INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Reference Installation of the Ponder Policy Toolkit

Patricia Marcu

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Vitalian Danciu Dr. Bernhard Kempter

Abgabetermin: 12. April 2005

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Reference Installation of the Ponder Policy Toolkit

Patricia Marcu

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Vitalian Danciu Dr. Bernhard Kempter

Abgabetermin: 12. April 2005

Hiermit versichere ich, dass ich das vorliegende Fortgeschrittenenpraktikum selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 12. April 2005

. . . .

(Unterschrift des Kandidaten)

Abstract

The topic of Policy based Management is one of the most important today in the context of networks. This work describes my advanced practical which focused on creating a reference installation of the Ponder Toolkit.

My work focuses on installing, configuring and running of Ponder Toolkit on the computer network of the University of Munich CS lab. I wrote also a detailed documentation about Ponder with all the possible problems I encountered through installing, configuring and running the toolkit.

Contents

List of Figures

Chapter 1

Introduction

Why do we need Policies?

Policies are needed in the management of computer -networks and in the management of distributed systems. That is why we need a platform in which we can have overview about all policies that exist, run, which are enabled or disabled. Also this platform must help administrators to write and manage all the policies they need in their system.

The main concept what we focus on is **Policy based Management** as described in the book of Prof. Hegering [HAN 99]. Why is this concept so important? Let's suppose we have large networks, distributed networks, which are administrated with heterogenic tools. So here we have, as a first problem too many different tools! The solution for this is the Policy based Management. Policies are sets of rules which are defined for an existing system. They are declarative, can be run parallel. But there are two more important advantages we have a common "Knoledge Database" and it is used only one language, the "Policy-language".

There have been many attempts to develop such a tool some with more and some with less succes. One of the known Policy Mangement Platform is the **Ponder Policy Based Mangement Toolkit** developed at Imperial College, London.

The topic of my advanced practical was to make a reference installation of the Ponder Toolkit on the computer-network of the Munich Network Management Team. For what is this installation important? There are at least three motives:

- Comparation with other implementations which were made and which are running on our network
- Gaining experience with a known management tool (writing policies to test Ponder)
- It is a mandatory preparatory work for the topics: conflict resolution or the management of mobile networks.

My project is composed of five chapters which reflect my work in the advanced practical. The first chapter makes an introduction to the subject and defines my purpose. The second chapter shortly describes the concepts of policy and policy architecture. The third chapter deals with the policy toolkit Ponder. I describe here the Ponder Policy Specification Language, then the Ponder Toolkit, the third party components which enables the software to run and their interactions.

Chapter four is dedicated to my work so everything I did through my practical I write it here down. That means how I configured LDAP and Elvin and how Ponder is configured. I also made references to the errors which ocurred through my tryings. in the last chapter I summarize what was important in my work, what I did and with what a result.

Chapter 2

What are Policies?

In this chapter we will do an introduction to the topic of policies. We will define them and enumerate some of their properties, and in the last subsection we present a known policy architecture.

2.1 Definition

According to Sloman, **”Policies are rules governing the choices in behaviour of a system”** [POND 02]

An alternative definition of **network policy** is a statement defining which traffic should be treated differently in the network, and how so. It is defined by an administrator and specifies the rules related to handling different types of traffic whithin the network [Verm 01].

Here is a example of a policy: ”The administrator of a network will inactivate the account of a user who enters a wrong password three times in a row.” or ”In our company the staff can receive E-mails only through a special mail gateway” and so on. As we see, policies are very important for the description of the system in which they apply, they actually descibe the behaviour of the system. These rules of the system describe what the system does or may do and what the system doesn’t or may not do.

We see that this approach is elementary, but somehow the policy must ”come” into the system. The administrator of the network is the one who defines them, writes and enforce them. Some general rules must be followed by each administrator at writing policies (it concerns the way how the policy may look like, its functionality etc.). In the following some requirements:

- **Precision.** The policy must be precise. That means it must be specified for a specific network component and it must be understood by this network element. Ambiguity is not acceptable in a policy definition.
- **Consistency.** The policies which are written for a component must be consistent with each other so that they are not in contradiction to other policies written for the same component.
- **Compatibility.** The policy written for a certain network element must be compatible with the capabilities the network element can support.
- **Ease of specification.** The policy must be simple and easy to specify. An administrator must be able to specify the policies with the minimum effort required to specify a consistent and precise set.
- **Intuitiveness.** The policy definition must be done in familiar terms for the administrator.

There are two different levels of policies:

- **High-level policies** - more human oriented, which are understood by humans

- **Low-level policies** - more divices oriented, which are understood by devices (which then enforce them)

For each of these two levels exist regarding specification, different requirements. An administrator uses a **higher-level policy** to express his objectives, which is then translated into a **lower-level policy** that is interpreted by specific devices.

1. **Low-Level Policies**
   This kind of policies must be precise and consistent for each specific device to which they will apply in accordance with the device capabilities. They must also describe exactlly what kind of actions the devices must execute and when this happens. The low-level policy must specify in unambiguous terms the exact operations that the device must perform.

2. **High-Level Policies**
   The requirements include ”ease of specification” and ”intuitiveness” so that the administrator can specify the policies without effort. The definition of the high-level policies depends on the end goal of the administrator. We can see the high-level policies as requirements and the low-level policies as implementations of them.

2.2 Policy Architecture Overview

The policy architecture as defined in the IETF/DMTF consists of four basic elements as we can see it represented in Figure 2.1:

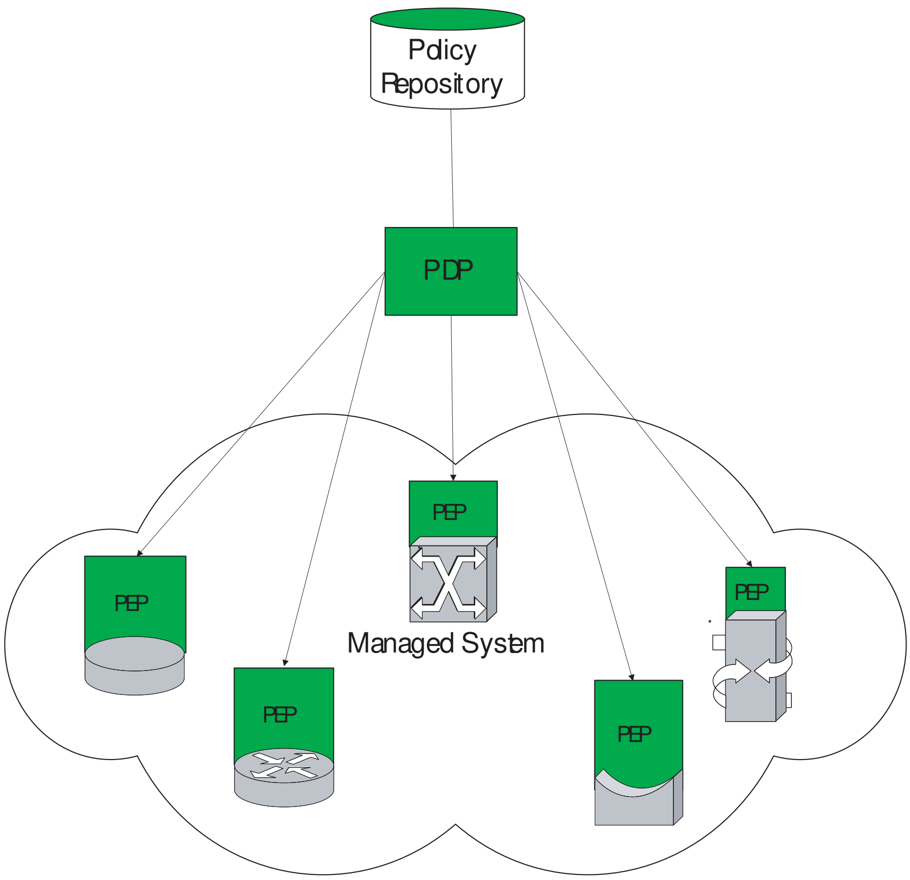

Figure 2.1: The Policy Architecture

- A policy management tool
- A policy repository
- A *Policy Decision Point* (PDP)
- A *Policy Enforcement Point* (PEP)

The *policy management tool* is used by administrators to input the different policies that are active in the network. The policy management tool is the interface between the repository where low-level policies are stored and policy decision point on the one side and the admins (the human part) on the other side. The policy management tool take at each moment the ”pulse” of the system where it controls the whole activities. Thus, the administrator does not have to know how a low level policy must be like because he can enter the high-level policy which will then be translated into low-level policies.

The *policy repository* is used to store the policies generated by the management tool. In the repository we can store both high-level and low-level policies as well as other information important for the system. That’s a very vital point of this architecture that we have a general database where all information about the managed system is stored.

The devices that can then apply and execute the different low-level policies are known as the *Policy Enforcement Points* (PEPs). There are so much PEPs as devices to which policies apply. Each policy has in its definition a reference on the PEP which can enforce it.

The *Policy Decision Point* (PDP) is responsible for interpreting the policies stored in the repository and transmiting the proper policy to the proper PEP where it is executed.

The management tools, the PDPs, the PEPs, and the repository can communicate with each other using a variety of protocols. Which protocol is used to communicate with the repository depends on the type of repository selected. The preferred choice for the repository is a directory that suports the LDAP protocol.

2.3 General Workflows

When something happens in the system the policy decision point is alerted about this. The PDP then decides whether this is an important event or not for the good functioning of the system. It then looks in Repository for the right policy (if one exists), loads it and forwards it to the PEP which is ready to do the action specified in the policy. The PEP executes the action and the system regains balance.

2.4 Summary

In this chapter, I described what policies are and what the policy architecture is with the three important componets policy repository, policy decision point and policy enforcement point. In the next chapter we will see how the teoretical concepts, described until now, apply to the practical example of the Ponder Framework. Ponder is an implementation of the policy architecture, developed at the Imperial College in London. It contains the Ponder Toolkit and the Ponder Policy Specification Language. This will be described in the next chapter.

Chapter 3

Ponder-Framework

In this chapter we will present the Ponder Framework and its components: the Ponder Policy Specification Language and the Ponder Toolkit. Also third party components will be described which are mandatory for the execution of the Toolkit. At the end of the chapter, we will describe how the components interact whith each other.

3.1 The Ponder Policy Specification Language

Ponder is a language for specifying management and security policies. It is a declarative language because all the policy we write are composed actually only of declaration over the system and its components. The policies in Ponder are classified in basic types (authorisations, obligations, refrains and delegations) and composite types (roles, relationships and management structures).

The implemetation of this policy language is realised in Java; policies are stored in databases, and for the control and coordination of the policy objects there are used some control mechanisms for operating systems. Templates were implemented for each type of policy, and due to the inheritance property of the language we can build new composite, complex types of policies. Once the policies are written they can be compiled and so we have either a java-policy-object or a xml-policy-object which can be then stored in the database (this is done by the Ponder-Toolkit). The Ponder Framework was developed with the goal to provide a policy based management tool which runs independently from platform, and which is flexible in writing, administrating, running policies for large networks or distributed systems. To realize this goal is one of the subject of this practical.

In the following sections, we present some of the most important concepts of the Ponder language. The policy examples used are mostly taken from the documentation about the Ponder Policy Language [DDLS 01].

3.1.1 Domains

Domains provide a means of grouping objects to which policies apply and can be used to partition the objects in a large system according to geographical boundaries, object type, responsibility and authority or for the convenience of human managers.[POND 02]

Actually a domain is a set whose elements are references to objects. These object references are the members of the domain. The structuring of the domain is a hierarchical one, which means that there exist **sub-domains** that are members of the parent domain. Domains can overlap, which means that an object in a sub-domain can be an indirect member of more parent domains. The reason why Ponder uses domains for the objects to which policies may apply is that objects can be added and removed from it without change to the policy [DDLS 01]. For the implementation of the domains, the developers used directories organised in LDAP (Lightweight Directory Access Protocol) Service [lda].

3.1.2 Ponder Basic Policies

Basic policies are defined over sets of objects formed by applying set operations, such as union, intersection and differnce to the objects held in the domains. [DLSD 01]

Policy objects and the specified network elements are grouped in domains. For the execution of theses policies, some operation on these domains are defined. Now the question arises as to how we can diffrentiate between ”who” executes the policy and ”to whom” the policies apply. These two concepts are key concepts in the policy definition in Ponder, namely: the **subject** and the **target**. We find these two ”key words” in each definition and in each type of policy. The references of the subject as well as that of the target are stored in domains. Through these two keywords, target and subject, we can control where the policy applies, on which level of the domain, where it may be enforced, and who is qualified to enforce it.

Authorisation and Delegation Policies

Authorisation policies specify what a member of the subject domain may do with members of the target domain. Thus, authorisation policies are designed to protect target objects and are conceptually enforced by the target objects [DLSD 01].

Authorisation policies are classified in: positive (that permit an action) and negative (that forbid an action) authorisation policies. The syntax [DDLS 01] is:

inst	(**auth+** | **auth-**)	policyName”{”
	subject [<type>]	domain-Scope-Expression;
	target [<type>]	domain-Scope-Expresion;
	action	action-list;
	[when	constraint-Expression;] ”}”

So the policy is composed of: **inst** which stands for instance, **auth+ (auth-)** is used to identify that this instance is a positive (negative) authorisation policy, **subject** to designate the subject-domain, **target** to designate the target-domain, **action** to specify which actions are allowed (at positive authorisation policy) or forbidden (at negative authorisation policy), **when** stands for the condition for the enforcingof this policy.

The following are examples of a positive authorisation policy and of a negative authorisation policy:

Example 1: Positive Authorisation Policy [DDLS 01]

inst	**auth+**	routerNMPolicy {
	subject	/NMAdministrator;
	target <ProfileT>	/NM/routers;
	action	load(), remove(), enable(), disable();
}		

This policy specifies that the members of the NMAdministrator domain (the admins) are authorised to load, remove, enable and disable objects of type ProfileT (routers) in the NM/routers domain.

Example 2: Negative Authorisation Policy [DDLS 01]

inst	**auth-**	/negativeAuth/testRouters {
	subject	//testEngineers/trainee;
	target <routerTT>	/routers;
	action	performancetest();
}		

This policy specifies that the trainee engineers are not allowed to perform performance test on routers. This policy will be stored within the /negativeAuth domain [Dami 02].

A delegation policy is similar to an authorisation policy but it is more powerful because it is written to permit the subjects of an authorisation policy (grantors) to delegate some or all of their access rights to new subjects (grantees) [Dami 02]. The effect of a delegation action (by the grantor) is the creation of a new authorisation policy, which is identical with the original policy, except for the new subject (the grantee).

The syntax of the delegation policy is quite similar to that of the authorisation policy [DDLS 01].

inst	(**deleg+** | **deleg-**)	”(”associated-auth-policy”)” policyName”{”
	subject [<type>]	domain-Scope-Expression;
	[**subject** [<type>]	domain-Scope-Expression;]
	[**target** [<type>]	domain-Scope-Expresion;]
	[**action**	action-list;]
	[**when**	constraint-Expression;]
	[**valid**	constraint-Expression;]”}”

A positive delegation policy specifies the authority to delegate and the negative delegation policies forbid delegation.

Obligation and Refrain Policies

Obligation policies are event-triggered condition-action rules, which assign to subjects actions which must be executed on objects in the target domain [POND 02].
The syntax for obligation policies in Ponder is as follows:

inst	**oblig**	policyName”{”
	on	event-pecification;
	subject [<type>]	domain-Scope-Expression;
	[**target** [<type>]	domain-Scope-Expresion;]
	do	obligation-action-list;
	action	action-list;
	[**catch**	exception-specification;]
	[**when**	constraint-Expression;] ”}”

So the policy is composed of: **inst** which stands for instance, **oblig** is used to identify that this instance is an obligation policy, **on** stands for the specification of the event which triggers the policy, **subject** to designate the subject-domain, **target** to designate the target-domain, **action** to specify which actions are allowed, **do** stands for the obligation-action which must be done, **catch** is an optional field to specify some possible exceptions which can occur, **when** stands for the condition for the enforcing of this policy.

Example 3: Obligation Policy

inst	**oblig**	loginFailure {
	on	3*loginfail (userid);
	subject	s=/NM/Admins;
	target [<userT>]	t=/NM/users^{userid};
	do	t.disable()-> s.log(userid);
}		

The event on which this policy ”reacts” is when an user fails his login three times n sequence. Then the administrator of the domain NM/Admins (the subject) must disable this userid (target) of the user in the domain /NM/users. The userid will then be stored in the domain /NM/Admins by the administrator(subject) with the method log(userid). The arrow means that the operations disable() and log() are enforced sequentially [Dami 02].

Refrain policies are sets of rules which define actions which the subjects must not perform on targets. So they are a kind of restraints on the actions that subjects may do. Their syntax is similar to the negative authorisation policies

3.1.3 Ponder Composite Policies

Groups

Groups are used for the organisation and reusability of the policies. This purpose is realised through grouping related policies in packages by differnt criteria. They may reference the same targets, relate to the same department or apply to the same application [DDLS 01].

The syntax for a group instance is shown below:

inst	**group** groupName "{"
	{ basic-policy-definition }
	{ group-definition }
	{ meta-policy-definition }"}"

Roles

A Role has similarities with a group (it is a special case of group), but is more specific in that the policies are grouped by a common subject, for instance a position within an organisation (department manager, project manager). Defining roles contribute to the organisational purposes because when another person ccupies a certain position the policies refering to this position (rights and duties) don't have to be respecified; therefor, it simplifies rganiational proedures [LuSl 97].

A Role definition can help for the description of organisational positions (which can be mapped in Ponder on domains) and it consists of a set of authorisation, obligation, refrain and delegation policies with the subject **subject-domain** of the role. The role syntax [DDLS 01] in Ponder is as follows:

inst	**role** roleName "{"
	{ basic-policy-definition }
	{ group-definition }
	{ meta-policy-definition }"}"

3.1.4 Conclusions

In this section, we described the Ponder Policy Specification Language, what the policies are that it uses, what their syntax are, what domains are. In the next chapter, I'll describe the components of the Ponder Toolkit such as the interfaces that each user or administrator uses at editing, storing, writing and enforcing policies.

3.2 The PONDER Toolkit

Ponder Management Toolkit is an Open Source tool for the Specification and management of Ponder Policies developed at the Imperial College in London. It implements the Policy architecture described in section 2 and is a practical application of the principles of the Policy based Management. From the main console of the management toolkit, one can reach all the tools available, and this allows a user to manage them from a central location (e.g. to start them and to close them). The Ponder Toolkit includes following components: The Domain Browser, the Configuration Manager, the Policy Editor and compiler, and the Management Console Tool.

3.2.1 The Domain-Browser

The domain browser is one of the tools of Ponder which gives the user a graphical mapping of the network (see Figure 3.1). In this tree-like representation, we can see the domains to which the policies apply, we see the domains where the policies are stored, and we can retrieve information about all the LDAP entries.

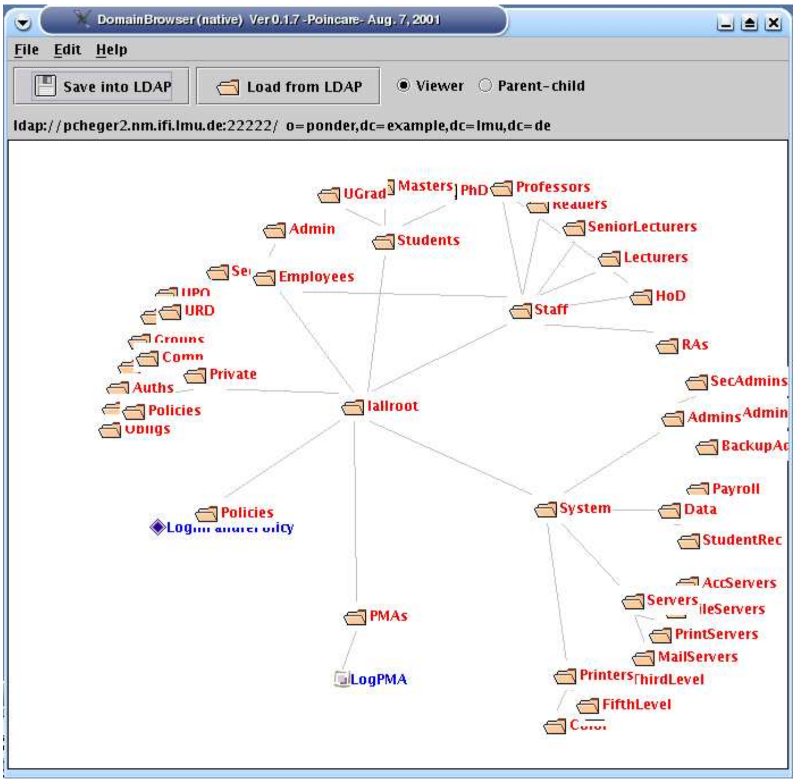

Figure 3.1: Ponder Domain Browser

We have here the whole tree which we store through LDAP (see decription in Section 4.4). Notice that this information stored in LDAP cannot be called when the configuration of Ponder doesn't correspond to that of the LDAP.

3.2.2 The Policy Editor and Compiler Framework

The policy Editor tool (Figure 3.2) compiles the policies written and then stores them into LDAP. Due to the Editor tool in collaboration with the domain browser and the compiler, it is very easy to use the development environment for specifying, reviewing and modifying policies.

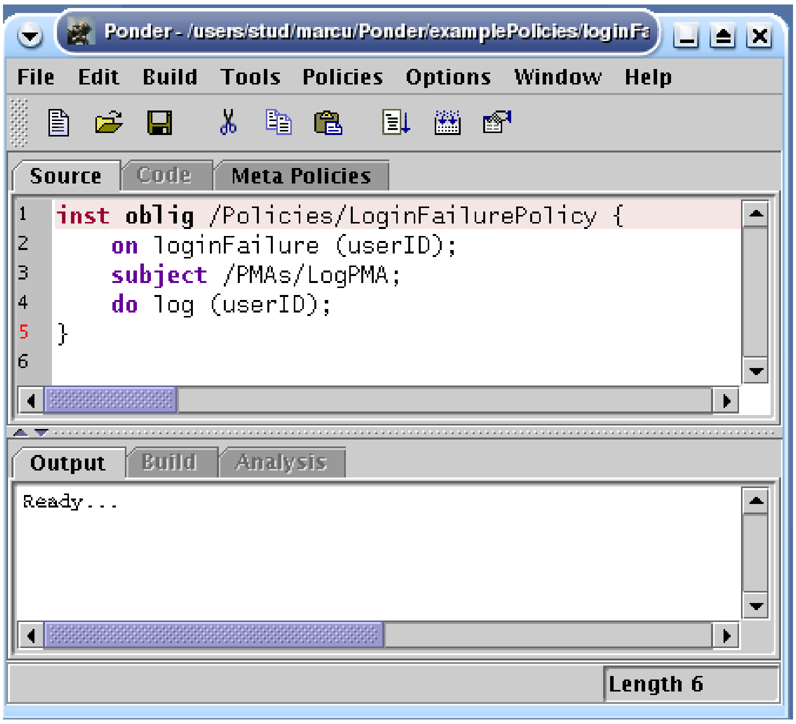

Figure 3.2: Ponder Policy Editor

The Ponder compiler maps policies (high-level) into low-level XML or java code policies. A policy is edited in the Ponder editor and after that compiled and stored in LDAP.

See how the Policy Editor can be used in section 4.4.3

Policies are very easy to write in the policy editor because it has the possibility to call templates for the specific policies. We can select the subject and target domains from the domain browser which is accessible each time from the editor.

3.2.3 The Management Console Tool

The Management Console Tool is a management console for managing policies dynamically. The Management Console Tool has two views: the Policy Objects View and the Enforcement Components View.

All policies stored in LDAP could be called through the management console 3.3 using the domain browser. Once they are loaded in the Policy Objects View, the policies can be ”Load”, ”Enable”, ”Disable”, ”Unload” and stopped, and in this overview we can read the complete informations about the policy, where it is stored, what its name is, what kind of a policy it is, what the subject and target are, what the event is which triggered the policy and so on.

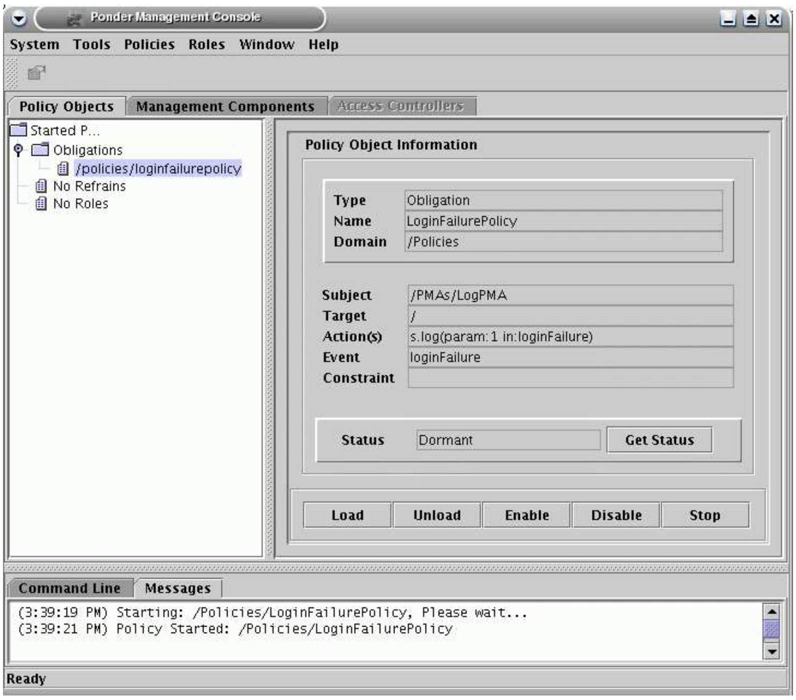

Figure 3.3: Management Console Tool

Details about the selected policy are displayed including the policy-status.

The same function but for Policy Management Components,is fullfilled the Enforcement Components View(see section 4.4.1). Here we can see which policies are written for this special component and what their status is.

I described here the components of the Ponder Toolkit, but this is not enough for running them. There is need for some other components without which the software doesn't run. This third party components will be presented in the following section.

3.3 Third Party Components

The Ponder Toolkit is a very powerful tool, but for running it one must know about additional components Ponder Toolkit uses at runtime. In this section I will give a short overview on the ”other” software which is supposed to be installed and running when we want to use Ponder.

3.3.1 LDAP

LDAP stands for **Lightweight Directory Access Protocol**. It is a lightweight protocol for accessing X.500-based directory services.

What kind of information can be stored in the directory? The basic information model for LDAP is the **entry**. It is a set of attributes that has a **distinguished name** (dn) which is a unique name. Each LDAP-entry has a dn which distinguishes it from other entries. Other attributes are **”cn”** for common name, **”o”** for organisation, **”ou”** for organisation unit and others. So we can actually store everything in LDAP.

How is the information arranged? The entries are hierarchical arranged in a tree structure. We can imagine this tree as having on top the countries, below them are entries representing states and on the next level national organisation and so on [lda]. The tree may also be arranged based on Internet domain names (similar to the DNS).

LDAP allows to control which attributes are required and allowed in an entry through the use of a special attribute **objectClass**. The value of the objectClass attribute can be determined in a LDAP-Schema and so an entry must always be conform to this schema.

From this brief description of LDAP, we can understand why and how it can be used as a policy repository. The implementation of LDAP for Ponder was made not only for the policy objects but also for the policy management components (so for the domain where the policies are active).

After an entry is written, the directory is updated. Other operations which may occur in LDAP are adding, deleting or changing an entry. The most ”sophisticated” operation, however s **searching**. The LDAP search operation allows some portion of the directory to be searched for entries that match some criteria specified by a search filter.

I’ll give in section 4.1 more technical information about how LDAP can be configured for Ponder and how it runs. I used in my experiments OpenLDAP despite the recomandations from Imperial College Ponder Group. The Ponder Schema which is embeded in the Ponder code cannot be used because it is written only for a specific LDAP server and it doesn’t work (for more details see section 4.1), so I transformed it into an Ponder Schema for OpenLDAP in conformity to the [RFC 2256].

3.3.2 RMI

What is RMI? RMI stands for Remote Method Invocation and it is an action of invoking a method of a remote interface on a remote object, and it based on RPC (Remote Procedure Call). In this systems, local substitute object (stub) exist which manages the invocation on a remote object.

RMI applications are normally divided into two separate programs: a server and a client. A server application generates a number of remote objects, makes references to those remote objects and waits for clients to invoke methods on those remote objects. A client application gets a remote reference to one ore more remote objects in the server and then invokes methods on them [rmi].

What is its role in Ponder? Ponder uses in its implementation Java RMI as middleware for the comunication between components. The policy management components are RMI-objects, the policy objects are RMI-objects, and also the domains are RMI-objects. The policy control objects which connect policy objects to the LDAP are also implemented using RMI.

3.3.3 Elvin

What is Elvin? Elvin is the notification service which is used with Ponder. It is actually a distributor of events and notifications. *Producers* detect events (and are responsible for determining that the status change is significant), and send descriptions to the notification service for dissemination to interested *consumers* [Seea 99].

Notification is the concept which defines the sending of an information (message) to an intermediate system to be forwarded to one or more receivers. The result is that the components much better communicate whith each other, there is more flexibility between them and the dependency is lower. Elvin router based on this concept is dynamic, scalable, and used for very many sorts of communication.

What is its role in Ponder?

As briefly outlined above, the notification service n our case Elvin collects and composes events which derive from outside and inside the system, and forwards them to the policy management componenets which then can enforce the specified policy.

3.4 Interactions Between Components

The most important thing is how these separate components can communicate with each other, to establish the framework of the Ponder Management Toolkit. I try to explain in short how this works (see Figure 3.4 which is actually a simplified presentation of Ponder).

When an event happens in the managed system, this event is caught from the Elvin (event service) and forwarded to the policy service which identifies the source of the event and decides which policy s to be initiated. After this decision will be looked up in the LDAP for this policy. This policy has a target and a subject and with help of this, the policy can then be applied to the appropriate component from the right agent.

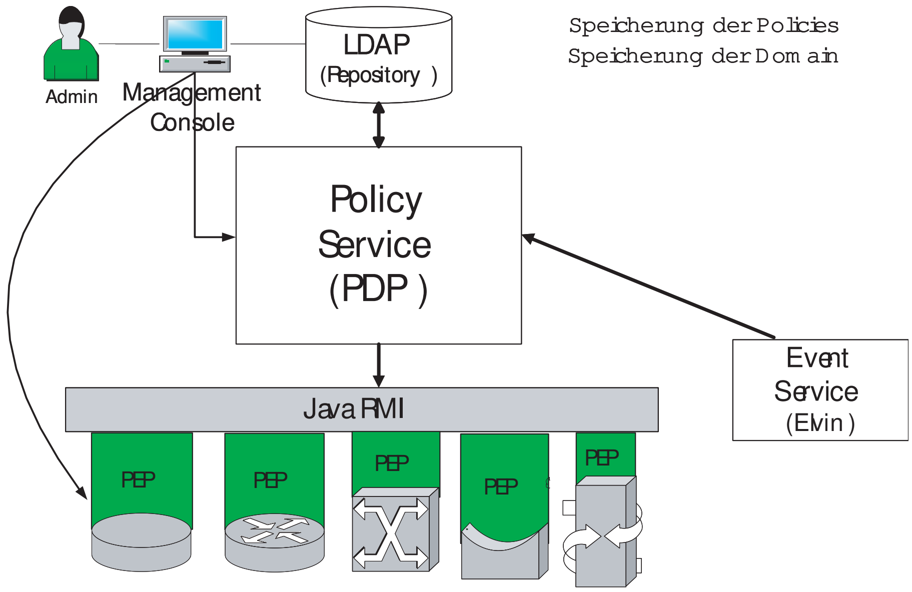

Figure 3.4: Ponder Framework

In addition to the policies, also the network components to which policies may be applied are stored n the LDAP(this is a charakteristics of Ponder). The LDAP is accessed through the domain browser (provided that the connection to the LDAP and the configuration of the Ponder are the same), or it can be called throught the Management console (which implies that the Policy Service is running). We can then check the state of each policy object and we can start it from here, but we can also control the policy management component and its activity.

The Management console is a tool developed at the Imperial College. It is a Java Swing GUI application. It can access all the compnents (network components, policies, domais etc).From here we can start the policy service, we can access the LDAP and check policies and network components.

The Policy Service is the interconnection point of this implementation. It communicates with the network components through the middleware RMI, and with the LDAP. Due to the Policy Service, policies can be stored in LDAP. The Policy Service gets the events transmitted from the event service and distributes them to the appropriate policies, and afterwards connects this policy to the network component which is to enforce it.

3.5 Conclusions

In this chapter, I presented the Ponder Framework with all the components and their interactions. In the following chapter, I will exactly describe what the steps of my work were, what I did and how I installed and configured each component.

Chapter 4

Installation and Configuration

In the last chapters I described Ponder from a theoretical point of view. In this chapter, I will describe what I have done In course of my project, how I applied the knowledge in practical. This will be quite detailed description because I didn't find a detailled Documentation of Ponder and the exact steps of installing and configuring it. I will also describe the installing and running of the OpenLDAP server, of Elvin Event Service, and propose a new Ponder Schema for OpenLDAP.

Prerequisites: To run the Ponder Toolkit sucessfully, it is necesary to have done some downloads: OpenLDAP (it is included in the newer versions of Linux), Ponder Toolkit (open source software from Imperial College London), JVM with rmi, Elvin Router from http://www.mantara.com/products/. These downloads are very important because the absence of one of them causes Ponder to fail.

4.1 LDAP

In the Ponder code, there is a Ponder-schema for IPlanet LDAP server but after I tried to run this server (the code doesn't recognize the LDAP server at all though it is running) and couldn't do this, I decided to take another one, the OpenLDAP. Open LDAP is open source and can be downloaded on `http://www.openldap.org/` , or at the newer SUSE Linux is automatical inserted.

I've installed the software in user space so I can be independent from root access. If you have root access, however, it is surely better to install it as root. The Open LDAP is installed in /users/stud/marcu/lall/openldap/.

The configuration files of LDAP are in lall/openldap (see the configuration file slapd.conf in the Listing 4.1) . Because I will run the slapd only in userspace, I've done some modification to this file: all the files which are included are supposed to be in the root directory so we must give the whole path here.
(e.g
`include /users/stud/marcu/lall/openldap/schema/core.schema` or
`include /users/stud/marcu/lall/openldap/schema/ponderClasses.schema`)

In the pidfile the process ID will be stored:
`pidfile /users/stud/marcu/lall/openldap/slapd.pid`

The label argsfile shows which is the file in which will be stored the arguments with those slapd will be started:
`argsfile /users/stud/marcu/lall/openldap/slapd.args` In the File `slapd.args` I stored the arguments I used
`/usr/lib/openldap/slapd -f /users/stud/marcu/lall/openldap/slapd.conf -d 9 -h ldap://127.0.0.1:22222`

or
`/usr/lib/openldap/slapd -f /users/stud/marcu/lall/openldap/slapd.conf -d 9 -h ldap://pcheger2:22222`

At the end of the file `slapd.conf` exists a database definifion where you can write the type of database you want to use (e.g. bdb, dnssrv, ldap, ldbm, meta, monitor, null, passwd, perl, shell, sql, or tcl)

Other important entries in the file slapd.conf:

`rootdn <dn>`:

Specify the distinguished name of the LDAP root.

`rootpw <password>`:

Specify a password (or hash of the password) for the rootdn. The password can only be set if the rootdn is within the namingContext (suffix) of the database.

`suffix <dn suffix>`:

Specify the DN suffix of queries that will be passed to this backend database.

Listing 4.1: Database configuration in slapd.conf

```

#######################################################################
# ldbm database definitions
#######################################################################

database    ldbm
suffix      "dc=example,dc=lmu,dc=de"
rootdn      "cn=myroot,dc=example,dc=lmu,dc=de"
# Cleartext passwords, especially for the rootdn, should
# be avoid. See slappasswd(8) and slapd.conf(5) for details.
# Use of strong authentication encouraged.
rootpw      root
# The database directory MUST exist prior to running slapd AND
# should only be accessible by the slapd and slap tools.
# Mode 700 recommended.
directory   /users/stud/marcu/lall/openldap/lib
# Indices to maintain
index cn,sn pres,eq,sub
index objectClass eq

#######################################################################
```

The database must exist before slapd will be started. With the entry
`directory /users/stud/marcu/lall/openldap/lib` we assign the place where the database will be stored.

We start the slapd with:
`/usr/lib/openldap/slapd -f /users/stud/marcu/lall/openldap/slapd.conf -d 9 -h ldap://127.0.0.1:22222 start` or
`/usr/lib/openldap/slapd -f /users/stud/marcu/lall/openldap/slapd.conf -d 9 -h ldap://pcheger2.nm.ifi.lmu.de:22222 start`

A new root for the LDAP database is done by:

1. Writing a new .ldif file (or two - one for organistion and one for root)

```
 ********************************************************************
dn:dc=example,dc=lmu,dc=de
objectClass:dcObject
```

```
objectClass:organization
o:lmu
dc:example
*********************************************************************
dn:cn=myroot,dc=example,dc=lmu,dc=de
objectClass:organizationalRole
cn:myroot
*********************************************************************
```

2. This file will be inserted to LDAP with command ldapadd as follows:
   `ldapadd -x -h ldap://pcheger2:22222 -D "cn=myroot,dc=example,dc=lmu,dc=de" -w root -f firstentry.ldif`
   or
   `ldapadd -x -H ldap://pcheger2.nm.ifi.lmu.de:22222/ -D "cn=myroot,dc=example,dc=lmu,dc=de" -W -f secondentry.ldif`

And so we have now a new root for the LDAP database with the name **myroot**.

After doing this, you have to create a group (In my configuration `ponder`which is an entry of the domain `dc=example,dc=lmu,dc=de`). This will be created as follows:

1. Write a .ldif file with the content:

```
 *********************************************************************
dn:o=ponder,dc=example,dc=lmu,dc=de
objectClass:top
objectClass:organization
o:ponder
*********************************************************************
```

   and save it for example as `ponderldif.ldif` in the directory where the `slapd.conf` is saved.(e.g. ∼`/lall/openldap/` in my configuration).

2. Then run the command
   `ldapadd -x -H ldap://pcheger12.nm.ifi.lmu.de:22222/ -D "cn=myroot,dc=example, dc=lmu,dc=de" -w root -f ponderldif.ldif`
   to add the group `ponder` to the LDAP.

We have now the group `ponder` which we use in the next step during the configuration of the Ponder Toolkit.

4.2 Ponder Toolkit

4.2.1 Installation

Download the software from http://www-dse.doc.ic.ac.uk/Research/policies/ponder.shtml. At the time of this writing, the Ponder Toolkit has been taken offline; it is unclear for how long. Install it on your computer. Under linux may be some problems with the installation so you can use a console. The software to download is a file with the name `PonderTkInstall.bin` for Linux (note that for different operating systems, you must download diferent installation files) and I stored it in the directory `~/Downloads`. E.g. if the Software is downloaded in the directory `~/Downloads`, go in this directory and do:
`marcu@pcheger2:~/Downloads> ./PonderTkInstall.bin`
and you can then see:

```
Preparing to install...
Extracting the installation resources from the installer archive...
Configuring the installer for this system's environment...
Launching installer...
```

We assume that Ponder-Toolkit is now installed in the directory `~/Ponder` and contains the following directories and files:

```
marcu@pcheger2:~/Ponder> ls
 code                      exampleDS.sh              ponderEditor.sh      Ponder_Toolkit_InstallLog.log
 compilePolicy.sh          examplePolicies           ponderEnforcement    ponderToolkit.sh
 compilerDebug.txt         lib                       ponder.jar           settings
 configurationManager.sh   LICENSE.txt               ponderSchema         test
 docs                      LoginFailurePolicy.java   ponderToolkit        UninstallerData
```

Now that we have an installed Ponder Toolkit, let's take a look at the configuration.

4.2.2 Ponder Schema

There exists a ponderSchema for LDAP in the directory `~/Ponder/ponderSchema` in the files: attributes.def, classes.def, prev_attributes.def, prev_classes.def; but when we try to run the script `schemadef.bat` first, it only runs under Windows, and second there are so many errors that you don't know what to do first. This file is a skript which should atomatically configure the LDAP whith the Ponder schema. The first thing what you can do in this situation write a new ponderSchema in a new file, and whith a correct syntax according to [RFC 2256] and [RFC 2252]. See new file in Apendix A.

What I did exactly: I took the whole attributes and objectclass definitions for sure from the ponder schema in the 4 other files and I wrote then for each attribute and for each objectclass a new definition in acordance with the specifications on the homepage OpenLDAP [ldap2].
Example for the objectclass **ponderObject**:
in the old schema 4.2

Listing 4.2: Objectclass definition in old ponder schema

```
('PonderObject-oid' NAME 'ponderObject'
   DESC 'Policy Object including Basic Policy and Composite Policy'
   SUP 'managedObject'
   ABSTRACT -- AUXILIARY debugged at 2001/2/23
   MUST ('ponderPolicyType' $ 'policyIsType' $ 'controlObjectRef')
   MAY ('xmlCode') -- added on Feb. 28, 2001 --
)
```

in the new schema 4.3

Listing 4.3: Objectclass definition in new ponder schema

```
objectclass (PonderObject-oid NAME 'ponderObject'
      DESC 'Policy Object including Basic Policy and Composite Policy
              AUXILIARY debugged at 2001/2/23'
      SUP managedObject
      ABSTRACT
      MUST (ponderPolicyType \$ policyIsType \$ controlObjectRef)
      MAY xmlCode)
```

I defined the new ponder schema as described in the OpenLDAP reference, an OID Macro (in the Listing 4.4) so that I can more easily manage the whole oid's I have to use (for each objectclass and for each attribute one). For instance, for the objectclass `ponderObject` which I defined, I have an oid and this is `PonderObject-oid`. This must be defined somewhere so that LDAP can recognize ponderObject as a valid LDAP-object.

As we see in the 8th line of the next listing, the objectidentifier (oid) `PonderObject-oid` is defined as an `managedObject-oid:1` (so it derives from the previous defined `managedObject-oid`). The oid `managedObject-oid` is defined as `ldapobject-oid:1`, and the last one is defined as `lallObjects:1`. The oid `lallObjects` is an `lallElemente:2` oid and this one is an `lall:2 oid`. The oid `lall` is the base oid defined by myself and is the root of my tree; the oid I used is 1.1.1.1.1.223 which is a random number.

If one wants to obtain a registered OID, one must apply for an OID under the Internet Assigned Numbers Authority (IANA). For private experiments, OID's under 1.1 may be used, which is why the `lall` oid begins with 1.1.

Listing 4.4: Ponder Schema as OID Macro

```
   objectIdentifier lall 1.1.1.1.1.223
 2 objectIdentifier lallElemente lall:2
   objectIdentifier lallAttributes lallElemente:1
 4 objectIdentifier lallObjects lallElemente:2
   objectIdentifier ldapobject-oid lallObjects:1
 6 objectIdentifier managedObject-oid ldapobject-oid:1
   objectIdentifier ObjectReference-oid ldapobject-oid:2
 8 objectIdentifier PonderObject-oid managedObject-oid:1
   objectIdentifier BasicPolicyObject-oid PonderObject-oid:1
10 objectIdentifier CompositePolicyObject-oid PonderObject-oid:2
   objectIdentifier DomainObject-oid managedObject-oid:2
12 objectIdentifier EnforcementComponent-oid managedObject-oid:3
   objectIdentifier PolicyMgmtComponent-oid EnforcementComponent-oid:1
14 objectIdentifier AccessController-oid EnforcementComponent-oid:2
   objectIdentifier UserProfileObject-oid managedObject-oid:4
16 objectIdentifier MetaPolicyObject-oid PonderObject-oid:3
```

You have to save this file in the directory `~/lall/openldap/schema/` (relative the directory where slapd.conf is). After you did this, you must go into the file slapd.conf and at the beginning of the file write:
`include /users/stud/marcu/lall/openldap/schema/ponderClasses.schema`
and
`include /users/stud/marcu/lall/openldap/schema/java.schema`
Otherwise ldap wouldn't recognize the ponderSchema.

So now we can start the slapd and from the beginning must accept all the classes and attributes. If the schema is not written as I wrote it here, some interesting errors could occur. Two example of errors are listed in Appendix B. In the first one B.1 there is a syntax error (when the punctuation is incorectly used) and in the second one B.2 is an error which occurs when the newly defined objectclasses aren't recognized from LDAP.

So a very important thing is that slapd runs without errors, for only then can we run and configure Ponder Toolkit. You can try now to run the script `ponderToolkit.sh` (after the installation in the directory `Ponder`) which starts Ponder or if it doesn't work, enter directly the java... command (content in this script) at your terminal. The Ponder Management Toolkit window look like Figure 4.1.

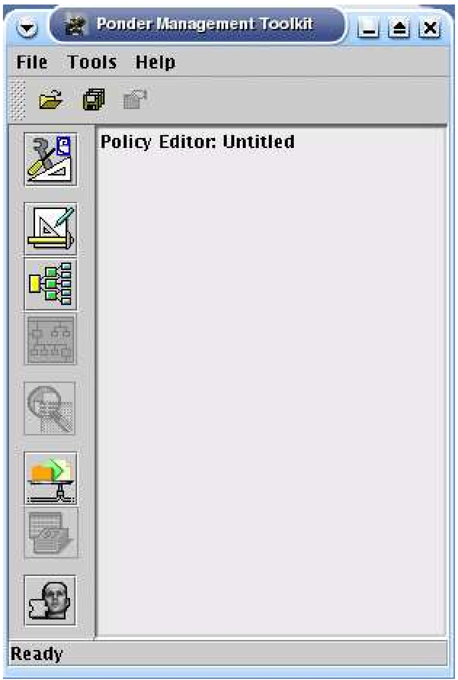

Figure 4.1: Ponder Management Toolkit

Each of the other components of the toolkit can be opened from this window, but each of them can also be opened through their own scrips (e.g. the configuration manager can be opened through running the `configurationManager.sh` skript in the directory `Ponder`). Don't worry if an error occurs because ponder is not yet configured. But after you set and save all configuration options, you can restart ponder. So click the OK button and go on.

4.2.3 Configuration and linking to LDAP

To open the Configuration Manager run the script configurationManager.sh or `java -cp ponder.jar ponderToolkit.configuration.ConfigManager`. Now the Configuration Manager should appear on the screen (see Figure 4.2).

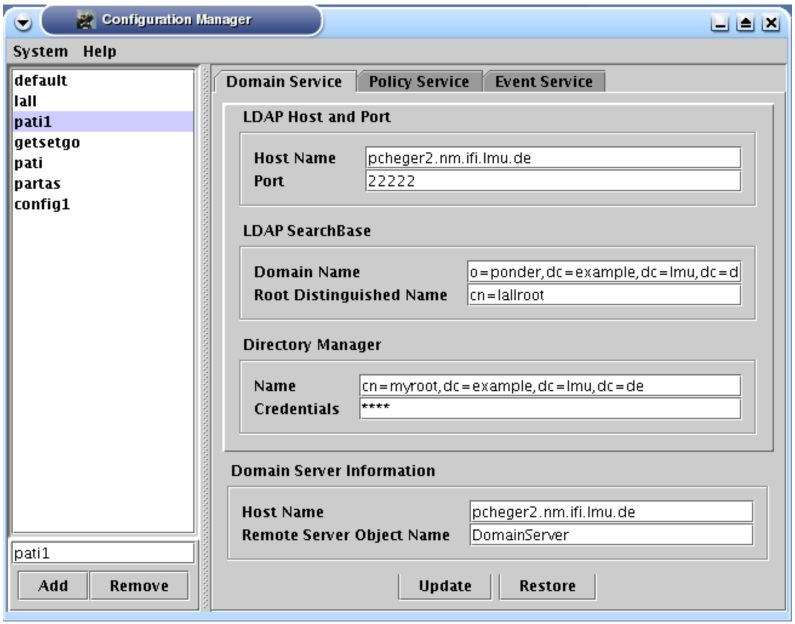

Figure 4.2: Configuration Manager

You must now fill in the Configuration Manager with the informations which match to the settings of the LDAP as follows:

LDAP Host and Port

Host Name: name of the user (here: pcheger2.nm.ifi.lmu.de)

Port: the port number on which you configured the LDAP server (here: 22222)

LDAP SearchBase

Domain Name: the LDAP group which we created in section 4.1 so o=ponder, dc=example, dc=lmu, dc=de

Root Distinguished Name: this name doesn't exist yet. This name domain will be created in LDAP only after runing the next script. (I used here the common name cn=lallroot)

Directory Manager

Name: name of the root in LDAP (cn=myroot,dc=example,dc=lmu,dc=de)

Credentials: credentials as you wrote in the slapd.conf file at the configuration of slapd

Domain Server Information

Host Name: the same hostname as up pcheger2.nm.ifi.lmu.de

Remote Server Object Name: DomainServer(This name is not optional; it must remain unchanged)

Policy Server Information

Host Name: name of the host for policy service (pcheger2.nm.ifi.lmu.de)

Remote Server Object Name: PolicyServer (it is mandatory)

Event Server Information

Host Name: name of the host for event service(pcheger2.nm.ifi.lmu.de)

Remote Server Object Name: EventServer (it is mandatory)

After making these changes, you must go on Update and also on the menu System Save for each one of the three services: domain, policy and event service.

In the directory ~`Ponder/code/ponderToolkit/uttilities/testData` there exists a file CreateUniversityDS.java which stores in LDAP an default Example of an University Department as a tree structure.

The script `exampleDS.sh` must be run so that this treee structure can be stored in `o=ponder,dc=example,dc=lmu,dc=de`. It creates an common name `cn=lallroot`(the Root Distinguished Name of the LDAP SearchBase which we configured in Configuration Manager).

It is very important to do this operation in exactly this sequence:

- run the LDAP-Server with the new PonderSchema (without errors)
- take the changes into the Configuration Manager (as described)
- run the script exampleDS.sh

When you did this, you must have in OpenLDAP the whole tree structure of the University department.

4.3 Event Service

Ponder relies on Elvin for event transport. This is why we must run it on the system. The download can be made from the `http://www.mantara.com/products/`. I used an evaluation version so I get the key sent by email. Elvin is a commercial product but there can be downloaded a trail version for 90 days or for academic use there is free use of the software (for both cases you need a key which will be sent by email).

Elvin Router is an rpm package which you must install on your system and then run it with the command: `/usr/sbin/elvind -l -dd`.

4.4 Policy Enforcement

4.4.1 Running the Policy Service

This step can be done only wenn all the other configurations which I described in the last subsections are done. Actually it s very eas; you only havr to run the script `policyService.bat` in the directory ∼`/Ponder/ponderEnforcement>`. The only problem is that when you have a linux machine, the script cannot be run. I wrote a new script for Linux `policyService.sh` with the content from the former script `policyService.bat` as in the Listing 4.5:

Listing 4.5: Script policyService.sh

```
java -cp .:./classes:./ponder.jar:./lib/httpserver.jar
      ponderEnforcement.RMIStarter 1200 ./classes
sleep 5
java -cp .:./classes:./ponder.jar:./lib/je-4.0b4.jar
      -Djava.rmi.server.codebase=http://localhost:1200/
      -Djava.security.policy=../settings/ponderEnforcement/policyService/policy
      ponderEnforcement.policyService.PolicyServerStarter
```

The problem of another error now occurs (Policy Service error: acces denied). A more detailed listing of the error can be found in Appendix B, the third errorlisting B.3

That means that RMI-registry cannot be started. The error is in the code in ∼ `/Ponder/code/ponderEnforcement/RMIStarter.java` in line 38 where the start command **rmiRegistry** is not correctly used. It must be **rmiregistry**. You must change this and recompile the class ∼ `/Ponder/code/ponderEnforcement/RMIStarter.java`, and be sure to create a new stub too. If you don't want to create a new stub, you can start the registry from console:
`rmiregistry 1099`

If despite the changes, the script `policyService.sh` doesn't run, you can split it into two separate scripts. The first one could be named `startwebserver.sh` (see listing 4.6) and `startPolServ.sh` (see listing 4.7). The first one starts the HTTP-server.

Listing 4.6: Script startwebserver.sh

```
marcu@pcheger2:~/Ponder> echo 'java -cp /users/stud/marcu/Ponder/classes:
/users/stud/marcu/Ponder/ponder.jar:/users/stud/marcu/Ponder/lib/httpserver.jar
ponderEnforcement.RMIStarter 1200 /users/stud/marcu/Ponder/classes' >
startwebserver.sh
marcu@pcheger2:~/Ponder> chmod +x startwebserver.sh

marcu@pcheger2:~/Ponder>./startwebserver.sh
Port occupied. Assuming RMI registry is running on default port: 1099
Nothing is running on port: 1200
Starting HTTP class-server on: 1200
java -cp "lib/httpServer.jar" lib.rmi.http.ClassServer -port 1200 -dir
      "./classes" -verbose
HTTP-server started
```

The second script is actually used to start the Policy Service. It starts the RMI-client and connects it to the registry.

Listing 4.7: Script startPolServ.sh

```
marcu@pcheger2:~/Ponder> echo 'java -cp /users/stud/marcu/Ponder/classes:
/users/stud/marcu/Ponder/ponder.jar:/users/stud/marcu/Ponder/lib/je-4.0b4.jar
-Djava.rmi.server.codebase=http://localhost:1200/
-Djava.security.policy=/users/stud/marcu/Ponder/settings/ponderEnforcement/
policyService/policy ponderEnforcement.policyService.PolicyServerStarter' >
startPolServ.sh
marcu@pcheger2:~/Ponder> chmod +x startPolServ.sh
```

When we start the second script, it could be that a new error occurs (See Apendix B, the 4-th Listing)

The origin of this error is that `/classes` are not found; with classes is meant the code from Ponder. So the solution is to make a link on code:
`marcu@pcheger2: /Ponder> ln -vs code classes`
Then you can run again `startPolServ.sh` and you must get the following output:

Listing 4.8: Output Started Policy Service

```
marcu@pcheger2:~/Ponder> ./startPolServ.sh
>>>http://localhost:1200
>>>1200
The Policy Service is up and running
```

The Policy Service is now up and running and the next step is to create a PMA Agent.

4.4.2 Creating a Policy Management Agent(PMA)

What is a Policy Management Agent?
An agent interprets policies relevant to it. It is an object that implements a policy enforcement interface. This interface can load, enable and disable the enforcement objects created from policy objects. *PMA enforces all the enabled refrain and obligation methods for a subject [DLSD 01].*

The first (important!) step is to create through the domain browser in the root (in my configuration **lallroot**) a directory PMAs.

Use the example that is included in the downloaded software: the script `~/Ponder/ponderEnforcement/LogPMA.bat`. It could be that this script doesn't run so you must write a `LogPMA.sh` script and then copy it in `~/Ponder` and run it there. So a policy management agent is created named LogPMA which is stored in LDAP as child from the newly created directory PMAs. The following output (listing 4.9) will appear on the terminal.

Listing 4.9: Output at runing LogPMA.sh

```
marcu@pcheger2:~/Ponder> ./LogPMA.sh
Initialising Ponder management-Component: please wait
LDAP NAME: cn=LogPMA,cn=PMAs,cn=lallroot
Utils::connectToDomainService at: pcheger2.nm.ifi.lmu.de
Initial DirConext Created Ok
Loading Action: log
Try to Read bytes for: ponderEnforcement/ponderMC/PonderMCimpl.class
#result = 32768 ## = 11754
javaByteCode = [B@1dc423f size = 11754
Started PMC: rmi://pcheger2.in.nm.ifi.lmu.de/LogPMA
The Ponder management-Component is up and running
```

4.4.3 Editing, Compiling and Storing Policies

The first step is to create through the domain browser in the root of LDAP (here in lallroot) a directory with the name Policies. Here the policies will be stored, obviously.

Figure 3.2 shows the policy editor. This Editor is not only for writing policies but also for compilng policies. The policy editor can be opend directly with the comand `./policyEditor.sh` or when you have already started Ponder management Tool (see Figure 4.1), you have to click the second icon from top on the vertical toolbar. There is an example policy in `~/Ponder/examplePolicies/loginFail.pol`

Presuming the editor is on screen, open the file to view it in the editor. It is an obligation policy which tells the LogPMA (agent) to log the user that the event logUser carries as parameter. Now you want to compile it so you must do some configuration. The last Symbol of the topbar is "Choose the settings of the compiler".When you click this, Figure 4.3 appears.

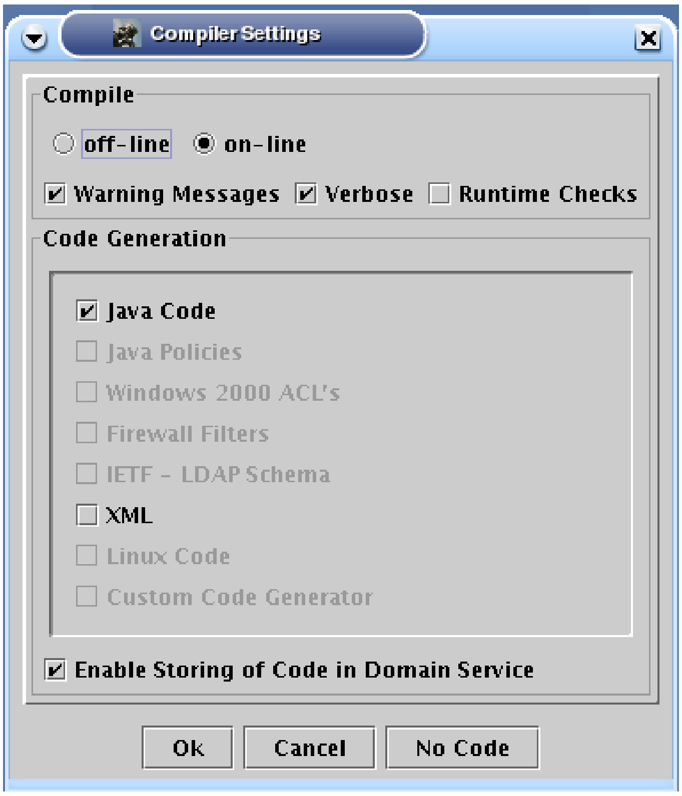

Figure 4.3: Compiler Settings

Select Compile "on line", "Java Code" and "Enable Storing of Code in Domain Service", then "OK". Then you must click the second symbol in the topbar for compiling. When everything is running appropriately, then you must see in the window at the botom "Compiling done".

One requisit of unobstructed compiling is to do all the configurations I described until now, to start slapd, the policy service and the event service (i.e. all components). Now we can load and enable compiled policies.

4.4.4 Loading and Enabling Policies

You can do this through the Management Console Tool (Figure 3.3) described in Section 3.2.3. To load the policy, enter the menu Policies as in Figure 4.4:

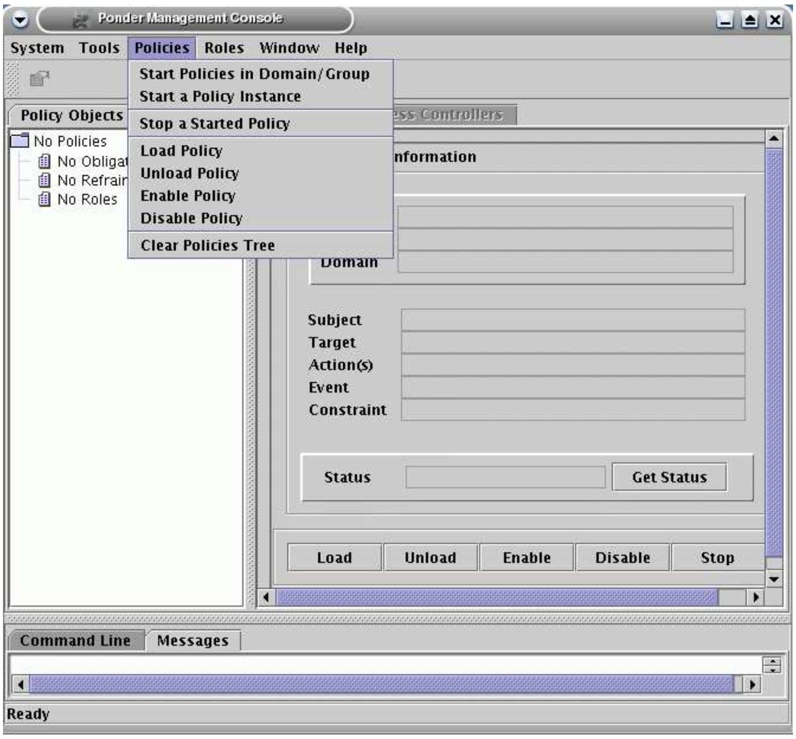

Figure 4.4: Invoking a Policy Instance through the Management Console Tool 1

Then you choose ”Start a policy instance” and the domain browser open as shown in Figure 4.5. Now you must go in the tree to the policy you want to load (here ”LoginFailurePolicy”) and click the right mouse button and then choose ”Close and return”; then the policy in the left part of the management console is loaded.

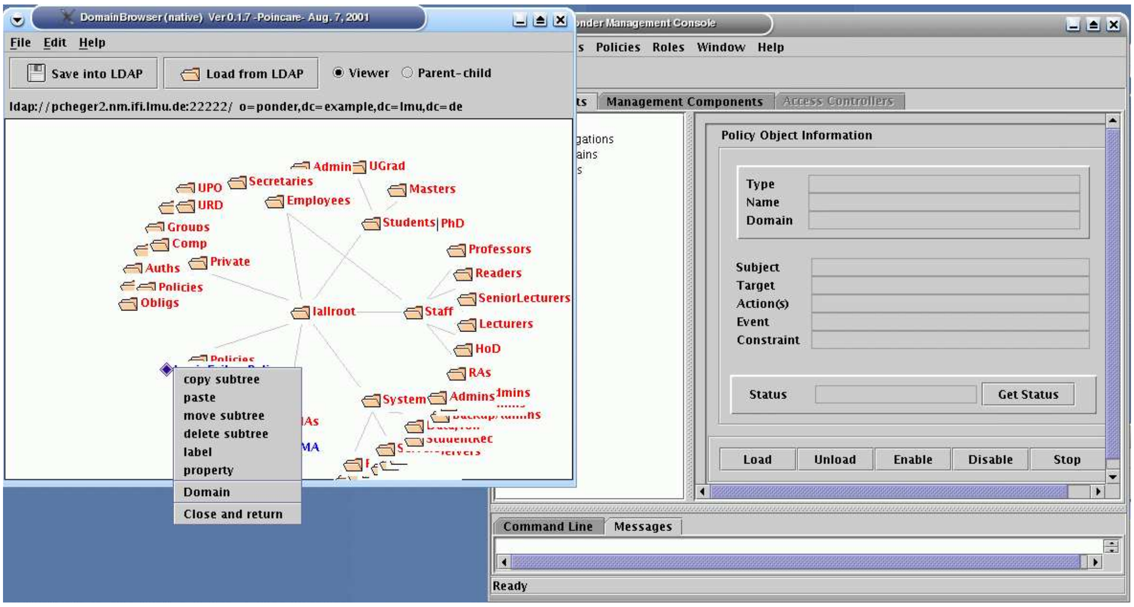

Figure 4.5: Invoking a Policy Instance through the Management Console Tool 2

In the beginning, the status of the policy is ”Dormant”, but there are five buttons on the bottom of the right window where you can ”load”, ”enable”, ”disable” or ”unload” the policy. The order I give here is important because only then does Ponder run without error. **Dormant** is the state of a instantiated policy (it exists but it is inactive). When you click **load** then the policy is loaded into its enforcement agents. Now you can click **enable** and the policy goes into state the enabled where the enforcement agents actively implement the policy. You can attain again the state loaded if you now click **disable** (that means the policy is not implemented any more but can be enabeled again at any time), or you can start the policy when you click **start**. The state dormant can be reache from the state loaded by clicking **unload**; it means that the policy is removed from its enforcements agents. I observed that the policy has to be in dormant state when you close the program; otherwise, you will get at error at a new start.

The cause for this error ies in the fact that the contol object refernce of the policy object is not deleted, except when in state dormant. Consequently, when this reference is not deleted, at a new start you get the error because t tries to build a new reference which already exists. I think that is a bug in code. The error does not occur anymore however, when you accurately bring the polices into the state dormand each time.

You can also load for all existent policies a management component. This is similar to loading policies, but you have to choose the window Management Component.

4.5 Errors in Code

After all these changes and configurations of the toolkit and LDAP, I recognized there where some errors in the Ponder Toolkit code. In the following we discuss those that need to be fixed for the toolkit to run properly.

In the ponder schema I changed the definition of the policyMgmtComponent:

Listing 4.10: New Definition for policyMgmtComponent

```
objectclass (PolicyMgmtComponent-oid NAME 'policyMgmtComponent'
      DESC 'Execution engine for obligation policies'
      SUP enforcementComponent
      AUXILIARY
      MAY appliedPolicies)
```

Because of the error: `[LDAP: error code 65 - object class 'basicPolicyObject' requires attribute 'controlObjectRef']` we must change the definition of basicPolicyObject. That means that in the definition of basicPolicyObject we see `SUP ponderObject` which means we must change ponderObject definition.

Listing 4.11: Old Definition for ponderObject

```
objectclass (PonderObject-oid NAME 'ponderObject'
        DESC 'Policy Object including Basic Policy and Composite Policy
                 AUXILIARY debugged at 2001/2/23'
        SUP managedObject
        ABSTRACT
        MUST (ponderPolicyType $ policyIsType $ controlObjectRef)
        MAY xmlCode)
```

Listing 4.12: New Definition for ponderObject

```
objectclass (PonderObject-oid NAME 'ponderObject'
        DESC 'Policy Object including Basic Policy and Composite Policy
                AUXILIARY debugged at 2001/2/23'
        SUP managedObject
        ABSTRACT
        MUST (ponderPolicyType $ policyIsType)
        MAY (xmlCode $ controlObjectRef))
```

In the File ∼/Ponder/code/ponderEnforcement/policyService/PolServiceImpl.java ControlobjRef=" " appear several times. This is a mistake because according to the definition of the ponder schema, the controlObjectRef is never empty string. I modified this with a default string. I wrote "null" (which has nothing to do with the null = empty string, but is a default,non-empty string). The places where this must be changed are the lines: 174, 246, 308, 388.

In the file ∼/Ponder/code/ponderEnforcement/policyService/LocalPolicyService.java do the same changes as above in the lines: 128, 367, 453, 701. Another mistake in this file is that in line 752 and 1396 it is used "userID" which is not in concordance with the definition in the ponder Schema where "ponderUserID" is used.

In the file ∼/Ponder/code/ponderEnforcement/ponderMC/PMCServer.java there is a typewriteing mistake: instead of "javaReferenceAdress" must be used "javareferenceAdress". This mistake can lead to errors in the policy service at runtime.

Chapter 5

Conclusions

During my practical, I sometimes felt happy to do this topic and sometimes hopeless... At the end it RUNS. I have summerized some of the problems I encountered and how I solved them

In the begining, there was very little documentation and references that I could read, so that it was iposible to start the toolkit quickly. Conseuntly I wrote a detailled documentation about installing and configuring the Toolkit.

What I also noticed was that some important components without which the toolkit does not run are missing. They are not given anywhere at the beginning of the installation, so that one cannt see what the prerequisits for ponder are. It's disturbing not to know this because for each step when you want to run or configure something you must find out what other components are missing. Thus, I documented how we configure LDAP and Elvin properly for the use with the Ponder Toolkit.

Another problem was the Ponder Schema. It is written only for the special iPlanet LDAP-server (now SunOne 5.2) and it doesn't run at all. I analyzed then the schema provided with the toolkit, and I rewrote it so as to be syntax conform (after RFC2256 and RFC2252) for OpenLDAP. I kept the whole names of the attributes and objectclasses (because the Toolkit relies on them).

Most starting and configuration scripts were written for Windows so they couldn't be run on linux machine. I rewrote most of them in shell script.

Some times there were some errors whose origin was difficult to locate and that is why looking for errors in code was quite difficult. I concentrated on fixing errors that prevented Ponder from running.

The products of my practical delivered are:

- a full and complete reference installation with documentation of the Ponder Toolkit
- a reference work for other projects at the MNM department such as: "Handling of Policy Conflicts" and "The management of Mobile Networks"

Appendix A

RFC 2556 compliant schema for Ponder

Listing A.1: New ponder schema

```
#PONDER SCHEMA FOR OPENLDAP
#Written by Patricia Marcu
#12.09.2004
#the Names of the ObjectClasses and Atributes are taken over the 4 old files


objectIdentifier lall 1.1.1.1.1.223
objectIdentifier lallElemente lall:2
objectIdentifier lallAttributes lallElemente:1
objectIdentifier lallObjects lallElemente:2

objectIdentifier ldapobject-oid lallObjects:1
objectIdentifier managedObject-oid ldapobject-oid:1
objectIdentifier ObjectReference-oid ldapobject-oid:2
objectIdentifier PonderObject-oid managedObject-oid:1
objectIdentifier BasicPolicyObject-oid PonderObject-oid:1
objectIdentifier CompositePolicyObject-oid PonderObject-oid:2
objectIdentifier DomainObject-oid managedObject-oid:2
objectIdentifier EnforcementComponent-oid managedObject-oid:3
objectIdentifier PolicyMgmtComponent-oid EnforcementComponent-oid:1
objectIdentifier AccessController-oid EnforcementComponent-oid:2
objectIdentifier UserProfileObject-oid managedObject-oid:4
objectIdentifier MetaPolicyObject-oid PonderObject-oid:3
####################################################################

#objectIdentifier ReferredObjectReference-oid lallAttributes:1
#objectIdentifier ActualObjectReference-oid lallAttributes:2
#objectIdentifier PonderPolicyType-oid lallAttributes:3
#objectIdentifier PolicyIsType-oid lallAttributes:4
#objectIdentifier AccessControllerRef-oid lallAttributes:5
#objectIdentifier ControlObjectRef-oid lallAttributes:6
#objectIdentifier AppliedPolicies-oid lallAttributes:7
#objectIdentifier JavaByteCode-oid lallAttributes:8
#objectIdentifier ObjectType-oid lallAttributes:9
#objectIdentifier XmlCode-oid lallAttributes:10
#objectIdentifier ponderUserId-oid lallAttributes:11
#objectIdentifier policyOwner-oid lallAttributes:12
#objectIdentifier creationTime-oid lallAttributes:13
#objectIdentifier weight-oid lallAttributes:14
```

```
#objectIdentifier certificate-oid lallAttributes:15
#objectIdentifier publickey-oid lallAttributes:16
#objectIdentifier privatekey-oid lallAttributes:17

##########################################################################
#
#    ATTRIBUTEN DEFFINITION
#
##########################################################################

attributetype ( lallAttributes:1 NAME 'referredObjectReference'
      DESC 'reverse pointer to Object reference. SUP is distinguishedName
                        (Syntax is DN).Multiple values are allowed.'
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )

attributetype (lallAttributes:2 NAME 'actualObjectReference'
     DESC 'Pointer to Ponder Object from object reference. SUP is
                       distingushishedName (Syntax is DN)'
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
     SINGLE-VALUE)

attributetype (lallAttributes:3 NAME 'ponderPolicyType'
     DESC 'Type identifier. Syntax is Directory String (Ver 2.63)'
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
     SINGLE-VALUE)


attributetype (lallAttributes:4 NAME 'policyIsType'
     DESC 'Indicates whether the PonderObject is a type or not.Syntax is Boolean.
                       BecauseNetscape Directory Server does not support
                       Boolean,We use Directory String instead.'
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
     SINGLE-VALUE)

attributetype (lallAttributes:5 NAME 'accessControllerRef'
     DESC 'A pointer to Access Controller. SUP is distinguishedName (Its syntax is DN).'
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
     SINGLE-VALUE)

attributetype (lallAttributes:6 NAME 'controlObjectRef'
     DESC 'A pointer to objectControl. Its syntax is Directory String (URL of RMI).'
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
     SINGLE-VALUE)

attributetype (lallAttributes:7 NAME 'appliedPolicies'
     DESC 'A reference to all the policies that apply to the domain -
                       Multiple values allowed. SUP is distinguishedName
                       (Its syntax is DN).'
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.12)

attributetype (lallAttributes:8 NAME 'javaByteCode'
     DESC 'Java byte code programming a Java class. Syntax is Octet String.
                       Because Netscape Directory server does not support
                       Octet String,we use Binary instead.'
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.5
     SINGLE-VALUE)
```

```
attributetype (lallAttributes:9 NAME ’objectType’
     DESC ’The type of a resource managed object.Its syntax is Directory String.
                        There could be multiple’
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.15)

attributetype (lallAttributes:10 NAME ’xmlCode’
     DESC ’The XML code for the Ponder Object. Syntax is Directory String.’
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
     SINGLE-VALUE)

attributetype (lallAttributes:11 NAME ’ponderUserId’
     DESC ’A unique Id for the user. The syntax is Directory String.’
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
     SINGLE-VALUE)

attributetype (lallAttributes:12 NAME ’policyOwner’
     DESC ’The unique Id of the creator of the basic policy’
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
     SINGLE-VALUE)

attributetype (lallAttributes:13 NAME ’creationTime’
     DESC ’An Integer indicating the creation time of the basic policy’
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
     SINGLE-VALUE)

attributetype (lallAttributes:14 NAME ’weight’
     DESC ’An Integer indicating a weight-value the basic policy’
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
     SINGLE-VALUE)

attributetype (lallAttributes:15 NAME ’certificate’
     DESC ’Local temporary’
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.5)

attributetype (lallAttributes:16 NAME ’publickey’
     DESC ’Local temporary’
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.5
     SINGLE-VALUE)


attributetype (lallAttributes:17 NAME ’privatekey’
     DESC ’Local temporary’
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.5
     SINGLE-VALUE)


###############################################################################
#
#    OBJECTCLASSES DEFFINITION
#
###############################################################################

objectclass (ldapobject-oid NAME ’ldapObject’
  DESC ’Top of Ponder objects (Abstract Class)’
  SUP top
  ABSTRACT)

objectclass (managedObject-oid NAME ’managedObject’
```

```
   DESC 'Object handled in Ponder (Abstract Class)'
   SUP ldapObject
   ABSTRACT
   MAY (accessControllerRef $ referredObjectReference $ objectType))

objectclass (ObjectReference-oid NAME 'objectReference'
   DESC 'The psudo node for plural parents'
   SUP ldapObject
   STRUCTURAL
   MUST actualObjectReference
      MAY cn)

objectclass (PonderObject-oid NAME 'ponderObject'
      DESC 'Policy Object including Basic Policy and Composite Policy
      SUP managedObject
      ABSTRACT
      MUST (ponderPolicyType $ policyIsType $ controlObjectRef)
      MAY xmlCode)


objectclass (BasicPolicyObject-oid NAME 'basicPolicyObject'
     DESC 'A Ponder Basic Policy'
     SUP ponderObject
     AUXILIARY
   MAY (policyOwner $ creationTime $ weight))

objectclass (CompositePolicyObject-oid NAME 'compositePolicyObject'
     DESC 'A Ponder Composite Policy'
     SUP ponderObject
     AUXILIARY)

objectclass (DomainObject-oid NAME 'domainObject'
   DESC 'Domain'
   SUP managedObject
   STRUCTURAL
   MUST cn
   MAY appliedPolicies)

objectclass (EnforcementComponent-oid NAME 'enforcementComponent'
    DESC 'Execution engine for Policies (Abstract class)'
    SUP managedObject
    ABSTRACT
    MUST javaByteCode )

objectclass (PolicyMgmtComponent-oid NAME 'policyMgmtComponent'
     DESC 'Execution engine for obligation policies'
     SUP enforcementComponent
     AUXILIARY
      MAY appliedPolicies)


objectclass (AccessController-oid NAME 'accessController'
     DESC 'Execution engine for authorization policies'
     SUP enforcementComponent
     AUXILIARY )

#-- Added on Mar. 5, 2001 --
objectclass (UserProfileObject-oid NAME 'userProfileObject'
```

```
   DESC 'An entry to represent the user in the system'
   SUP managedObject
   AUXILIARY
   MUST ponderUserId)


objectclass (MetaPolicyObject-oid NAME 'metaPolicyObject'
     DESC 'A Ponder Composite Policy'
     SUP ponderObject
     AUXILIARY)
```

Appendix B

Posible Errors by starting LDAP

Listing B.1: Error at the definition of attributes

```
marcu@pcheger2:˜> /usr/lib/openldap/slapd -f /users/stud/marcu/lall/openldap/slapd.conf
 -d 9 -h ldap://pcheger2.nm.ifi.lmu.de:22222
@(#) $OpenLDAP: slapd 2.1.22 (Sep 23 2003 21:38:34) $
        root@E180:/usr/src/packages/BUILD/openldap-2.1.22/servers/slapd
daemon_init: listen on ldap://pcheger2.nm.ifi.lmu.de:22222
daemon_init: 1 listeners to open...
ldap_url_parse_ext(ldap://pcheger2.nm.ifi.lmu.de:22222)
daemon: initialized ldap://pcheger2.nm.ifi.lmu.de:22222
daemon_init: 1 listeners opened
slapd init: initiated server.
slap_sasl_init: initialized!
bdb_initialize: initialize BDB backend
bdb_initialize: Sleepycat Software: Berkeley DB 4.1.25: (September 23, 2003)
>>> dnNormalize: <cn=Subschema>
=> ldap_bv2dn(cn=Subschema,0)
<= ldap_bv2dn(cn=Subschema,0)=0
=> ldap_dn2bv(272)
<= ldap_dn2bv(cn=subschema,272)=0
<<< dnNormalize: <cn=subschema>
/users/stud/marcu/lall/openldap/schema/ponderClasses.def: line 57: Missing closing parenthesis
AttributeTypeDescription = "(" whsp
  numericoid whsp ; AttributeType identifier
  [ "NAME" qdescrs ]      ; name used in AttributeType
  [ "DESC" qdstring ]     ; description
  [ "OBSOLETE" whsp ]
  [ "SUP" woid ]          ; derived from this other
                            ; AttributeType
  [ "EQUALITY" woid ]     ; Matching Rule name
  [ "ORDERING" woid ]     ; Matching Rule name
  [ "SUBSTR" woid ]       ; Matching Rule name
  [ "SYNTAX" whsp noidlen whsp ] ; see section 4.3
  [ "SINGLE-VALUE" whsp ] ; default multi-valued
  [ "COLLECTIVE" whsp ]   ; default not collective
  [ "NO-USER-MODIFICATION" whsp ]; default user modifiable
  [ "USAGE" whsp AttributeUsage ]; default userApplications ; userApplications;
            directoryOperation; distributedOperation ; dSAOperation whsp ")"
slapd shutdown: freeing system resources.
slapd stopped.
connections\_destroy: nothing to destroy.
```

Listing B.2: Error when the ldap doesn't recognize the Objectclasses

```
marcu@pcheger2:~/lall/openldap/schema> /usr/lib/openldap/slapd -f
      /users/stud/marcu/lall/openldap/slapd.conf -d 9 -h ldap://pcheger2.nm.ifi.lmu.de:22222
@(#) $OpenLDAP: slapd 2.1.22 (Sep 23 2003 21:38:34) $
      root@E180:/usr/src/packages/BUILD/openldap-2.1.22/servers/slapd
daemon_init: listen on ldap://pcheger2.nm.ifi.lmu.de:22222
daemon_init: 1 listeners to open...
ldap_url_parse_ext(ldap://pcheger2.nm.ifi.lmu.de:22222)
daemon: initialized ldap://pcheger2.nm.ifi.lmu.de:22222
daemon_init: 1 listeners opened
slapd init: initiated server.
slap_sasl_init: initialized!
bdb_initialize: initialize BDB backend
bdb_initialize: Sleepycat Software: Berkeley DB 4.1.25: (September 23, 2003)
>>> dnNormalize: <cn=Subschema>
=> ldap_bv2dn(cn=Subschema,0)
<= ldap_bv2dn(cn=Subschema,0)=0
=> ldap_dn2bv(272)
<= ldap_dn2bv(cn=subschema,272)=0
<<< dnNormalize: <cn=subschema>
/users/stud/marcu/lall/openldap/schema/ponderClasses.def: line 13: unknown directive
      "objectClass('ldapobject-oid'" outside backend info and database
            definitions (ignored)
/users/stud/marcu/lall/openldap/schema/ponderClasses.def: line 21: unknown directive
      "objectClass('managedObject-oid'" outside backend info and database
            definitions (ignored)
/users/stud/marcu/lall/openldap/schema/ponderClasses.def: line 27: unknown directive
      "objectClass('ObjectReference-oid'" outside backend info and database
            definitions (ignored)
/users/stud/marcu/lall/openldap/schema/ponderClasses.def: line 34: unknown directive
      "objectClass('PonderObject-oid'" outside backend info and database
            definitions (ignored)
/users/stud/marcu/lall/openldap/schema/ponderClasses.def: line 35: unknown directive
      ")" outside backend info and database definitions (ignored)
/users/stud/marcu/lall/openldap/schema/ponderClasses.def: line 41: unknown directive
      "objectClass('BasicPolicyObject-oid'" outside backend info and database
            definitions (ignored)
/users/stud/marcu/lall/openldap/schema/ponderClasses.def: line 46: unknown directive
      "objectClass('CompositePolicyObject-oid'" outside backend info and database
            definitions (ignored)
/users/stud/marcu/lall/openldap/schema/ponderClasses.def: line 53: unknown directive
      "objectClass('DomainObject-oid'" outside backend info and database
            definitions (ignored)
/users/stud/marcu/lall/openldap/schema/ponderClasses.def: line 59: unknown directive
      "objectClass('EnforcementComponent-oid'" outside backend info and database
            definitions (ignored)
/users/stud/marcu/lall/openldap/schema/ponderClasses.def: line 64: unknown directive
      "objectClass('PolicyMgmtComponent-oid'" outside backend info and database
            definitions (ignored)
/users/stud/marcu/lall/openldap/schema/ponderClasses.def: line 69: unknown directive
      "objectClass('AccessController-oid'" outside backend info and database
            definitions (ignored)
/users/stud/marcu/lall/openldap/schema/ponderClasses.def: line 71: unknown directive
      "--" outside backend info and database definitions (ignored)
/users/stud/marcu/lall/openldap/schema/ponderClasses.def: line 76: unknown directive
      "objectClass('UserProfileObject-oid'" outside backend info and database
            definitions (ignored)
```

Listing B.3: Error at starting of the Policy Service

```
pcheger2:~/Ponder> java -cp .:./classes:./ponder.jar:./lib/httpserver.jar
      ponderEnforcement.RMIStarter 1200 ./classes
RMI registry is not running on default port: 1099
Starting RMIRegistry 1099...
Could not start RMI registry: java.io.IOException: rmiRegistry: not found



marcu@pcheger2:~/Ponder> java -cp .:./classes:./ponder.jar:./lib/je-4.0b4.jar
      -Djava.rmi.server.codebase=http://localhost:1200/
      -Djava.security.policy=../settings/ponderEnforcement/policyService/policy
      ponderEnforcement.policyService.PolicyServerStarter
>>>http://localhost:1200/
>>>1200
Initialising Policy-Service: please wait
Configuration Manager: Using build-in default properties.
Warning: properties could not be saved
java.security.AccessControlException: access denied (java.io.FilePermission
      settings/configuration.ini write)
Configuration Manager: Using build-in default properties.
Policy Service error: access denied (java.net.SocketPermission 127.0.0.1:1099
            connect,resolve)
java.security.AccessControlException: access denied (java.net.SocketPermission
            127.0.0.1:1099 connect,resolve)
      at java.security.AccessControlContext.checkPermission
            (AccessControlContext.java:269)
      at java.security.AccessController.checkPermission
            (AccessController.java:401)
      at java.lang.SecurityManager.checkPermission(SecurityManager.java:524)
      at java.lang.SecurityManager.checkConnect(SecurityManager.java:1026)
      at java.net.Socket.connect(Socket.java:446)
      at java.net.Socket.connect(Socket.java:402)
      at java.net.Socket.<init>(Socket.java:309)
      at java.net.Socket.<init>(Socket.java:124)
      at sun.rmi.transport.proxy.RMIDirectSocketFactory.createSocket
            (RMIDirectSocketFactory.java:22)
      at sun.rmi.transport.proxy.RMIMasterSocketFactory.createSocket
            (RMIMasterSocketFactory.java:128)
      at sun.rmi.transport.tcp.TCPEndpoint.newSocket(TCPEndpoint.java:562)
      at sun.rmi.transport.tcp.TCPChannel.createConnection
            (TCPChannel.java:185)
 at sun.rmi.transport.tcp.TCPChannel.newConnection(TCPChannel.java:171)
      at sun.rmi.server.UnicastRef.newCall(UnicastRef.java:313)
      at sun.rmi.registry.RegistryImpl_Stub.rebind(Unknown Source)
      at java.rmi.Naming.rebind(Naming.java:160)
      at ponderEnforcement.policyService.PolicyServerStarter.<init>
            (PolicyServerStarter.java:45)
      at ponderEnforcement.policyService.PolicyServerStarter.main
            (PolicyServerStarter.java:90)
```

Listing B.4: Error at starting of startPolServ.sh

```
pcheger2:~/Ponder> ./startPolSer.sh
Tue Aug 24 13:46:23 CEST 2004: ponderEnforcement/policyService/PolicyService
      Impl_Stub.class re
>>>http://localhost:1200/
>>>1200
Initialising Policy-Service: please wait
Policy Service error: undeclared checked exception; nested exception is:
      java.lang.ClassNotFoundException: ponderEnforcement.policyService.
            PolicyServiceImpl_Stub not found in [http://localhost:1200/]
java.rmi.UnexpectedException: undeclared checked exception; nested exception is:
      java.lang.ClassNotFoundException: ponderEnforcement.policyService.
            PolicyServiceImpl_Stub not found in [http://localhost:1200/]
      at sun.rmi.registry.RegistryImpl_Stub.rebind(Unknown Source)
      at java.rmi.Naming.rebind(Naming.java:160)
      at ponderEnforcement.policyService.PolicyServerStarter.
            <init>(PolicyServerStarter.java:45)
```

Bibliography

[Dami 02] DAMIANOU, C. NICODEMOS: *A Policy Framework for Management of Distributed Systems*. PhD thesis, Imperial College of Science, Technology and Medicine, University of London, Department of Computing, 2002.

[DDLS 01] DAMIANOU, N., N. DULAY, E. LUPU and M. SLOMAN: *The Ponder Policy Specification Language*. In *Workshop on Policies for Distributed Systems and Networks*, January 2001.

[DLSD 01] DULAY, N., E. LUPU, M. SLOMAN and N. DAMIANOU: *A Policy Deployment Model for the Ponder Language*. In *IEEE/IFIP International Symposium on Integrated Network Management (IM'2001)*. IEEE Press, 2001.

[HAN 99] HEGERING, H.-G., S. ABECK and B. NEUMAIR: *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application*. Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999. 651 p.

[lda] *Introduction to OpenLDAP Directory Services*, http://www.openldap.org/doc/admin22/intro.html .

[ldap2] *Introduction to OpenLDAP Directory Services*, http://www.openldap.org/doc/admin22/schema.html .

[LuSl 97] LUPU, E. and M. SLOMAN: *Towards a Role Based Framework for Distributed Systems Management*. In *Journal of Network and Systems Management*, 1997.

[POND 02] *The PONDER Policy Based Management Toolkit*, 2002.

[RFC 2252] WAHL, M., A. COULBECK, T. HOWES and S. KILLE: *RFC 2252: Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions*. Technical Report, December 1997, ftp://ftp.isi.edu/in-notes/rfc2252.txt .

[RFC 2256] WAHL, M.: *RFC 2256: A Summary of the X.500(96) User Schema for use with LDAPv3*. Technical Report, December 1997, ftp://ftp.isi.edu/in-notes/rfc2256.txt .

[rmi] *JavaRMI Specifications*, http://java.sun.com/products/jdk/rmi/ .

[Seea 99] SEGALL, BILL and ET AL.: *Augumenting the Workaday World with Elvin*. In *ESCW'99*. Kluwer Academic Publishers, Sept 1999.

[Verm 01] VERMA, DINESH C.: *Policy-Based Networking: Architecture and Algorithms*. New Riders Publishing, 2001.