ("else" LCURLY!
      (side_effect)*
    RCURLY! )?
  )
);

action_list : ID ;
by_clause : "by" ID (COMMA! ID)* ;
when_clause : "when" condition_list;
meta_action: ("log"^|"notify"^) ID SEMICOLON! ;
condition_list : LPAREN! cond (COMMA! cond)* (COMMA!)? RPAREN! ;
```

```
cond : ID DOT! ID RELATION^ value ;

value : STRING | NUMBER | "true" | "false" ;

LPAREN : '(';
RPAREN : ')';
LCURLY : '{';
RCURLY : '}';
COMMA : ',';
SEMICOLON : ';';
ASSIGN : { LA(2)!='=' }? '=';
RELATION : ">" | "<" | "<=" | ">=" | "==" | "!=" ;

STAR : '*' ;
ID options { testLiterals = true; }
: ('a'..'z'|'A'..'Z') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*;
STRING
: '"'!
  ( ~('"'|'\n') ) *
  '"'!
;

NUMBER
: ( "+"|"-" ) ?
( (INT ( "." (INT)?)? )
| ( "." {$setType(DOT);} (INT {$setType(NUMBER);} )? )
)
;

// Whitespace -- ignored
WS : ( ' '
| '\t'
| '\f'
// handle newlines
| ( options {generateAmbigWarnings=false;}
: "\r\n"  // DOS
| '\r'    // Macintosh
| '\n'    // Unix
)
{ newline(); }
)+
{ _ttype = Token.SKIP; }
;

// Single-line comments
SL_COMMENT
: "//"
(~('\n'|'\r'))* ('\n'|'\r'('\n')?)
{$setType(Token.SKIP); newline();}
;
```

```
// multiple-line comments
ML_COMMENT
: "/*"
( /* This specification is ambiguous for OS
   that allow all of '\r', "\r\n" and '\n'
*/
options {generateAmbigWarnings=false;}
:
{ LA(2)!='/' }? '*'
| '\r' '\n'{newline();}
| '\r'{newline();}
| '\n'{newline();}
| ~('*'|'\n'|'\r')
)*
"*/"
{$setType(Token.SKIP);}
;

protected
INT : ('0'..'9')+;
```

9 Source Code Listing

Below is the entire source listing of both the Compiler and the PolicyEngine, as well as the core interface classes and the PolicyClient.

All source code is available from the CVS or from the project website at

`http://www.cs.columbia.edu/~locasto/projects/spcl/releases/`

That code may be easier to read.

```
/*********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: March 27, 2003
*
```

```
**********************************************/

package spcl.client;

import spcl.data.*;
import java.net.*;
import java.io.*;
import java.util.*;

/**
 * A basic utility for communicating with
 * the policyengine. Starts up an
 * interactive command line interpreter
 *
 * @author locasto@cs.columbia.edu
 *
*/
public class PolicyEngineClient{

private Socket client;
private ObjectInputStream oin;
private ObjectOutputStream oout;
private InputStreamReader isr;
private BufferedReader keyboard;


public PolicyEngineClient() throws Exception{
//prompt user for 'connect ip port'

String command = "";
Vector args = new Vector( 7 );
isr = new InputStreamReader(System.in);
keyboard = new BufferedReader(isr);
System.out.print( "\npec> " );
command = keyboard.readLine();
command = command.trim();
if( command==null || command.equals( "" ) || command.equalsIgnoreCase( "bye" ) ||
System.out.println( "PolicyEngine client exiting..." );
System.exit( 0 );
}else if( command.equalsIgnoreCase( "help" ) ){
System.out.println( "\nHELP:\n\tbye,exit,quit,help,connect [host] [port]\n" );
}else if( command.startsWith( "connect " ) ){
StringTokenizer tokens = new StringTokenizer( command, "\t\f\n ", false );
String ipaddr = "0.0.0.0";
int port = -1;
while( tokens.hasMoreTokens() ){
args.addElement( tokens.nextToken() );
}
if( args.size() == 3 ){
ipaddr = ((String)args.elementAt( 1 ));
```

```
port = Integer.parseInt( ((String)args.elementAt( 2 )) );

client = new Socket( InetAddress.getByName( ipaddr ), port );
oout = new ObjectOutputStream( client.getOutputStream() );
oout.flush();
oin = new ObjectInputStream( client.getInputStream() );
PolicyRequest request = new PolicyRequest();
request.setContent( "hello" );
oout.writeObject( request );

PolicyResponse response = (PolicyResponse)oin.readObject();
System.out.println( response.toString() );

runUI();

}else{
throw new Exception( "bad args." );
}


}else{
   System.out.println( "Unrecognized command." );
   System.exit( -1 );
}

}


private void runUI(){

PolicyRequest request;
PolicyResponse response;
boolean die = false;
int num=0;
System.out.println( "" );
String input = "";

while( !die ){
request = new PolicyRequest();
response = null;
System.out.print( "pec["+num+"] " );

try{

input = keyboard.readLine();

/* if readLine() == null, die */
if( input == null || input.equals( "bye" ) || input.equals( "exit" ) ){
die=true;
keyboard.close();
```

```
request.setContent( "bye" );

}else{

request.setContent( input );

}

response = send( request );
System.out.println( response.toString() );

}catch( Exception e ){
  System.err.println( "Problem with command: "+e.getMessage() );
}
num++;
}

try{
if( oin!=null )
oin.close();
if( oout!=null ){
oout.flush();
oout.close();
}
if( client!=null && !client.isClosed() )
client.close();
}catch( Exception ex ){
System.err.println( "Problem closing connection: "+ex.getMessage() );
}finally{
oin=null;
oout=null;
keyboard = null;
client = null;
}
System.exit( 0 );
}

private PolicyResponse send( PolicyRequest request ) throws Exception{

oout.writeObject( request );
oout.flush();

return ((PolicyResponse)oin.readObject());

}

/**
 * SPCL client commands:
 * server returns "hi, allow, no, deny, bye"
 *
```

```
 * 'switch zone [zone]' : search this collection of policies
 *                        this cmd only succeeds if the s-o client is
                          recognized in this zone
 * 'switch policy [policy]' : search this policy for rules first
 * 'switch object [systemobject]' : become another system object (if it exists)
 * 'context' or 'current' : prints out current zone and policy namespace
 * 'can [principal] do [action] on [systemobject/me] with (param1=x AND param2=y);
 * 'list [active|suspended] rules'
 * 'list groups'
 * 'list zones'
 * 'list objects'
 * 'list principals'
 * 'list policies'
 * 'exit' 'bye' : end client session
 * 'kill' : shutdown server
 * eventually, we will provide 'install [fooinstaller.class]' where the
 *   policyengine takes the name of a client-local file, reads the bytes over
 *   the network and writes them to a file in SPCL_HOME/conf/policy, and then uses
 *   its install() method to load the policy in like normal.
 *
*/
public static void main( String [] args ){
//we could accept command line args, but for right now, prompt user
try{
new PolicyEngineClient();
}catch( Exception e ){
System.err.println( "PolicyEngineClient::main() problem="+e.getMessage() );
}

}

}

/*********************************************
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: May 8, 2003
*
*********************************************/
```

```
package spcl.data;

import java.util.*;

/**
 * A regular expression denoting the variations
 * on a command that a Principal can take wrt
 * a particular Object.
 *
 * actions{
      action a = "GET /~locasto/docs/*.html";
  a.meta_action = notify mom;
  you notify principals and
  you log to objects
   }
 *
 * @author locasto@cs.columbia.edu
 *
*/

public class Action{

public static final int NULL_META_ACTION = 5;
public static final int LOG_META_ACTION = 10;
public static final int NOTIFY_META_ACTION = 20;

private int meta_action = Action.NULL_META_ACTION;
private String meta_action_target;

private String regular_expression;
private SystemObject sysObj;
private String name;

public Action( String name ){
this.name = name;
}

public String getName(){
return name;
}

public String getActionPattern(){
return regular_expression;
}

public void setActionPattern( String regex ){
regular_expression = regex;
}

public void setObject( SystemObject sysObj ){
```

```
this.sysObj = sysObj;
}

public SystemObject getObject(){
return sysObj;
}

public void setMetaActionTarget( String target ){
meta_action_target=target;
}

public String getMetaActionTarget(){
return meta_action_target;
}

public int getMetaAction(){
return meta_action;
}

public void setMetaAction( int t ){
switch( t ){
case Action.LOG_META_ACTION: this.meta_action = t; break;
case Action.NOTIFY_META_ACTION: this.meta_action = t; break;
default: this.meta_action = Action.NULL_META_ACTION;
}
}


}

/*********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: May 9, 2003
*
*********************************************/

package spcl.data;
```

```
/**
 * The condition class represents something for the
 *  PolicyEngine to unify with the current params
 *  in a query.
 *
 * A Condition is a boolean expression:
 *   [ variable relop rval ]
 *   [ channel == 5 ]
 *
 * whereas a param is a name/value pair:
 *   channel/7
 *
 * This needs to be unified during the search.
 *
 * @author locasto@cs.columbia.edu
*/
public class Condition{

  public Variable var;
  public String relop;
  public boolean b_rval;
  public String rval;

  public Condition( Variable var, String relOp, boolean rval ){
    this.var = var;
    this.relop = relOp;
    this.b_rval = rval;
  }

  public Condition( Variable var, String relOp, String rval ){
    this.var = var;
    this.relop = relOp;
    this.rval = rval;
  }

  public boolean eval(){
int nval;
    switch( var.getType() ){
case Variable.NUMBER:
//all relops
try{nval = Integer.parseInt( rval );
}catch( NumberFormatException nfex ){ return false; }
if( relop.equals( "==" ) ){
return (nval==var.getNumberValue());
}else if( relop.equals( "!=" ) ){
return (nval!=var.getNumberValue());
}else if( relop.equals( ">" ) ){
return (var.getNumberValue()>nval);
```

```
}else if( relop.equals( "<" ) ){
return (var.getNumberValue()<nval);
}else if( relop.equals( ">=" ) ){
return (var.getNumberValue()>=nval);
}else if( relop.equals( "<=" ) ){
return (var.getNumberValue()<=nval);
}else{
return false;
}
case Variable.STRING:
//only != and ==
if( relop.equals( "==" ) ){
    return (rval.equals(var.getStringValue()));
}else if( relop.equals( "!=" ) ){
return (!rval.equals(var.getStringValue()));
}else{
return false;
}
case Variable.BOOLEAN:
//only != and ==
if( relop.equals( "==" ) ){
    return (b_rval==var.getBooleanValue());
}else if( relop.equals( "!=" ) ){
return (b_rval!=var.getBooleanValue());
}else{
return false;
}
}
return false;
  }

  public String toString(){
   return (var.toString()+relop+rval);
  }
}

import java.util.Hashtable;

/**
 * Title:
 * Description:
 *      This class is intended to handle compiler errors.
 *      The following are some possible ways to handle
 *      errors :
 *          1) Direct the error message to stderr and
 *             continue searching for other errors
 *          2) Direct the error message to a buffer and
 *             continue searching for other errors. At
 *             the end of compilation, perform some analysis
 *             on the error.
```

```
 *          3) The error is not recoverable, so the compilation
 *             must stop immediately.
 *
 * @see the method reportError for further detail
 *
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class ErrorHandler {

    /********************
     * a flag used to indicate whether error has been
     * found since the last time this flag is reset.
     */
    protected boolean error_found = false;

    /***************
     * a mapping from Exception class to the corresponding
     * message
     * @see constructor
     */
    protected Hashtable message = new Hashtable(10);

    /*********
     * constructor
     */
    public ErrorHandler() {
        message.put("VariableNotFoundException", "Undeclared variable");
        message.put("VariableConflictException", "Variable redeclaration");
        message.put("InvalidRelationException", "Invalid relation");
        message.put("TypeMisMatchException", "Type miss match");
        message.put("InvalidActionException", "Action not defined");
        message.put("NoRegexDefinedException", "No action regex defined for target
        message.put("SPCLCompilerException", "General Compiler Error");
        message.put("Exception", "Unexpected Error");
    }

    /***********
     * A way to reset the error flag, so that this
     * handler thinks that no error has been encountered.
     * This shall be called after the caller is sure that
     * previous reported errors could be ignored.
     */
    public void resetErrorFlag() {
        this.error_found = false;
    }
```

```
    /**********
     * a method that tells whether certain error has been
     * reported since the last time the flag is reset
     */
    public boolean hasError() {
        return error_found;
    }

    /*******************************
     * This is the major function that links all error
     * together. For now, just print all errors to stderr.
     * May possibly change the way compiler error
     * is handled later on.
     *
     * @a   the AST node where the error occur
     * @e   the exception that is thrown
     */
    public void reportError(LineAST a, Exception e) {
        this.error_found = true;
        System.out.print("line " + a.getLine() + " ( " + a.getText() + " ) : ");
        System.out.println(hash_message(e) + " - " + e.getMessage());
        if (!( e instanceof SPCLCompilerException )) {
            // this line shall never be called by the time
            // the project is done.
            e.printStackTrace();
        }
    }

    /************
     * A mapping from Exception class to the corresponding
     * message.
     *
     * @e   an exception
     *
     * @see protected Hashtable message
     * @see constructor
     */
    protected String hash_message(Exception e) {
        String s = (String)(message.get(e.getClass().getName()));
        if ( s == null ) {
            s = "Unexpected Error";
        }
        return s;
    }

}

/*********************************************
* Columbia University CS Department
* Network Security Lab
```

```
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: March 25, 2003
*
*********************************************/

package spcl.data;

/**
 * A utility for implementing the side-effect clause.
 * The <code>doFire()</code> method is invoked when a Rule
 * has been matched. The Rule has a list of Firing objects
 * and will iterate through them in the <code>Rule.fire()</code>
 * method, calling each <code>Firing.doFire()</code> method.
 * <p>
 * The body of the <code>doFire()</code> method itself
 * is dynamically generated as an anonymous inner class
 * of the PolicyInstaller.
 *
 * @author locasto@cs.columbia.edu
 *
 *
*/
public class Firing{

public void doFire(){}

}

/*********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
```

```
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: March 27, 2003
*
*********************************************/

package spcl.data;

import java.util.Hashtable;
import java.util.Enumeration;

/**
 * The Group class represents a collection of
 * associated Principals.
 *
 * Each Principal receives an implicit group.
 * The <code>rules</code> hashtable for this
 * implicit group is always empty, since these
 * Rules already exist inside the Principal.
 *
 * @author locasto@cs.columbia.edu
 *
*/
public class Group{

  private String name;
  private Hashtable rules = new Hashtable();
  private Hashtable members = new Hashtable();

  public Group( String n ){
    this.name = n;
  }

  public String getName(){
    return name;
  }

  public void addRule( Rule r ){
      rules.put( r.getName(), r );
  }

  public Rule getRule( String ruleName ){
    return ((Rule)rules.get( ruleName ));
  }

  public int getNumRules(){
    return (rules.size());
  }
```

```
  public Enumeration getRules(){
    return rules.elements();
  }

  public void addMember( Principal p ){
    members.put( p.getName(), p );
  }

  public boolean isMember( String principalName ){
    if( null==members ) return false;
    return members.containsKey( principalName );
  }
}


/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class InvalidActionException extends SPCLCompilerException {

    public InvalidActionException() {
        super();
    }

    public InvalidActionException(String object_id, String action) {
        super('"' + action + "\" on " + object_id);
    }

}


/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class InvalidRelationException extends SPCLCompilerException {

    public InvalidRelationException() {
```

```
    }

    public InvalidRelationException(String s) {
        super(s);
    }

    public InvalidRelationException(int first_type, String relation, int second_ty
        super(
            SymbolTable.typeToName(first_type)+" "
            +relation+" "
            +SymbolTable.typeToName(second_type)
        );
    }

}


/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class LineAST extends antlr.CommonAST {

    private int line;

    public LineAST() {
        super();
    }

    public void initialize( antlr.collections.AST t ) {
        super.initialize( t );
        this.line = -1;
    }

    public void initialize( antlr.Token t ) {
        super.initialize( t );
        this.line = t.getLine();
    }

    public void initialize( int i, String s ) {
        super.initialize(i, s);
        this.setType(i);
        this.setText(s);
        this.line = -1;
    }
```

```
    public void setLine(int line) {
        this.line = line;
    }

    public int getLine() {
        if ( this.line == -1 ) {
            return ((LineAST)this.getFirstChild()).getLine();
        }
        return this.line;
    }

    public String toStringList() {
//        return " line " + this.getLine() + ":" + super.toStringList();
        return super.toStringList();
    }
}



/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class NoRegexDefinedException extends SPCLCompilerException {

    public NoRegexDefinedException() {
        super();
    }

    public NoRegexDefinedException(String s) {
        super(s);
    }
}

/*********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
```

```
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: April 10, 2003
*
*********************************************/

package spcl.data;

import java.util.Hashtable;


/**
 * maintains the current in focus policy, zone, object
 * for the policy engine.
 *
 * @author locasto@cs.columbia.edu
 *
*/
public class PolicyContext{

//the current zone
public Zone zone = null;

//the currently connected system object (client)
public SystemObject object = null;

//the policy to search first. By default, the first policy in the zone.
public Policy policy = null;

public String toString(){
StringBuffer sb = new StringBuffer( 256 );
sb.append( "<context>" );
sb.append( "\n\t<zone name=\"" );
if( zone!=null ) sb.append( zone.getName() );
sb.append( "\"/>" );
sb.append( "\n\t<policy name=\"" );
if( policy!=null ) sb.append( policy.getName() );
sb.append( "\"/>" );
sb.append( "\n\t<system name=\"" );
if( object!=null ) sb.append( object.getName() );
sb.append( "\"/>" );
sb.append("\n</context>");
return sb.toString();
}
}
```

```
/*********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: March 27, 2003
*
*********************************************/

package spcl.client;

import spcl.data.*;
import java.net.*;
import java.io.*;
import java.util.*;

/**
 * A basic utility for communicating with
 * the policyengine. Starts up an
 * interactive command line interpreter
 *
 * @author locasto@cs.columbia.edu
 *
*/
public class PolicyEngineClient{

private Socket client;
private ObjectInputStream oin;
private ObjectOutputStream oout;
private InputStreamReader isr;
private BufferedReader keyboard;


public PolicyEngineClient() throws Exception{
//prompt user for 'connect ip port'

String command = "";
Vector args = new Vector( 7 );
isr = new InputStreamReader(System.in);
keyboard = new BufferedReader(isr);
```

```
System.out.print( "\npec> " );
command = keyboard.readLine();
command = command.trim();
if( command==null || command.equals( "" ) || command.equalsIgnoreCase( "bye" ) ||
System.out.println( "PolicyEngine client exiting..." );
System.exit( 0 );
}else if( command.equalsIgnoreCase( "help" ) ){
System.out.println( "\nHELP:\n\tbye,exit,quit,help,connect [host] [port]\n" );
}else if( command.startsWith( "connect " ) ){
StringTokenizer tokens = new StringTokenizer( command, "\t\f\n ", false );
String ipaddr = "0.0.0.0";
int port = -1;
while( tokens.hasMoreTokens() ){
args.addElement( tokens.nextToken() );
}
if( args.size() == 3 ){
ipaddr = ((String)args.elementAt( 1 ));
port = Integer.parseInt( ((String)args.elementAt( 2 )) );

client = new Socket( InetAddress.getByName( ipaddr ), port );
oout = new ObjectOutputStream( client.getOutputStream() );
oout.flush();
oin = new ObjectInputStream( client.getInputStream() );
PolicyRequest request = new PolicyRequest();
request.setContent( "hello" );
oout.writeObject( request );

PolicyResponse response = (PolicyResponse)oin.readObject();
System.out.println( response.toString() );

runUI();

}else{
throw new Exception( "bad args." );
}


}else{
   System.out.println( "Unrecognized command." );
   System.exit( -1 );
}

}


private void runUI(){

PolicyRequest request;
PolicyResponse response;
boolean die = false;
```

```
int num=0;
System.out.println( "" );
String input = "";

while( !die ){
request = new PolicyRequest();
response = null;
System.out.print( "pec["+num+"] " );

try{

input = keyboard.readLine();

/* if readLine() == null, die */
if( input == null || input.equals( "bye" ) || input.equals( "exit" ) ){
die=true;
keyboard.close();
request.setContent( "bye" );

}else{

request.setContent( input );

}

response = send( request );
System.out.println( response.toString() );

}catch( Exception e ){
  System.err.println( "Problem with command: "+e.getMessage() );
}
num++;
}

try{
if( oin!=null )
oin.close();
if( oout!=null ){
oout.flush();
oout.close();
}
if( client!=null && !client.isClosed() )
client.close();
}catch( Exception ex ){
System.err.println( "Problem closing connection: "+ex.getMessage() );
}finally{
oin=null;
oout=null;
keyboard = null;
client = null;
```

```
}
System.exit( 0 );
}

private PolicyResponse send( PolicyRequest request ) throws Exception{

oout.writeObject( request );
oout.flush();

return ((PolicyResponse)oin.readObject());

}

/**
 * SPCL client commands:
 * server returns "hi, allow, no, deny, bye"
 *
 * 'switch zone [zone]' : search this collection of policies
 *                        this cmd only succeeds if the s-o client is
                          recognized in this zone
 * 'switch policy [policy]' : search this policy for rules first
 * 'switch object [systemobject]' : become another system object (if it exists)
 * 'context' or 'current' : prints out current zone and policy namespace
 * 'can [principal] do [action] on [systemobject/me] with (param1=x AND param2=y);
 * 'list [active|suspended] rules'
 * 'list groups'
 * 'list zones'
 * 'list objects'
 * 'list principals'
 * 'list policies'
 * 'exit' 'bye' : end client session
 * 'kill' : shutdown server
 * eventually, we will provide 'install [fooinstaller.class]' where the
 *   policyengine takes the name of a client-local file, reads the bytes over
 *   the network and writes them to a file in SPCL_HOME/conf/policy, and then uses
 *   its install() method to load the policy in like normal.
 *
*/
public static void main( String [] args ){
//we could accept command line args, but for right now, prompt user
try{
new PolicyEngineClient();
}catch( Exception e ){
System.err.println( "PolicyEngineClient::main() problem="+e.getMessage() );
}

}

}
```

```
/*********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: March 26, 2003
*
*********************************************/

package spcl.policyengine;

import java.net.*;
import java.io.*;

/**
 * A basic utility for communicating with
 * the policyengine. Starts up on a
 * port to listen for remote shutdown requests.
 *
 * @author locasto@cs.columbia.edu
 *
*/
public class PolicyEngineConsole implements Runnable{

private Socket client;
private ServerSocket server;
private Thread listener;
private boolean stop=false;
private int port;
private String message;
private PolicyEngine policyEngine;

public PolicyEngineConsole(){}

public void setPolicyEngine( PolicyEngine pe ){ this.policyEngine = pe; }
public void setPort( int p ){ port = p; }
public void setMessage( String s ){ message=s; }

void init() throws Exception{
  server = new ServerSocket( port, 2, InetAddress.getLocalHost() );
```

```
  listener = new Thread( this );
  listener.start();
}

public void run(){
String s;
ObjectInputStream ois;
ObjectOutputStream oos;

  //wait for connections telling you to shutdown
  while( !stop ){
    listener.yield();

try{

listener.sleep( 15 );
client = server.accept();
ois = new ObjectInputStream( client.getInputStream() );
oos = new ObjectOutputStream( client.getOutputStream() );
oos.flush();

s = (String)ois.readObject();
if( s.equals( message ) ){
stop = true;
policyEngine.shutDown();
}else{
  //log attempt and continue
  System.err.println( "PolicyEngineConsole::run() bad commend="+s );
}

ois.close();
oos.close();
client.close();
}catch( Exception e ){
ois=null;
oos=null;
client=null;
}
  }
try{server.close();}catch( Exception ex ){}
}




}

/**********************************************
* Columbia University CS Department
```

```
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: March 28, 2003
*
*********************************************/

package spcl.policyengine;

import spcl.data.*;


/**
 * A simple driver for the PolicyEngine
 *
 * @author locasto@cs.columbia.edu
 *
*/
public class PolicyEngineDriver{

private PolicyEngine policyEngine;

public PolicyEngineDriver() throws Exception{
   policyEngine = new PolicyEngine( this.getClass().getClassLoader() );
}

public static void main( String [] args ){

  try{

     PolicyEngine.execute( args );

     new PolicyEngineDriver();

  }catch( Exception e ){
     System.err.println( "Could not start PolicyEngine: "+e.getMessage() );
     System.exit( -1 );
  }

}
```

```
}

/*********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: May 9, 2003
*
*********************************************/

package spcl.policyengine;

import spcl.data.*;
import java.util.*;
import java.net.*;
import java.io.*;
import java.util.regex.*;
import java.text.*; //DateFormat

/**
 * <p>
 * The PolicyEngine is the core of execution of a
 * policy. A general policy is intially loaded and resolved and
 * then a sub-policy is computed based on input events and
 * returned to the querying system.
 * </p>
 * <p>
 * The model of computation is very Prolog-like. We first narrow the
 * search tree by considering the possible namespace (zone), policy,
 * principal, object, and group we are talking about. Then we match
 * the rules that apply and determine if there are any conflicts.
 * </p>
 * <p>
 * This class is a concrete implementation of a machine that
 * understands the execution of policy as defined by SPCL.
 * </p>
 * <p>
 * This class wraps a serversocket and is a runnable single-threaded
 *  server, a PolicyLoader, and a ClassLoader.
```

```
 * </p>
 * <br><br>
 * The model of execution is thus:
 *  <ul>
 *   <li> there is a period of data initialization
 *    of the primary data structures in this machine</li>
 *
 *   <li> These data structures are the 'map' of the relations
 *    between policies, principals, zones, objects, and groups.</li>
 *
 *   <li>looped interpretation and updating of rule state based on input</li>
 *
 *   <li>possible loading of new data structures, followed by a resolution
 *    phase</li>
 *  </ul>
 *
 * @author locasto@cs.columbia.edu
 *
 *
*/
public class PolicyEngine extends ClassLoader implements Runnable, PolicyLoader{

private Hashtable policyFiles = new Hashtable();

private Hashtable zones=new Hashtable();

/** no Zone can be named zone in the source code, so naming our
 * default Zone 'zone:zone' protects it.
*/
private Zone defaultZone = new Zone( "zone:zone" );

private PolicyEngineConsole console;
private Thread listener;
private boolean stop = false;
private ServerSocket server = null;
private Socket client = null;
private int port = -1;

private StringBuffer sb = new StringBuffer( 1024 );

/* these variables are maintained by the policyengine's server thread */

/** This object represents the current zone, policy to search
 * first, and system object we are currently dealing with
 */
private PolicyContext current = new PolicyContext();

/* these are vars used on startup for getting different config files, etc */

public static String SPCL_HOME = System.getProperty( "spcl.home" );
```

```
private static String configFileName = SPCL_HOME+File.separatorChar+"conf"+File.se
private static String policyInstallerClassName;


/** Create a policy engine instance */
public PolicyEngine( ClassLoader parent ) throws Exception{
    super( parent );

        if( SPCL_HOME == null || SPCL_HOME.equals( "" ) )
          throw new Exception( "SPCL_HOME not defined." );

        defaultZone.attach( new Policy( "defaultPolicy" ) );
        zones.put( defaultZone.getName(), defaultZone );

        //install supplied 'bootstrap' policy given on command line
        install( policyInstallerClassName );

        console = new PolicyEngineConsole();

        configure();

        init();

        System.out.println( "\nPolicyEngine bound to :"+port );
    }

/**
 * Read the class file and create a PolicyInstaller
 * subclass instance. Then, call the doInstall() method.
 *
*/
private void install( String classname ) throws PolicyException, IOException, Exce

String filename = "";

filename = classname.replace( '.', File.separatorChar );
filename+=".class";
filename = SPCL_HOME+File.separatorChar+"conf"
                            +File.separatorChar+"policy"
                            +File.separatorChar+filename;

policyFiles.put( classname, pull( new File( filename ) ) );

Object o = this.loadClass( classname, true ).newInstance();

PolicyInstaller installer = ((PolicyInstaller)o);
installer.setPolicyLoader( ((PolicyLoader)this) );
installer.doInstall();
System.out.println( "PolicyEngine::install( "+classname+" )" );
```

```
//save on memory
policyFiles.remove( classname );
}

private byte [] pull( File f ) throws IOException{

String name=f.getName();
int totalFileBytes = (int)f.length();
byte [] b = new byte[ totalFileBytes ];

FileInputStream fin = new FileInputStream( f );

int i=fin.read( b );
fin.close();
return b;
}

/**
 * open the config file and get our startup information:
 *   spcl.port
 *   spcl.console.port
 *   spcl.shutdownmessage
 *
 * additional info we can put in here is a list of all legal
 * policyinstaller files to bootstrap with (for the future)
 *
*/
private void configure() throws Exception{

Properties configs = new Properties();

configs.load( new FileInputStream( new File( configFileName ) ) );

port = Integer.parseInt( configs.getProperty( "spcl.port", "3780" ) );
console.setPolicyEngine( this );
console.setPort( Integer.parseInt( configs.getProperty( "spcl.console.port", "3781
console.setMessage( configs.getProperty( "spcl.shutdownmessage", "" ) );
configs = null;

}

public void init() throws Exception{

server = new ServerSocket( port, 10, InetAddress.getLocalHost() );
listener = new Thread( this );
listener.start();

console.init();

}
```

```
public void run(){

PolicyRequest request=null;
PolicyResponse response=null;

ObjectInputStream oin=null;
ObjectOutputStream oout=null;

while( !stop ){
listener.yield();
request = new PolicyRequest();

try{

client = server.accept();
System.out.println( "PolicyEngine got client: "+client.getInetAddress() );

oout = new ObjectOutputStream( client.getOutputStream() );
oout.flush();
oin = new ObjectInputStream( client.getInputStream() );

try{
while( !request.getContent().equals("bye") ){
request = ((PolicyRequest)oin.readObject());

response = process( request );

oout.writeObject( response );
oout.flush();
}
}catch( ClassCastException ccex ){
System.err.println( "Wrong input type: "+ccex.getMessage() );
}catch( NullPointerException npex ){
System.err.println( "Null pointer: "+npex.getMessage() );
npex.printStackTrace();
}catch( Exception ex ){
  //maybe class cast
  System.err.println( "PolicyEngine::run() problem getting input:"+ex.getMessage()
  ex.printStackTrace();
}
oin.close();
oout.close();
client.close();
listener.yield();
}catch( Exception e ){
e.printStackTrace();
System.err.println( "PolicyEngine::run() problem running server:"+e.getMessage() )
}finally{
  oin=null;
```

```
  oout=null;
  client=null;
}
}
System.exit( 0 );
}

private PolicyResponse process( PolicyRequest request ){

PolicyResponse response = new PolicyResponse( request );
String content = request.getContent();
content = content.trim();
sb.setLength(0);
Enumeration e=null;

if( content.equals( "hello" ) ){
response.setContent( "<hi/>" );
}else if( content.equals( "bye" ) ){
response.setContent( "<bye/>" );
}else if( content.equals( "list principals" ) ){
e = current.policy.getPrincipals();
sb.append("<principals>");
while( e.hasMoreElements() ){
sb.append( "\n\t<principal name=\"" );
sb.append( ((Principal)e.nextElement()).getName() );
sb.append( "\"/>" );
}
sb.append("\n</princpals>");
response.setContent( sb.toString() );
}else if( content.equals( "list objects" ) ){
e = current.policy.getObjects();
sb.append("<objects>");
while( e.hasMoreElements() ){
sb.append( "\n\t<object name=\"" );
sb.append( ((SystemObject)e.nextElement()).getName() );
sb.append( "\"/>" );
}
sb.append("\n</objects>");
response.setContent( sb.toString() );
}else if( content.equals( "list groups" ) ){
e = current.policy.getGroups();
sb.append("<groups>");
while( e.hasMoreElements() ){
sb.append( "\n\t<group name=\"" );
sb.append( ((Group)e.nextElement()).getName() );
sb.append( "\"/>" );
}
sb.append("\n</groups>");
response.setContent( sb.toString() );
}else if( content.equals( "list policies" ) ){
```

```
e = current.zone.getPolicies();
sb.append("<policies>");
while( e.hasMoreElements() ){
sb.append( "\n\t<policy name=\"" );
sb.append( ((Policy)e.nextElement()).getName() );
sb.append( "\"/>" );
}
sb.append("\n</policies>");
response.setContent( sb.toString() );
}else if( content.equals( "list active rules" ) ){
//for all principals and groups in this policy, list all active rules
}else if( content.equals( "list suspended rules" ) ){
//for all principals and groups in this policy, list suspended rules
}else if( content.equals( "list rules" ) ){
//for all principals and groups in this policy, list all rules
}else if( content.equals( "current" ) || content.equals( "context" ) ){
//print out current context
sb.append( current.toString() );
response.setContent( sb.toString() );
}else if( content.startsWith( "switch policy " ) ){
int lastSpacePos = content.lastIndexOf( " " );
if( lastSpacePos==-1 ){
response.setContent( "<no reason=\"bad format\"/>" );
}else{
String to_policy = content.substring( lastSpacePos+1 );
if( switchPolicy( to_policy ) ){
response.setContent( "<new-policy name=\""+to_policy+"\"/>" );
}else{
response.setContent( "<no reason=\"named policy dne\"/>" );
}
}
}else if( content.startsWith( "switch zone " ) ){
//only if current object is in new zone!!
int lastSpacePos = content.lastIndexOf( " " );
if( lastSpacePos==-1 ){
response.setContent( "<no reason=\"bad format\"/>" );
}else{
String to_zone = content.substring( lastSpacePos+1 );
if( switchZone( to_zone ) ){
response.setContent( "<new-zone name=\""+to_zone+"\"/>" );
}else{
response.setContent( "<no reason=\"named policy dne\"/>" );
}
}
}else if( content.startsWith( "switch object " ) ){
int lastSpacePos = content.lastIndexOf( " " );
if( lastSpacePos==-1 ){
response.setContent( "<no reason=\"bad format\"/>" );
}else{
String to_object = content.substring( lastSpacePos+1 );
```

```
if( switchObject( to_object ) ){
response.setContent( "<new-object name=\""+to_object+"\"/>" );
}else{
response.setContent( "<no reason=\"named object dne\"/>" );
}
}
}else if( content.equals( "list zones" ) ){
sb.append("<zonelist>");
e = zones.elements();
while( e.hasMoreElements() ){
sb.append( "\n\t<zone name=\"" );
sb.append( ((Zone)e.nextElement()).getName() );
sb.append( "\"/>" );
}
sb.append("\n</zonelist>");
response.setContent( sb.toString() );
}else if( content.startsWith( "CAN" ) || content.startsWith( "can" ) ){
//evaulate the "CAN" query
System.err.println( "\t>> checking query ["+content+"]" );
sb.append( doQuery( content ) );
response.setContent( sb.toString() );
}else{
response.setContent( "<no/>" );
}

return ((PolicyResponse)response);
}

/**
 * use the policy engine to answer questions like:
 * can michael do "open" with (fridge.door=locked)?
   can [principal] do [action] with (param1=x AND param2=y);
   can, michael, do, action, with, param1, x, AND, param2, y
   args[0]=CAN
   args[1]=principalName;
   args[2]=do
   args[3]=actionString
   current.policy
   current.object
   current.zone
*/
private String doQuery( String content ){
if( current.zone==null || current.zone.isEmpty() ||
    current.policy==null || current.object==null )
return "<no/>";
Principal dude = null; //current client principal
Group dudeGroup=null;
String response = "<no/>";
Vector paramList = new Vector(); //store new Variable( param1, x );
Variable newVar = null;
```

```
StringTokenizer cmdTokens = new StringTokenizer( content, " \n\t\r(),;?\"", false
String [] args = new String[cmdTokens.countTokens()]; //load cmdTokens into args
int i=0;
if( args.length<4 ){
System.err.println( "\t>> args < 4" );
return "<no/>";
}

System.err.print( "\t>> " );
while( cmdTokens.hasMoreTokens() ){
args[i]=cmdTokens.nextToken();
System.err.print( " ["+args[i]+"] " );
i++;
}
System.err.print( "\n" );
if( !args[2].equalsIgnoreCase( "do" ) )
return "<no/>";
if( (args.length>=5) && !args[4].equalsIgnoreCase( "with" ) ){
System.err.println( "\t>> missing \'with\'" );
return "<no/>";
}
String principalName = args[1];
String actionString = args[3];
String objectName = current.object.getName();

//parse content
//consume tokens from args
//AND is a delimiter
//x=5
//y=true (express bindings)
// we should get this pattern:  id eq val AND id eq val
// string EQ QUOTE string QUOTE AND string EQ number AND string EQ TRUE | FALSE ..
// create new variable, guess its type: ''==string true or false == boolean, else
i=5; //args[5]
String newVarName="";
int newVarType=-1;
String newVarValue="";
while( i<args.length ){

newVarName = consumeString( args[i] );
if( newVarName==null ){
System.err.println( "\t>> parseConditions::bad var name" );
return "<no/>";
}
i++;
if( i<args.length ){
if( !consumeEquals( args[i] ) ){
System.err.println( "\t>> parseConditions::expected =" );
return "<no/>";
```

```
}
i++;
}else{
System.err.println( "\t>> parseConditions::expected =, saw EOS" );
return "<no/>";
}
if( i<args.length ){
newVarValue = consumeString( args[i] );
if( newVarValue==null ){
System.err.println( "\t>> parseConditions::bad value" );
return "<no/>";
}
i++;
}else{
System.err.println( "\t>> parseConditions::expected value, saw EOS" );
return "<no/>";
}
if( i<args.length ){
if( !consumeAND( args[i] ) ) {
System.err.println( "\t>> parseConditions::expected AND" );
return "<no/>";
}
i++;
}else{
//no error, there are just no more clauses
}
//decide type
newVarValue = newVarValue.trim();
if( newVarValue.equalsIgnoreCase( "true" ) ||
    newVarValue.equalsIgnoreCase( "false" ) ){
newVarType = spcl.data.Variable.BOOLEAN;
}else if( newVarValue.startsWith( "\'" ) &&
    newVarValue.endsWith("\'") ){
newVarType = spcl.data.Variable.STRING;
newVarValue = newVarValue.replace( '\'', '\0' );
}else{
try{
Integer.parseInt( newVarValue );
newVarType = spcl.data.Variable.NUMBER;
}catch( NumberFormatException nfex ){
return "<no/>";
}
}
newVar = new Variable( newVarName, newVarType, newVarValue );
paramList.addElement( newVar );
}

//get principal, object, action, param list from current Policy
//find the principal in our current policy
if( !current.policy.hasPrincipal( principalName ) ){
```

```
System.err.println( "\t>> no named principal in policy" );
return "<no/>";
}else{
dude = current.policy.getPrincipal( principalName );
}

//do principal resolution
response = checkRules( dude.getRules(), (new int[dude.getNumRules()]),
                       actionString, paramList );
if( response.equals( "<allow/>" ) || response.equals( "<deny/>" ) ){
return response;
}else if( response.equals( "<wishy-washy/>" ) ){
//fall through
System.err.println( "\t>> principal def was wishy-washy" );
}else{
//dont' know, error, return no
System.err.println( "\t>> principal def returned unknown answer" );
return "<no/>";
}
//if principal exists, but no rule applied (all wishy washy), find
//all groups with him a member of them and look at their rules
Vector userGroups = current.policy.getUserGroups( principalName );
System.err.println( "\t>> current.policy.getUserGroups("+principalName+");" );
//for each group, checkRules( group.getRules(), ...);
//do group resolution...it is permissive, first one to make a decision
//wins...
System.err.println( "\t>> usergroups size="+userGroups.size() );
for( i=0;i<userGroups.size();i++ ){
dudeGroup = (Group)userGroups.elementAt(i);
response = checkRules( dudeGroup.getRules(), (new int[dudeGroup.getNumRules()]),
                       actionString, paramList );
if( response.equals( "<allow/>" ) || response.equals( "<deny/>" ) ){
return response;
}else if( response.equals( "<wishy-washy/>" ) ){
//fall through
System.err.println( "\t>> group def was wishy-washy" );
}else{
System.err.println( "\t>> group def had unknown response" );
return "<no/>";
}
}

//finally, check default_p
Principal defP = current.policy.getDefaultPrincipal();
response = checkRules( defP.getRules(), (new int[defP.getNumRules()]),
                       actionString, paramList );
if( response.equals( "<allow/>" ) || response.equals( "<deny/>" ) ){
return response;
}else if( response.equals( "<wishy-washy/>" ) ){
//default response
```

```
System.err.println( "\t>> default principal says no (wishy-washy)" );
return "<no/>";
}else{
System.err.println( "\t>> default principal says no" );
return "<no/>";
}
}

private
String checkRules( Enumeration rules, int [] responses,
                   String actionString, Vector paramList ){
int i=0;
Rule tempRule=null;
if( rules==null || responses.length==0 || paramList==null )
return "<wishy-washy/>";
//ok, now we have enough info. we've found the appropriate ruleset/group/principal
//we look through the rules that are active and match params from paramList
//if a rule matches, we getAccess() on it into responses
while( rules.hasMoreElements() && i<responses.length ){
tempRule = (Rule)rules.nextElement();
if( tempRule.isActive() ){
responses[i]=tempRule.evaluate( paramList, actionString, current.object );
if( responses[i]==spcl.data.Rule.ALLOW ||
responses[i]==spcl.data.Rule.DENY ){
System.err.println( "\t\t>> tempRule["+tempRule.getName()+"].fire() " );
tempRule.fire();
}
}else{
responses[i]=spcl.data.Rule.WISHY_WASHY;
}
System.err.println( "\t\t\t>> responses["+tempRule.getName()+"]="+responses[i] );
i++;
}
//responses[all]==spcl.data.Rule.WISHY_WASHY, fall through
boolean foundAllow=false;
boolean foundDeny=false;
boolean foundWishyWashy=false;
for( i=0;i<responses.length;i++ ){
switch( responses[i] ){
case spcl.data.Rule.DENY:
if( foundDeny ){
//double deny?? do nothing...
}else if( foundAllow ){
//inconsistent
return "<inconsistent-policy-deny/>";
}else if( foundWishyWashy ){
//let it fly
}
foundDeny=true;
break;
```

```
case spcl.data.Rule.ALLOW:
if( foundAllow ){
//double allow?? do nothing...
}else if( foundDeny ){
//inconsistent
return "<inconsistent-policy-deny/>";
}else if( foundWishyWashy ){
//let it fly
}
foundAllow=true;
break;
case spcl.data.Rule.WISHY_WASHY:
if( foundDeny ){
//wishy washy and deny
}else if( foundAllow ){
//wishy washy and allow
}else if( foundWishyWashy ){
//let it fly
}
foundWishyWashy=true;
break;
}
}
if( foundDeny ){
return "<deny/>";
}else if( foundAllow ){
return "<allow/>";
}
return "<wishy-washy/>";
}

private boolean consumeAND( String arg ){
arg=arg.trim();
return (arg.equalsIgnoreCase( "and" ));
}

private boolean consumeEquals( String arg ){
arg=arg.trim();
return (arg.equals( "=" ));
}

private String consumeString( String arg ){
if( consumeAND(arg) || consumeEquals(arg) )
return null;
else
return (arg.trim());
}

/** Explicit zone context switch. Also occurs implicitely after a load() */
public boolean switchZone( String newZone ){
```

```
Zone currZone = current.zone;
current.zone = lookupZone( newZone );
if( current.zone==null ){
current.zone=currZone;
return false;
}
current.policy = current.zone.getFirst();
current.object = null;
return true;
}

public boolean switchPolicy( String newPolicy ){
Policy currPol = current.policy;
current.policy = current.zone.getPolicy( newPolicy );
if( current.policy==null ){
current.policy = currPol;
return false;
}
current.object = null;
return true;
}

public boolean switchObject( String to_objectName ){
if( current.policy.hasObject( to_objectName ) ){
current.object = current.policy.getObject( to_objectName );
return true;
}else{
return false;
}
}

void shutDown(){
try{
stop = true;
if( server!=null )
server.close();
}catch( Exception ex ){
System.out.println( "PolicyEngine::shutDown() problem: "+ex.getMessage() );
}
System.exit( 0 );
}

//---------------------------------- ClassLoader methods

protected Class findClass( String classname ) throws ClassNotFoundException{

Class c=null;
ClassLoader myParent = this.getClass().getClassLoader();
if( myParent == null )
myParent = ClassLoader.getSystemClassLoader();
```

```
if( null == c ){

//should try to say "parent, find class X" to prevent
//malicious programs from redefining java.lang.String for
//example
c = super.findClass( classname );

byte [] b = loadClassData( classname );

c = defineClass( classname, b, 0, b.length );

if( null == c )
throw new ClassNotFoundException( classname );
}

resolveClass( c );

return c;
}

/**
 * locate the appropriate bytes and return them
*/
private byte [] loadClassData( String classname ){
byte [] b = new byte[0];
b = ((byte[])policyFiles.get( classname ));
if( b==null || b.length==0 ){
return new byte[0];
}else{
return ((byte[])b);
}
}



//---------------------------------- interface PolicyLoader

public void load( Zone zone ){
//install the zone in this policyengine
zones.put( zone.getName(), zone );
current.zone = zone;
current.policy = zone.getFirst();
}

/**
 * Assumes that zone already exists.
 * @throws PolicyException - if zone DNE
*/
public void load( Policy policy, String zoneName ) throws PolicyException{
```

```
Zone z = lookupZone( zoneName );
if( z==null )
throw new PolicyException(
"Could not load policy "+policy.getName()+". Zone "+zoneName+" does not exist.",
PolicyException.NULL_ZONE
);
z.attach( policy );
current.zone = z;
current.policy = z.getFirst();
}

/** Look up the current zone. Does not auto-switch zone contexts. */
public Zone lookupZone( String zoneName ){
if( defaultZone.getName().equals( zoneName ) ){
return defaultZone;
}else{
return ((Zone)zones.get( zoneName ));
}
}

public boolean unload( String policyName ){
return false;
}

public boolean suspend( String policyName ){
return false;
}

public boolean resume( String policyName ){
return false;
}

//for special case 'system' and 'time' and boolean, no meaning
public
boolean resolveCondition( int sys_var, String relop, boolean rval ){
return false;
}

public
boolean resolveCondition( int sys_var, String relop, String rval ){

if( sys_var!=spcl.data.PolicyLoader.SYSTEM_TIME )
return false;

Date dnow = new java.util.Date();
long now = -1;

DateFormat df = DateFormat.getDateInstance( DateFormat.SHORT );
DateFormat dtf = DateFormat.getDateTimeInstance( DateFormat.SHORT, DateFormat.SHOR
DateFormat tf = DateFormat.getTimeInstance( DateFormat.SHORT );
```

```
Date rvalDate = null;
long rvaltime = 0;

try{
rvalDate = df.parse( rval );
dnow = df.parse( df.format( dnow ) );
System.err.println( "\t>> resolveCondition( SYSTEM_TIME )::full date" );
}catch( Exception dfex ){
System.err.println( "\t>> full date failed:"+dfex.getMessage() );
try{
rvalDate = dtf.parse( rval );
dnow = dtf.parse( dtf.format( dnow ) );
System.err.println( "\t>> resolveCondition( SYSTEM_TIME )::full date full time" );
}catch( Exception dtfex ){
System.err.println( "\t>> full date+time failed:"+dtfex.getMessage() );
try{
rvalDate = tf.parse( rval );
dnow = tf.parse( tf.format( dnow ) );
System.err.println( "\t>> resolveCondition( SYSTEM_TIME )::time only" );
}catch( Exception tfex ){
System.err.println( "\t>> full time failed:"+tfex.getMessage() );
System.err.println( "\t>> resolveCondition( SYSTEM_TIME )::no parse time" );
return false;
}
}
}

now = dnow.getTime();

if( relop.equals( "==" ) ){
System.err.println( "\t>> resolveCondition( SYSTEM_TIME )::dates are equ" );
return ((dnow.equals( rvalDate )));
}else if( relop.equals( "!=" ) ){
System.err.println( "\t>> resolveCondition( SYSTEM_TIME )::dates are !equ" );
return ((!dnow.equals( rvalDate )));
}else if( relop.equals( "<" ) ){
System.err.println( "\t>> resolveCondition( SYSTEM_TIME )::now < rval" );
return ((dnow.before( rvalDate )));
}else if( relop.equals( ">" ) ){
System.err.println( "\t>> resolveCondition( SYSTEM_TIME )::now > rval" );
return ((dnow.after( rvalDate )));
}else if( relop.equals( "<=" ) ){
System.err.println( "\t>> resolveCondition( SYSTEM_TIME )::now <= rval" );
return ((dnow.before( rvalDate ))||((dnow.equals( rvalDate ))));
}else if( relop.equals( ">=" ) ){
System.err.println( "\t>> resolveCondition( SYSTEM_TIME )::now >= rval" );
return ((dnow.after( rvalDate ))||((dnow.equals( rvalDate ))));
}else{
return false;
```

```
}
}

//for special case boolean
public
boolean resolveCondition( Variable var, String relop, boolean rval ){
if( var.getType()!=spcl.data.Variable.BOOLEAN )
return false;

if( relop.equals( "==" ) ){
return (var.getBooleanValue()==rval);
}else if( relop.equals( "!=" ) ){
return (var.getBooleanValue()!=rval);
}else{
return false;
}
}

//rc( ip_addr,">", "rval", spcl.data.Variable.STRING );
public
boolean resolveCondition( Variable var, String relop, String rval ){
int i_rval;

switch( var.getType() ){

case spcl.data.Variable.NUMBER:
i_rval = Integer.parseInt( rval );
if( relop.equals( "==" ) ){
return ((i_rval==var.getNumberValue()));
}else if( relop.equals( "!=" ) ){
return ((i_rval!=var.getNumberValue()));
}else if( relop.equals( ">" ) ){
return ((var.getNumberValue()>i_rval));
}else if( relop.equals( "<" ) ){
return ((var.getNumberValue()<i_rval));
}else if( relop.equals( "<=" ) ){
return ((var.getNumberValue()<=i_rval));
}else if( relop.equals( ">=" ) ){
return ((var.getNumberValue()>=i_rval));
}else{
return false;
}
case spcl.data.Variable.BOOLEAN:
return false;
case spcl.data.Variable.STRING:
if( relop.equals( "==" ) ){
return ((rval.equals( var.getStringValue())));
}else if( relop.equals( "!=" ) ){
return ((!rval.equals( var.getStringValue())));
}else if( relop.equals( ">" ) ){
```

```
return false;
}else if( relop.equals( "<" ) ){
return false;
}else if( relop.equals( "<=" ) ){
return false;
}else if( relop.equals( ">=" ) ){
return false;
}else{
return false;
}
default:
}
return false;
}



//---------------------------------- private static methods

private static void doUsage(){

System.out.println( "\n\nUsage:\n\tjava spcl.PolicyEngine -stop\n" );
System.out.println( "\tjava spcl.PolicyEngine -policy [fully.qualified.classname]\
System.out.println( "The -policy file parameter is relative to SPCL_HOME/conf/poli
}

public static void execute( String [] args ){
    if( args.length==2 && args[0].equals( "-policy" ) ){
policyInstallerClassName = args[1];
}else if( args.length==1 && args[0].equals( "-stop" ) ){
try{
new StopTask( configFileName );
}catch( Exception e ){
System.err.println( "Problem running stoptask: "+e.getMessage() );
}
System.exit( 0 );
}else{
  doUsage();
  System.exit( -2 );
}
}

}


/**********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
```

```
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: March 28, 2003
*
*********************************************/

package spcl.data;

/**
 * A robust utility for the different types of
 * high-level errors that can be caused during
 * the loading, resolution, querying, and
 * retiring of policy.
 *
 * Types need maintenance. (setType() & explain())
 *
 * @author locasto@cs.columbia.edu
 *
*/
public class PolicyException extends Exception{

    public static final int DEFAULT_ERROR = 100;
    public static final int UNABLE_TO_LOAD_POLICY = 150;
public static final int POLICY_INCONSISTENT = 152;
public static final int NULL_ZONE = 155;
public static final int NULL_POLICY = 156;
public static final int NULL_LOADER = 157;

/** ..others as needed */

private int type = PolicyException.DEFAULT_ERROR;

public PolicyException( String m ){
super( m );
}

public PolicyException( String m, int t ){
  super( m );
  setType( t );
}

public void setType( int t ){
```

```
   switch( t ){
      case PolicyException.UNABLE_TO_LOAD_POLICY : type = t; break;
      default: type = PolicyException.DEFAULT_ERROR;
   }
}

public String explain( int error ){

switch( error ){
case DEFAULT_ERROR:

break;

case UNABLE_TO_LOAD_POLICY:

break;

case NULL_ZONE:

break;
default: return "Unrecognized error code ["+error+"]";
}
return "";
}

}

/**********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: March 27, 2003
*
**********************************************/

package spcl.data;

/**
 * This class represents the base class for a
```

```
 * generic policy installer. The code generation
 * facilities will essentially create a subclass
 * of this class.
 *
 * @author locasto@cs.columbia.edu
 * @author mb@cs.columbia.edu
 *
*/
public class PolicyInstaller{

  protected PolicyLoader policyLoader;

  public PolicyInstaller(){}

  public void setPolicyLoader( PolicyLoader loader ){
      this.policyLoader = loader;
  }

  /**
   * This method should be overidden.
   *
   * It should have three sections:
   *  <ol>
   *   <li>declarations</li>
   *   <li>parameterization</li>
   *   <li>setup & attaching</li>
   *  </ol>
   * and should finish with
   * <code>policyLoader.load( policy );</code>
   *
  */
  public void doInstall() throws PolicyException{
    if( null==policyLoader )
       throw new PolicyException( "No policy loader defined.", PolicyException.NUL
    return;
  }


}

/*********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
```

```
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: April 10, 2003
*
*********************************************/

package spcl.data;

import java.util.*;


/**
 * A container for rules about principals, groups,
 * and objects. The zone has the master list of
 * these, but the policy holds the rules related
 * to each. Keep this in mind when loading and
 * retiring policy (resolve conflicting rules and
 * make sure to delete a p,g,o as necessary)
 * <p>
 * We don't worry about this restriction right now
 * because we only allow one zone & policy to be
 * active at a time.
 * Eventually, the SPCL should become some kind of XML
 * configuration file so that it can be parsed in a
 * standard way.
*/
public class Policy{

private String name;
private Principal defaultPrincipal=null;
private Hashtable principals = new Hashtable();
private Hashtable groups = new Hashtable();
private Hashtable objects = new Hashtable();

public Policy( String name ){
this.name = name;
}

public String getName(){
return name;
}

public void setDefaultPrincipal( Principal def ){
defaultPrincipal=def;
principals.put( def.getName(), def );
}
```

```
public Principal getDefaultPrincipal(){
return defaultPrincipal;
}

/**
 * add a principal to the hashtable. Note that
 * this principal is not checked until the
 * resolution phase.
*/
public void addPrincipal( Principal p ){
principals.put( p.getName(), p );
}

public Principal getPrincipal( String pname ){
    return ((Principal)principals.get( pname ));
}

public void addGroup( Group g ){
groups.put( g.getName(), g );
}

public void addObject( SystemObject o ){
objects.put( o.getName(), o );
}

public SystemObject getObject( String soname ){
return ((SystemObject)objects.get( soname ));
}

public boolean hasObject( String objectName ){
return objects.containsKey( objectName );
}

public boolean hasPrincipal( String username ){
return principals.containsKey( username );
}

public Enumeration getPrincipals(){
return principals.elements();
}

public Enumeration getObjects(){
return objects.elements();
}

public Enumeration getGroups(){
return groups.elements();
}

public Vector getUserGroups( String username ){
```

```
Group temp = null;
Vector userGroups = new Vector();
Enumeration elements = groups.elements();
while( elements.hasMoreElements() ){
temp = (Group)elements.nextElement();
if( temp.isMember( username ) )
userGroups.addElement( temp );
}
return userGroups;
}


}

/**********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: March 26, 2003
*
**********************************************/

package spcl.data;

/**
 * An interface mechanism for loading new
 * zones and policies.
 *
 * @author locasto@cs.columbia.edu
*/
public interface PolicyLoader{

public static final int SYSTEM_TIME = 333;

/**
 * The primary mechanism for loading a zone in
 * If the zone does not have an attached policy,
 * merely creates the namespace.
*/
```

```
public abstract void load( Zone zone );

/**
 * Loads the provided Policy into the indicated zone.
 *
 * Basically looks up the zone in the PolicyEngine and calls
 *  <code>zone.attach( policy );</code>
 *
 * @throws PolicyException if the provided policy could not be loaded
 *   into the indicated zone (the zone or policy doesn't exist, is malformed,etc)
*/
public abstract void load( Policy policy, String zoneName ) throws PolicyExceptior

/** A policy installer may wish to use this so that it can
 * get a reference to a zone already present in the PolicyEngine.
 * This is implemented in the current version, but only used internally by
 * the policyengine.
*/
public abstract Zone lookupZone( String zoneName );

/**
 * used to retire the named policy
 *
 * @returns false if the policy is not found
 * @returns true if the policy was found and unloaded
 * @throws PolicyException if the policy was found, but could not be unloaded
*/
public abstract boolean unload( String policyName );

/**
 * used to suspend consultation of the named policy
*/
public abstract boolean suspend( String policyName );

/**
 * used to resume consultation of the named (suspended) policy
*/
public abstract boolean resume( String policyName );

//rc( "system", "time", Variable.DATE,

//for special case 'system' and 'time'
public abstract boolean resolveCondition( int sys_var, String relop, String rval )

//for special case 'system' and 'time'
public abstract boolean resolveCondition( int sys_var, String relop, boolean rval

//for special case boolean
public abstract boolean resolveCondition( Variable var, String relop, boolean rval
```

```
//rc( webserver, ip_addr, ,">", "rval", spcl.data.Variable.STRING );
public abstract boolean resolveCondition( Variable var, String relop, String rval
}

/**********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: April 10, 2003
*
**********************************************/

package spcl.data;

import java.io.Serializable;

/**
 * A request from a system object
 *
*/
public class PolicyRequest implements Serializable{

private String content = "";

/* these credentials must be verified */

//username is basically the systemObject name/id
private String username="";
private char [] password;


public PolicyRequest(){
//intially, we hard code this as "root"
//this means that any connection from the
//PolicyEngineClient acts as root and can
//query on any system object.
//note that this is no security at all
//in fact, this info is not even checked on
//the server side. The point of this project
```

```
//is to build a policy language, engine, and client
//and not to worry about solvable authentication
//issues.
//eventually, no system object can query another
//unless a policy specifies it, and every client
//must pass auth credentials upon connection
//so that the engine can trust who it is talking to
}

public void setContent( String c ){
content = c;
}

public String getContent(){
return content;
}

public String toString(){
return content;
}

}

/*********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: March 27, 2003
*
*********************************************/

package spcl.data;

import java.io.Serializable;

public class PolicyResponse implements Serializable{

private String content = "";
```

```
/** the associated request */
private PolicyRequest request = null;

public PolicyResponse(){
}

public PolicyResponse( PolicyRequest request ){
this.request = request;
}

public void setContent( String c ){
content = c;
}

public String toString(){
return content;
}

}

/**********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: April 10, 2003
*
**********************************************/

package spcl.data;

import java.util.Vector;
import java.util.Enumeration;

/**
 * An object representing a Principal to the
 * policy engine. It has a cache for log or
 * notify events. Some background thread should
 * actually run through this cache periodically
 * and use the contact information to actually
```

```
 * forward it to the user.
 *
 * @author locasto@cs.columbia.edu
 *
*/
public class Principal{

  private String name;
  private Vector aliases = new Vector();
  private Vector groups = new Vector();
  private String emailaddr ="";
  private String phone = "";

  //@deprecated
  private String url="";

  private Vector rules = new Vector( 25 );

  private Vector notifyBuffer = new Vector( 20 );

  public Principal( String id ){
    this.name = id;
  }

  public String getName(){
    return name;
  }

  public void addAlias( String a ){
    aliases.add( a );
  }

  public void addGroup( Group g ){
     groups.add( g );
  }

  public void addRule( Rule r ){
    rules.add( r );
  }

  public int getNumRules(){
    return rules.size();
  }

  public Enumeration getRules(){
      return rules.elements();
  }

  public void setEmailaddr( String e ){
    emailaddr = e;
```

```
  }
  public String getEmailaddr(){
    return emailaddr;
  }

  public void setPhone( String p ){
    phone = p;
  }
  public String getPhone(){
    return phone;
  }

  /** @deprecated with no replacement */
  public void setURL( String u ){
   url = u;
  }

  public Vector getGroups(){
    return groups;
  }

  public boolean hasAlias( String a ){
     return (aliases.contains( a ));
  }

  public boolean inGroup( Group group ){
     return (groups.contains(group));
  }

}

/*********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: May 9, 2003
*
*********************************************/
```

```
package spcl.data;

import java.util.Vector;

/**
 * The ultra important Rule class.
 *
 * PolicyEngine collects an array of values from
 *  evaluate(). If this array contains oneof either
 *  DENY or ALLOW and the rest WISHY-WASHY, policy is
 *  consistent and returns the DENY or ALLOW. If there is
 *  a mixture, the policy is inconsistent and returns DENY.
 *
 *
 * @author locasto@cs.columbia.edu
*/
public class Rule{

  public static final int ALLOW = 100;
  public static final int WISHY_WASHY = 50;
  public static final int DENY = 200;

  public static final int STAR = 1024;

  private String name;
  private Vector firings = new Vector();
  private Vector conditions = new Vector();
  private boolean active = false;
  private boolean appliesToAll=false;
  private Principal principal = null;
  private Group group = null;
  private SystemObject so=null;
  private Action action = null;
  private int response;

  public Rule( String n ){
    this.name = n;
  }

  public String getName(){
    return this.name;
  }

  public void addFiring( Firing f ){
    if( f == null ){
       return;
    }else{
       firings.addElement( f );
    }
  }
```

```
/**
 * Set the principal for this rule.
 *@deprecated - replaced by the setOwner() method
*/
public void setPrincipal( Principal p ){
  this.principal = p;
}

public void setOwner( Principal p ){
  this.principal = p;
}

public void setOwner( Group g ){
  this.group = g;
}

public void setObject( SystemObject sysObj ){
  this.so = sysObj;
}

public void setAccess( int res ){
  System.err.println( name+" access is "+res );
  switch( res ){
     case Rule.DENY:
         response=Rule.DENY;
         break;
     case Rule.ALLOW:
          response=Rule.ALLOW;
         break;
     default:
        response=Rule.DENY;
  }
  System.err.println( name+" access is "+response );
}

public void setAction( int blanket ){
    //for any object, if this rule has matched, return response
    if( blanket == spcl.data.Rule.STAR )
    appliesToAll=true;
action = new Action( "APPLY_ALL" );
action.setActionPattern( ".*" );
}

public void setAction( Action a ){
  this.action = a;
}

public void setActive( boolean b ){
  active = b;
```

```
    if( active ) System.err.println( "\t>> "+name+" is active." );
    else System.err.println( "\t>> "+name+" is deactivated." );
  }

  public boolean isActive(){
    return active;
  }

  //give this some args
  //params = Vector of valued-Variables
  //@returns DENY.. if rule says deny
  //@returns ALLOW .. if rule says allow
  //@returns NO_DEC .. if rule does not apply to object or
  //   conditions DO NOT UNIFY
  public int evaluate( Vector params, String actionString, SystemObject clientOn )
String regexp = action.getActionPattern();
Vector unifiedConditions=null;
int psize = params.size();
int csize = conditions.size();

if( psize!=csize && csize!=0 ){
System.err.println( "\t\t\t>> rule.evaluate()::psize("+psize+")!=csize("+csize+")"
return Rule.WISHY_WASHY;
}

//more specific first
if( so!=clientOn ){
System.err.println( "\t\t\t>> rule.evaluate()::so!=current.object " );
return Rule.WISHY_WASHY;
}

if( !subExpression( regexp, actionString ) ){
System.err.println( "\t\t\t>> actionString is !subexp( regexp )" );
return Rule.WISHY_WASHY;
}
if( (unifiedConditions = unifyConditions( params ))==null ){
System.err.println( "\t\t\t>> unifiedConditions=null " );
return Rule.WISHY_WASHY;
}else{
//go through unifiedConditions and see if all eval to true
if( evaluateConditions( unifiedConditions ) ){
return response;
}else{
return Rule.WISHY_WASHY; //should be DENY?
}
}
  }

  /** @requires Java 1.4 */
  private boolean subExpression( String regexp, String actionString ){
```

```
    boolean match=false;
try{
match = actionString.matches( regexp );
}catch( Exception REparseEx ){
return (actionString.equals( regexp ));
}
return match;
  }

  /** given the unified conditions, we can now evaluate them to a
   * boolean AND of each condition. We had their relop and their rval,
   * and we got their lvalue from unifyConditions()
   *
  */
  private boolean evaluateConditions( Vector uconds ){
    boolean truth=true;
boolean tempTruth=false;
Condition tempCond = null;
for( int i=0;i<uconds.size();i++ ){
tempCond = (Condition)uconds.elementAt(i);
tempTruth = tempCond.eval();
System.err.println( "\t\t\t>>> cond["+tempCond.toString()+"]="+tempTruth );
truth = (truth && tempTruth);
}
return truth;
  }

  /** call from rule.evaluate()
   * find and return the conditions that match
   * need to determine what is in params first!!!
   * basically, it'll just be AND x=6 AND y=true
   *  so in the PolicyEngine, we have to correctly parse this
   *  and create the appropriately typed variables
   * must unify on TYPE and NAME...then <-param sets the value
   *
   * add to results the first condition that unifies with a
   *  given param, and go on to next Condition
  */
  private Vector unifyConditions( Vector params ){
Vector results=new Vector();
Condition tempCond=null;
Variable tempParam=null;
int tempType=-1;

//tempCond.var.type will not change
//only tempCond.var.svalue,bvalue,nvalue
for( int i=0;i<conditions.size();i++ ){
tempCond = (Condition)conditions.elementAt( i );
System.err.print( "\t\t\t>> unify("+tempCond.var.getName()+", ");
for( int j=0;j<params.size();j++ ){
```

```
tempParam = (Variable)params.elementAt(j);
System.err.print( tempParam.getName()+" );" );
if( tempParam.getName().equals( tempCond.var.getName() )
    && (tempParam.getType()==tempCond.var.getType())
){
System.err.print( " = success\n" );
//condition lvalue matched in type and name with a param
//unify
tempType = tempParam.getType();
switch( tempType ){
case Variable.NUMBER:
tempCond.var.setValue( ""+tempParam.getNumberValue() );
break;
case Variable.STRING:
tempCond.var.setValue( tempParam.getStringValue() );
break;
case Variable.BOOLEAN:
tempCond.var.setValue( ""+tempParam.getBooleanValue() );
break;
}
results.add( tempCond );
}else{
System.err.print( " = fail\n" );
}
}
System.err.println( "" );
}

//must have supplied a value for every condition!!
if( results.size() != conditions.size() ) return null;
return results;
  }

  public void addCondition( Variable var, String relOp, boolean rval ){
conditions.add( new Condition( var, relOp, rval ) );
  }

  public void addCondition( Variable var, String relOp, String rval ){
conditions.add( new Condition( var, relOp, rval ) );
  }

  public synchronized void fire(){
    int allfirings = firings.size();
    Firing f = null;
    for( int i=0;i<allfirings;i++ ){
      f = ((Firing)firings.elementAt(i));
      if( f!=null )
        f.doFire();
    }
  }
```

```
}


/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class SPCLCompilerException extends Exception {

    public SPCLCompilerException() {
        super();
    }

    public SPCLCompilerException(String s) {
        super(s);
    }

}

import antlr.collections.AST;
import java.io.*;

public class SPCLCompiler {
    public static void main(String[] args) {
        ErrorHandler eh = new ErrorHandler();
        try {
    if (args.length != 1 || !args[0].endsWith(".spcl")) {
System.out.println("usage: java SPCLCompiler filename.spcl");
System.exit(0);
    }

    String infile = args[0];
    String outfile = args[0].replaceFirst(".spcl", "Installer.java");

    InputStream in = new FileInputStream(infile);
    PrintStream out = new PrintStream(new FileOutputStream(outfile));

            L lexer = new L(in);
            P parser = new P(lexer);
            parser.policydef();
            AST ast = parser.getAST();
            SymbolTable s = new SymbolTableBuilder( eh ).policydef( ast );
            new TypeChecker( eh ).policydef( ast, s );
            new CodeGenerator(out, eh).policydef( ast, s );
```

```
        } catch (SPCLCompilerException spcle) {
            System.out.println("SPCL error: " + spcle.getMessage());
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}


/*********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: March 28, 2003
*
*********************************************/

package spcl.policyengine;

import java.util.*;
import java.net.*;
import java.io.*;

public class StopTask{

   public StopTask( String filename ) throws Exception{
     Socket client;
 ObjectInputStream oin;
 ObjectOutputStream oout;
     String ipaddr,message;
 int port;

 Properties configs = new Properties();
 configs.load( new FileInputStream( new File( filename ) ) );
 port = Integer.parseInt( configs.getProperty( "spcl.console.port", "3781" ) );
 message = configs.getProperty( "spcl.shutdownmessage", "" );
 configs=null;
System.out.println( "read config info from "+filename );
```

```
client = new Socket( InetAddress.getLocalHost(), port );
oout = new ObjectOutputStream( client.getOutputStream() );
oout.flush();
oin = new ObjectInputStream( client.getInputStream() );

System.out.println( "connected to PolicyEngine console..." );

oout.writeObject( message );
oout.flush();

System.out.println( "wrote shutdown message..." );

//oin.close();
//oout.close();
//client.close();

System.out.println( "StopTask done." );

   }

}

import java.util.*;
import java.util.regex.*;
import java.io.PrintStream;

/**
 * Title:
 * Description:
 *
 *      A SymbolTable is a place for storing information about
 *      variable IDs and the corresponding types and scopes.
 *
 *      Type is represented by static integer constants.
 *
 *      The way this class represent scopes is to use a new
 *      SymbolTable for each scope. Thus, each instance of
 *      SymbolTable is associated with a given scope of the
 *      original program, and each instance of SymbolTable
 *      has a pointer to the outer scope symboltable and all
 *      pointers to its inner scope symboltables.
 *
 *      Scopes are changed through these methods:
 *          public void enterScope(String ID)
 *          public void escapeScope()
 *          public void resetToOuterMostScope()
 *        ID is used for identifying the object/principal... or other
 *      structure that the scope corresponds to. For example, to
 *      enter the scope of object fridge, one would call enterScope("fridge")
```

```
 *        The last method is used when there is a need to walk through the
 *      original program/AST several times and if error may occur during
 *      tree walking. If error occurs, then there is a possibility for
 *      the symbol table to be stucked at the middle of some inner scope, which
 *      would be undesirable.
 *
 *      Variables are manipulated through these methods:
 *          public void registerVariable(String id)
 *              throws VariableConflictException
 *          public int getType(String id)
 *              throws VariableNotFoundException
 *          public booelan hasID(String id)
 *
 *      Forward Reference:
 *          SPCL permits forward reference. When translating it to langauage
 *          like JAVA where forward reference is not permissible, there is a
 *          need to reorder codes. In order to solve this problem, a notion
 *          of pending statement is used. The idea is to Store all statements,
 *          that requires the declaration of certain JAVA objects, to a buffer.
 *          Once the object is declared, all pending statements shall be
 *          emitted appropriately.
 *          This pending statement is stored as an attribute of SymbolTable
 *          Entry, @see private class Entry for further information.
 *          The following public methods relates to this funtionality:
 *
 *          public void printIfNotForwardReference(String baseID,
 *              String targetID, PrintStream out, String statement);
 *          public void getPendingStatement(String id);
 *
 *
 *
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class SymbolTable {

    /************
     * debug mode
     */
    private boolean DEBUG = false;

    /*******************
     * outer_scope is the innest outer scope reachable from
     * the current scope.
     *
     * This is used to resolve symbols that does not exists
     * in the current scope.
```

```
 */
protected SymbolTable outer_scope = null;

/********************
 * A scope name is the name of the variable that
 * leads to this scope.
 *
 * For example,
 *
 * object web {
 *      string a = "def";
 * }
 *
 * The scope that contains variable "a" is object web.
 * Hence the name of the scope for variable a is called "web".
 */
protected String scope_name = null;

/************************
 * The scope that is currently dealing with.
 */
protected SymbolTable current_scope = this;


protected SymbolTable root = null;


/************************
 * an array of symbol table entries.
 *
 * @see
 *      private class Entry
 * for further information
 */
protected Vector symbolEntry = new Vector(10);

/************************
 * number of variables registered. this is used to generate
 * a distinct internal value for determining whether a variable
 * A is a forward reference of variable B.
 */
protected int number_of_variable_registered = 0;

/*************
 * a list of SPCL types
 */
public static final int TYPE_ZONE = 0;
public static final int TYPE_POLICY = 1;
public static final int TYPE_GROUP = 2;
public static final int TYPE_PRINCIPAL = 3;
```

```
    public static final int TYPE_PRINCIPAL_ALIAS = 4;
    public static final int TYPE_OBJECT = 5;
    public static final int TYPE_NUMBER = 6;
    public static final int TYPE_STRING = 7;
    public static final int TYPE_BOOLEAN = 8;
    public static final int TYPE_ACTION = 9;
    public static final int TYPE_RULE = 10;
    public static final int TYPE_ERROR = 11;

    /*
        for debugging only
     */
    private static final String[] type_name = {
        "TYPE_ZONE",    "TYPE_POLICY",
        "TYPE_GROUP",   "TYPE_PRINCIPAL",
        "TYPE_PRINCIPAL_ALIAS", "TYPE_OBJECT",
        "TYPE_NUMBER",  "TYPE_STRING",
        "TYPE_BOOLEAN", "TYPE_ACTION",
        "TYPE_RULE",    "TYPE_ERROR",
    };

    /*******
     * constructor
     */
    public SymbolTable() {
    }

    /***************
     * constructor of an inner symboltable.
     *
     * When a program enters a scope, a new SymbolTable
     * has to be created to store variables in the scope.
     *
     * @outer_scope is used to resolve symbols that's absent
     * in the innermost scope.
     *
     * @scope_name is used to regenerate the complete path
     * for naming a specific variable from the global view.
     */
    private SymbolTable(SymbolTable outer_scope, String scope_name) {
        this.scope_name = scope_name;
        this.outer_scope = outer_scope;
    }

    public void setRoot() {
root = this.current_scope;
    }

    /*****************************************
     * Return a string representation of the type.
```

```
 * Mainly intended to be used by error reporting.
 */
public static String typeToName(int type) {
    if ( (type>11) || (type<0) ) {
        return type_name[SymbolTable.TYPE_ERROR];
    }
    return type_name[ type ];
}

/**************
 * return true if the specified id is reachable
 * from the current scope.
 * i.e. it exists either in the current scope or
 * one of the outer scope.
 */
public boolean hasID(String id) {
    SymbolTable scope = this.current_scope;
    while ( scope != null ) {
        try {
            scope.getLocalEntry(id);
            return true;
        } catch (VariableNotFoundException e) {}
        scope = scope.outer_scope;
    }
    return false;
}

/*************
 * @return the entry specified by id, if the id is not found
 *      in local scope, outer scope would be consulted. This
 *      proceed on indefinitely util the outermost scope is
 *      reached or a corresponding variable is found.
 */
private Entry getEntry(String id) throws VariableNotFoundException {
    SymbolTable scope = this.current_scope;
    while ( scope != null ) {
        try {
            Entry e = scope.getLocalEntry(id);
            return e;
        } catch ( VariableNotFoundException e ) {
        }
        scope = scope.outer_scope;
    }
    throw new VariableNotFoundException(id);
}

/*************
 * @return the entry specified by id, only search
 *      for entries that are in the scope represented
 *      by this symboltable without consulting any
```

```
 *      outer scope
 */
private Entry getLocalEntry(String id) throws VariableNotFoundException {
    Vector v = this.symbolEntry;
    Entry e;
    for (int i=0; i<v.size(); i++) {
        e = (Entry)v.elementAt(i);
        if ( e.getID().equals(id) ) {
            return e;
        }
    }
    throw new VariableNotFoundException(id);
}

/*******************
 * Register a variable in the current scope.
 *
 * If this creates any conflicts, a VariableConflictException
 * would be thrown
 */
public void registerVariable(String id, int type) throws VariableConflictExcep
    SymbolTable scope = (type==TYPE_PRINCIPAL_ALIAS)?this.current_scope.outer_
    if (DEBUG) {System.out.println("registering " + id + "," + type + "\tin\t'
    try {
        scope.getLocalEntry(id);
        throw new VariableConflictException(id);
    } catch (VariableNotFoundException e) {
        if (type==TYPE_PRINCIPAL_ALIAS) {
            scope.symbolEntry.add(
                new Entry(
                    id,
                    type,
                    this.current_scope.scope_name,
                    this.number_of_variable_registered
                )
            );
        } else {
            scope.symbolEntry.add(
                new Entry(
                    id,
                    type,
                    scope,
                    this.number_of_variable_registered
                )
            );
        }
        this.number_of_variable_registered ++;
    }
}
```

```
    /*******************
     * Enter the scope specified by the given ID
     */
    public void enterScope(String ID) throws VariableNotFoundException {
        if (DEBUG) {System.out.println("enter scope " + ID );}
        Entry e = this.current_scope.getLocalEntry(ID);
        this.current_scope = e.getInnerScope();
    }

    /***************
     * Escape from the current scope to the innermost
     * reachable outer scope
     */
    public void escapeScope() {
        if (DEBUG) {System.out.println("escape scope " + this.current_scope.scope_
        if ( this.current_scope.outer_scope != null ) {
            this.current_scope = this.current_scope.outer_scope;
        }
    }


    public String lookupVar(String parent, String name, String separator) {
String ret = null;
try {
    SymbolTable ptr = root;

    Entry e = ptr.current_scope.getLocalEntry(parent);
    ptr = e.getInnerScope();

       ret = ptr.getCanonicalName(name, separator);
} catch (VariableNotFoundException f) {
    System.out.println("Variable not found: " + name + "\n");
}
return ret;

    }



    /****************
     * Reset the current scope to the outer most scope.
     *
     * This is useful when certain errors were found when
     * parsing the AST, and the SymbolTable is stucked at
     * the middle of some scope.
     */
    public void resetToOuterMostScope() {
        this.current_scope = this;
```

```
}

/***************
 * Retrieve the type that bound to id in
 * the current scope.
 *
 * If id is not found from current scope, throw
 * a VariableNotFoundException
 */
public int getType(String id) throws VariableNotFoundException {
    return this.getEntry(id).getType();
}

/***************
 * Retrieve the type that bound to the variable
 * specified by the given path.
 *
 * For example, zoneA.policyP.fridge.a, is a path.
 * A path doesn't has to be complete, e.g. fridge.a is
 * also acceptable, as long as the relative path could
 * be reached from the current scope.
 */
public int getType(String[] path) throws VariableNotFoundException {
    try {
        Entry entry = this.getEntry(path[0]);
        for (int i=1; i<path.length; i++) {
            entry = entry.getInnerScope().getLocalEntry(path[i]);
        }
        return entry.getType();
    } catch (VariableNotFoundException e) {
        String s = path[0];
        for (int i=1; i<path.length; i++) {
            s+="."+path[i];
        }
        throw new VariableNotFoundException(s);
    }
}

/*******************
 * If the id has the expected type, this method
 * should return peacefully. Otherwise, exception
 * should be thrown
 */
public void matchType(String id, int expected_type)
    throws VariableNotFoundException, TypeMisMatchException {
    int type = this.getEntry(id).getType();
    if ( type == TYPE_PRINCIPAL_ALIAS ) {
        type = TYPE_PRINCIPAL ;
    }
    if ( type != expected_type ) {
```

```
            throw new TypeMisMatchException(expected_type, type);
        }
    }

    /*********************
     * If targetID is defined later than baseID in the original
     * source code, targetID would not be defined at the moment
     * baseID set some actions or pointers to targetID in the
     * translated JAVA code. In that case, the statements shall
     * be stored to a buffer, instead of dumping to the output
     * directly.
     * These pending statements may be retrieved by method
     * getPendingStatement(String id);
     *
     * If it happends that targetID is defined prior to baseID,
     * the statement shall be directed to <code>out</code>
     * immediately.
     *
     * If targetID is not reachable from the current scope,
     * VariableNotFoundException would be thrown.
     *
     * @baseID      the variable currently being examined
     * @targetID    the variable to be refered to
     * @out         an output stream for emitting statement
     *              if targetID is not a forward reference of
     *              baseID
     * @statement   the statement to be printed
     *
     * @see getPendingStatement(String);
     */
    public void printIfNotForwardReference(String baseID, String targetID, PrintSt
        throws VariableNotFoundException {

        Entry b = this.getEntry( baseID );
        Entry t = this.getEntry( targetID );
        if ( b.appearBefore( t ) ) {
            t.appendStatement( statement );
        } else {
            out.print( statement );
        }

    }

    /*****************
     * Retrieve all pending statements that requires the declaration of
     * the given variable id.
     *
     * @see printIfNotForwardReference(String,String,PrintStream,String);
     */
    public String getPendingStatement(String id) throws VariableNotFoundException
```

```
    return this.getEntry(id).getPendingStatement();
}

/****
 * @return true if the specified id is a reference to
 *      an active rule. i.e. it is a direct rule under
 *      principal/default/group/ as opposed to a rule
 *      update.
 */
public boolean isActiveRule(String id) {
    try {
        Entry e = this.current_scope.outer_scope.getLocalEntry(this.current_sc
        return (e.getType() != this.TYPE_RULE );
    } catch (Exception e) {
        return false;
    }
}

/**************
 * @return the name that is bound to the current
 *      active scope
 *
 * e.g.
 * principal John {
 *    allow "action" on recycleBin {    // assume this rule has id 1
 *        deny * on recycleBin;         // assume this rule has id 2
 *    }
 * }
 *
 * Suppose the AST is now examining the variable "2",
 * this method should return "John".
 *
 * @see public String getCurrentInnermostScopeName();
 */
public String getCurrentScopeName() {
    SymbolTable scope = this.current_scope;
    try {
        while ( scope.outer_scope.getLocalEntry(scope.scope_name).getType() ==
            scope = scope.outer_scope ;
        }
    } catch (Exception e) {}
    return scope.scope_name;
}

/**************
 * @return the name that is bound to the current
 *      active scope
 *
 * e.g.
 * principal John {
```

```
 *    allow "action" on recycleBin {    // assume this rule has id 1
 *        deny * on recycleBin;         // assume this rule has id 2
 *    }
 * }
 *
 * Suppose the AST is now examining the variable "2",
 * this method should return "1".
 *
 * @see public String getCurrentScopeName();
 */
public String getCurrentInnermostScopeName() {
    return this.current_scope.scope_name;
}

/***********************************
 * @return  the ID of the principal that the given
 *      alias is refering to.
 *      An exception would be thrown if the given
 *      variable does not exist or it is not an alias.
 */
public String getAliasTarget(String alias_id) throws VariableNotFoundExceptior
    return this.getEntry(alias_id).getAliasTarget();
}

/*****************
 * Try to register a regex for the action specified by
 * action_id. If the action is not reachable from current
 * local scope, a VariableNotFoundException would be thrown.
 *
 * If action_id is not refering to a variable of TYPE_ACTION,
 * an SPCLCompilerException is thrown.
 *
 * However, none of these shall ever be thrown if the treeWalker
 * is written appropriately. There is no way that a user could
 * write an SPCL program to cause this method to throw an Exception.
 *
 * @action_id   the variable ID that refers to the action
 * @s           the regular expression of an action to be
 *              registered under action_id
 */
public void registerActionRegex(String action_id, String s)
    throws VariableNotFoundException, SPCLCompilerException
{
    this.current_scope.getLocalEntry(action_id).appendActionRegex(s);
}

/****************
 * throws a NoRegexDefinedException if the specified object
 * has not defined any action regex.
 *
```

```
 * throws a VariableNotFoundException if the specified id
 * could not be found.
 *
 * throws a TypeMisMatchException if object_id is not refering
 * to a variable of type object.
 *
 * Besides that, this method doesn't do anything at all.
 */
public void assertActionExist(String object_id)
    throws NoRegexDefinedException, VariableNotFoundException,
    TypeMisMatchException
{
    Entry e = this.getEntry(object_id);
    if ( e.getType() != TYPE_OBJECT ) {
        throw new TypeMisMatchException( TYPE_OBJECT, e.getType() );
    }
    Vector v = e.getInnerScope().symbolEntry;
    for (int i=0; i<v.size(); i++) {
        Entry ae = (Entry)v.elementAt(i);
        if ( (ae.getType() == TYPE_ACTION)
            && (ae.containSomeAction()) ) {
            return;
        }
    }
    throw new NoRegexDefinedException(object_id);
}

/************************************
 * Try to match a given action with at least one of the
 * regex defined under the target object specified by
 * object_id, if no such match could be found, throw
 * an InvalidActionException, otherwise, return peacefully.
 *
 * In addition, if object_id is not defined, a
 * VariableNotFoundException is thrown.
 *
 * If object_id is not refering to an object, a
 * TypeMisMatchException is thrown
 *
 * @action  the action to be examined
 * @object_id   the id of target object
 */
public void matchAction(String object_id, String action)
    throws InvalidActionException, VariableNotFoundException,
    TypeMisMatchException
{
    Entry e = this.getEntry(object_id);
    if ( e.getType() != TYPE_OBJECT ) {
        throw new TypeMisMatchException(TYPE_OBJECT, e.getType());
    }
```

```
        Vector v = e.getInnerScope().symbolEntry;
        for (int i=0; i<v.size(); i++) {
            try {
                ((Entry)v.elementAt(i)).matchAction(action);
                return;
            } catch (InvalidActionException iae) {}
        }
        throw new InvalidActionException(object_id, action);
    }


//      public void matchActionName(String s) throws InvalidActionException {
//   if ( this.actionRegex != null ) {
//       for (int i=0; i<this.actionRegex.size(); i++) {
//   Pattern p = (Pattern)this.actionRegex.elementAt(i);
//   if ( p.matcher(s).matches() ) {
//       return;
//   }
//       }
//   }
//   throw new InvalidActionException(ID, s);
//      }

    public void matchActionName(String object_id, String name)
        throws InvalidActionException, VariableNotFoundException,
        TypeMisMatchException
    {
        Entry e = this.getEntry(object_id);
        if ( e.getType() != TYPE_OBJECT ) {
            throw new TypeMisMatchException(TYPE_OBJECT, e.getType());
        }
        Vector v = e.getInnerScope().symbolEntry;
        for (int i=0; i<v.size(); i++) {
    if (((Entry)v.elementAt(i)).getID().equals(name))
return;
        }
        throw new InvalidActionException(object_id, name);
    }



    /**************
     * @return      the canonical name for the given id
     *              with each level of ID seperated by
     *              given seperator.
     *
     * @id  the id visible to the local scope. e.g.
     *      abc in the case of foo.def.abc
     * @seperator   the string used to concatenate the
     *              path. e.g. setting seperator to "_"
```

```
     *              would change "foo.def.abc"
     *              to "foo_def_abc"
     */
    public String getCanonicalName(String id, String seperator) {
        SymbolTable scope = this.current_scope;
        String name = id;
        while ( scope != null && scope.scope_name != null ) {
            name = scope.scope_name + seperator + name;
            scope = scope.outer_scope;
        }
        return name;
    }

    public String getCanonicalName(String seperator) {
        SymbolTable scope = this.current_scope;

        String name = this.current_scope.scope_name;
scope = scope.outer_scope;

        while ( scope != null && scope.scope_name != null ) {
            name = scope.scope_name + seperator + name;
    scope = scope.outer_scope;
        }
        return name;
    }


    /********
     * @return a string representation of this SymbolTable
     *
     * for debugging only
     */
    public String toString() {

        if ( (this.current_scope != this) && (this.current_scope != null) ) {
            return this.current_scope.toString();
        }

        String s = "{";
        try {
            s += "\n";
            for (SymbolTable scope = this; scope.outer_scope!=null; scope = scope.
                s += "\t";
            }
            for (int i=0; i<this.symbolEntry.size(); i++) {
                s += this.symbolEntry.elementAt(i).toString();
                for (SymbolTable scope = this; scope.outer_scope!=null; scope = sc
                    s += "\t";
                }
            }
```

```
    } catch (Exception e) {
        s += "[Error Table]";
    }
    s += "}";
    return s;
}

/*********************
 * An Entry is a symbol table entry that contains information
 * about a certain variable, such as
 *
 * - the ID of the variable
 * - the type of the variable
 * - the alias_target if the given variable is an alias
 * - the scope that contains this variable,
 * - the new scope that is generated by this variable
 *      (in the case of object/principal/group/...).
 * - pendingStatements : there is a problem when converting
 *      from language like SPCL that permits forward reference
 *      to language like JAVA that doesn't permits it. In order
 *      to solve this problem, a notion of pending statement is
 *      used for reordering codes. Whenever the translated
 *      statement requires some targets to be defined in advance,
 *      the statement is appended to the target. Right after the target
 *      is defined, all pending statements shall be emitted
 *      appropriately.
 * - chronological order, a non-decreasing value that is used to determine
 *      forward reference
 */
private class Entry {
    private String ID;
    private int type;
    private int chronological_order;
    private SymbolTable existing_scope = null;
    private SymbolTable inner_scope = null;
    private String alias_target = null;
    private String pendingStatement = "";
    private Vector actionRegex = null;

    /********
     * @ID      the ID of the variable for this entry
     * @type    the type of this variable
     * @alias_target    the real id that this alias refers to
     */
    public Entry(String ID, int type, String alias_target, int chronological_o
        this.ID = ID;
        this.type = type;
        this.existing_scope = null;
        this.alias_target = alias_target;
        this.chronological_order = chronological_order;
```

```
    }

    /**********
     * @ID      the ID of the variable for this entry
     * @type    the type of this variable
     * @existing_scope  the scope that this variable belongs to
     */
    public Entry(String ID, int type, SymbolTable existing_scope, int chronolo
        this.ID = ID;
        this.type = type;
        this.existing_scope = existing_scope;
        this.alias_target = null;
        this.chronological_order = chronological_order;
    }
    public void appendActionRegex(String s) throws SPCLCompilerException {
        if ( this.type != TYPE_ACTION ) {
            throw new SPCLCompilerException("Registering a regex under " + thi
        }
        if ( this.actionRegex == null ) {
            this.actionRegex = new Vector(5);
        }
        this.actionRegex.add( Pattern.compile(s) );
    }
    /****************
     * throw an Exception if s does not match with any
     * defined regex under this object. otherwise,
     * return peacefully
     */
    public void matchAction(String s) throws InvalidActionException {
        if ( this.actionRegex != null ) {
            for (int i=0; i<this.actionRegex.size(); i++) {
Pattern p = (Pattern)this.actionRegex.elementAt(i);
if ( p.matcher(s).matches() ) {
                    return;
}
            }
        }
        throw new InvalidActionException(ID, s);
    }
    /****************
     * return true if this variable has some action
     * regex defined
     */
    public boolean containSomeAction() {
        return (this.actionRegex!=null);
    }

    public void appendStatement(String s) {
        this.pendingStatement = this.pendingStatement + s;
    }
```

```
        /**********
         * return true if this variable appears before
         * e.ID in the SPCL source program
         */
        public boolean appearBefore( Entry e ) {
            return this.chronological_order < e.chronological_order ;
        }
        public String getPendingStatement() {
            return this.pendingStatement;
        }
        public String getAliasTarget() {
            return this.alias_target;
        }
        public String getID() {
            return this.ID;
        }
        public int getType() {
            return this.type;
        }
        /******
         * @return a new scope that is specified by this variable
         */
        public SymbolTable getInnerScope() {
            if ( this.inner_scope == null ) {
                this.inner_scope = new SymbolTable(this.existing_scope, this.ID);
            }
            return this.inner_scope;
        }

        /***********
         * @return a string representation of this Entry
         *
         * for debugging only
         */
        public String toString() {
            String s = "(";
            try {
                s += type_name[this.type];
                s += ",";
                s += this.ID;
                if ( this.alias_target != null ) {
                    s += "-->";
                    s += this.alias_target;
                }
                if ( this.actionRegex != null ) {
                    s += " actions : [";
    Vector v = this.actionRegex;
    for (int i=0; i<v.size(); i++) {
s += ((Pattern)v.elementAt(i)).pattern();
    }
```

```
    s += "]";
                }
                if ( this.inner_scope != null ) {
                    s += this.inner_scope.toString();
                }
            } catch (Exception e) {
                s += "[Error Entry]";
            }
            s += ")\n";
            return s;
        }
    }

}
```

```
/*********************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: May 8, 2003
*
*********************************************/

package spcl.data;

import java.util.*;
```

```
/**
 * An object representing a system in the network.
 * It could have a background thread to notify
 * the real object of log or notify side effects.
 * Here we merely provide a cache for that info.
 *
 * @author locasto@cs.columbia.edu
 *
*/
public class SystemObject{

private String url;
private String name;
private Hashtable variables = new Hashtable();
private Hashtable actions = new Hashtable();

public SystemObject( String n ){
  this.name = n;
}

public String getName(){
  return name;
}

public void addVariable( Variable v ){
variables.put( v.getName(), v );
}

public Variable getVariable( String vid ){
return ((Variable)variables.get( vid ));
}

public void addAction( Action a ){
actions.put( a.getName(), a );
}

public Action getAction( String n ){
return ((Action)actions.get( n ));
}


}

import antlr.collections.AST;

public class TestGrammar {
    public static void main(String[] args) {
        ErrorHandler eh = new ErrorHandler();
        try {
```

```
            L lexer = new L(System.in);
            P parser = new P(lexer);
            parser.policydef();
            AST ast = parser.getAST();
            SymbolTable s = new SymbolTableBuilder( eh ).policydef( ast );
            new TypeChecker( eh ).policydef( ast, s );
            new CodeGenerator(System.out, eh).policydef( ast, s );

        } catch (SPCLCompilerException spcle) {
            System.out.println("SPCL error: " + spcle.getMessage());
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}



/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class TypeMisMatchException extends SPCLCompilerException {

    public TypeMisMatchException() {
        super();
    }

    public TypeMisMatchException(String s) {
        super(s);
    }

    /********************
     * @expected_type   the expected type of a variable
     * @type_found      the actual type of the variable
     */
    public TypeMisMatchException(int expected_type, int type_found) {
        super(
            "expecting " + SymbolTable.typeToName(expected_type)
            + ", found " + SymbolTable.typeToName(type_found)
        );
    }

}
```

```
/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author
 * @version 1.0
 */

public class VariableConflictException extends SPCLCompilerException {

    public VariableConflictException() {
        super();
    }

    public VariableConflictException(String s) {
        super(s);
    }

}

/*************************************************
* Columbia University CS Department
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: March 27, 2003
*
*************************************************/

package spcl.data;

/**
 * An object representing optional variables
 * associated with a principal or systemobject.
 *
 * @author locasto@cs.columbia.edu
 *
*/
public class Variable{
```

```
public static final int STRING = 100;
public static final int BOOLEAN = 102;
public static final int NUMBER = 104;

private String name = "null";
private int type = Variable.STRING;
private String svalue="";
private int nvalue=0;
private boolean bvalue=false;

public Variable(){}

public Variable( String n, int t, String v ){
this.name = n;
setType( t );
setValue( v );
}

public String getName(){
return this.name;
}

public int getNumberValue(){ return nvalue; }
public String getStringValue(){ return svalue; }
public boolean getBooleanValue(){ return bvalue; }

public int getType(){ return type; }

public void setType( int t ){
switch( t ){
case Variable.STRING: this.type = t; break;
case Variable.BOOLEAN: this.type = t;  break;
case Variable.NUMBER: this.type = t; break;
default: this.type = Variable.STRING;
}
}

public void setValue( String v ){
if( v==null ){
bvalue=false;
nvalue=0;
svalue="";
}

switch( this.type ){
case Variable.STRING: svalue = v; break;
case Variable.BOOLEAN:
if(v.equals( "true" )){
bvalue=true;
```

```
}else{
bvalue=false;
}
break;
case Variable.NUMBER:
try{
nvalue=Integer.parseInt( v );
}catch(Exception e){
svalue=v;
}
break;
default:
svalue=v;
}
}

public String toString(){
switch( type ){
case Variable.STRING: return svalue;
case Variable.NUMBER: return ""+nvalue;
case Variable.BOOLEAN: return ""+bvalue;
default: return "";
}
}
}


/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:
 * @author Chun Li
 * @version 1.0
 */

public class VariableNotFoundException extends SPCLCompilerException {

    public VariableNotFoundException() {
        super();
    }

    public VariableNotFoundException(String s) {
        super(s);
    }

}

/***********************************************
* Columbia University CS Department
```

```
* Network Security Lab
*
* Open and Survivable Embedded Systems (OASES)
*
* Structured Policy Command Language (SPCL)
*
*
* This software is provided as is without any
* warranty or guarantee of merchantibility
* or fitness for a particular purpose. This
* software is released under the terms of the
* GNU General Public License (GPL).
*
* Last updated: May 10, 2003
*
*********************************************/

package spcl.data;

import java.util.*;


/**
 * The high-level container for a collection of
 * policies.
 *
 * The following is a discussion of the naming
 * scheme for zones, policies, principals, rules,
 * systemobjects, and groups.
 *
 * The general naming scheme is:
 *  rule:zone.policy.group.principal.rulenum
 *
 * thus, 'foo.bar.michael.michael.1' would refer to
 * the first Rule defined in Principal 'michael' in
 * Group 'michael' (each Principal gets its own Group
 * implicitly) in Policy 'bar' under Zone 'foo'.
 *
 * group:zone.policy.group
 * principal:zone.policy.group.principal
 * zone:zone
 * policy:zone.policy
 * action:zone.policy.object.action
 * object:zone.policy.object
 *
 * @author locasto@cs.columbia.edu
 *
*/
public class Zone{
```

```
private String name = "";
private Hashtable policies = new Hashtable();
private Hashtable principals = new Hashtable();
private Hashtable groups = new Hashtable();
private Hashtable rules = new Hashtable();

private int numPolicies=0;

public Zone( String name ){
  this.name = name;
}

public String getName(){
  return name;
}

/** return the first policy in your hash */
public Policy getFirst(){
  return ((Policy)(policies.elements().nextElement()));
}

/**
 * Import the supplied policy into this zone.
 *
 * @param Policy p - the candidate policy
 * @throws PolicyException - if the call to <code>resolve()</code>
 *   fails in any way
 *
*/
public void attach( Policy p ) throws PolicyException{

  resolve( p );

  //we are here only if resolve() returned peacefully
  policies.put( p.getName(), p );
  numPolicies++;

}

public int getNumPolicies(){
  return numPolicies;
}

public boolean isEmpty(){
  return (numPolicies<=0);
}

public Enumeration getPolicies(){
 return policies.elements();
```

```
  }

  public Policy getPolicy( String policyName ){
    return ((Policy)policies.get( policyName ));
  }

  private void resolve( Policy p ) throws PolicyException{
     if(null==p)
       throw new PolicyException( "resolve() given null policy",
                                  PolicyException.NULL_POLICY );

     return;
  }

  public boolean remove( String policyName ){

policies.remove( policyName );
numPolicies--;

    return true;
  }


}
```

10 Conclusion

This section discusses some (anonymized) feedback about the lessons learned from this project and experience. It also includes some words of advice for future project teams.

10.1 Lessons

The lessons learned from this project were fairly wide-ranging, from heavily technical compiler and language design issues to Latex and through to project and team management.

We were in a fairly unique situation from having started the project early, and were able to proceed at a deliberate pace. It was also extremely helpful to have the extra time to iterate over our understanding of the language and decide which features to keep and which features to throw overboard. Even though we had the project under control, there were two days before the project deadline when a massive effort was needed from everyone to push out the final product. The team worked well together and derived great benefit from everyone's knowledge.

A very welcome lesson was the interesting perspective individual developers were to get by having a well–defined interface and developing on either side of it. During the final stages of the project, some developers worked closely (at times, even in the same room) and were able to coordinate the adjustment of the implementation on either side of the interface to make the other's job easier.

Based on this reasonably involved project, one member has experienced a deeper understanding of teamwork and encapsulation. Without our well defined interface as well as the cooperativeness of every one in the group, the whole process could not have run as smoothly as it did.

Another team member has learned a lot over the course of this project. In addition to learning about how programming languages work and how to write one, he learned about teamwork. This course was the first computer science course for which he had the chance to work on a large-scale programming project with a team. He was able to learn a lot by observing the interaction of the other team members.

10.2 Advice

The primary advice we have to offer is to be prepared and to think through the goals and model of computation for the language rather than trying to pack feature after feature in the language. When in doubt, leave it out. It is also extremely helpful to have determined the grammar early on and iterate through it, refining as needed.

The project leader is a critical component of a successful project. The leader should be able to carefully design the project plan and provide the team with detailed guidelines along the road. Future groups, even though they may know the following advice in theory, should exhibit in practice the motivation to start early, plan ahead, and try to finish before the deadline.

- BadOrdering.spcl
- CyclicReference.spcl
- DuplicateNamedFLO.spcl
- DuplicateNamedFLO2.spcl
- DuplicateNamedSLO.spcl
- DuplicateNamedSLO2.spcl
- EmptyAllow.spcl
- EmptyDeny.spcl
- ForwardReference.spcl
- InternetWebsite_Policy.spcl
- ManyTests.spcl
- MetaAction.spcl
- NestedIfs.spcl
- NoZone.spcl
- PrincipalDeny.spcl
- TestCode.spcl
- TestPrincipal.spcl
- TestingComments.spcl
- TutorialProblem.spcl
- WithObject.spcl

Figure 2: Selected tests from the test suite. (Just to give you a feeling for the types of things we were testing.