1. Introduction

During the analysis phase of our project, we investigated the possible problems about web, its usage, and its control within restricted zones. We tried to identify and define those problems, and come out with creative solutions to those that are within our project's scope. We provided comprehensive information about those problems and our proposed solutions in our analysis report. After our delivery of the analysis report, we continued working on the problem set and our solutions. We decided on our project's technical aspects and infrastructure.

1.1 Problem Definition

In recent years, Internet have seen a great rise in its popularity and usage areas. People of all occupations and ages are on the internet, industries depend on it heavily, ant there are completeley new businesses that exist primarily in the internet. Companies depend heavily on internet usage, as a result most of the employees have access to the internet. Other than big companies, many other organizations are open to internet. Some examples are governmental bodies, schools and universities, non-profit organizations, hospitals, small offices. Examples are not limited to these, and the most common internet user is the home user.

The giant structure and variety of the internet makes it a valuable information source and communication medium. But any useful thing has some drawbacks, and internet is no exception. In fact ease of access and abundancy of uncontrolled traffic makes the internet a difficult place to stay safe. And a large organization is not only concerned with outside threats, but also inside threats which stem from workers or members. The workers or members of the organizations may visit sites or download files that may bring harm to the whole network of the organization. Some sensitive information may leak from the network to the outside world, either by a careless worker or one with a malicious intent.

Companies and organizations mostly used traditional firewalls, which does packet filtering by looking at easily extractable data, like IP adress or port of the communicating parties. This would be useful if the intent was to block and control some servers/protocols only. But the internet traffic around the world consists of web traffic mostly, distributed

to a large number of servers in many different locations throughout the world. Traditional firewalls are not able to tackle with difficulties presented by such a gigantic network, especially in a corporate environment defined above. It is the content of the traffic, not the source/destination that should be checked to provide comprehensive control over the information flow. In a present day environment, where nearly the whole internet traffic is web traffic, blocking ports have little influence on the information security.

Our project aims to solve the most important problems of the modern day corporate and onrganizational network, by providing an application layer gateway; which will enable the network manager to control web traffic of the network in many different ways. Name of our project is IronCurtain and it will not only solve problems of big corporations and organizations, but also home users and small networks. IronCurtain will provide flexible control over the whole web traffic, user and group based control, extensive logging capabilities, and a configuration interface that is accesible from web. We plan to design and build IronCurtain so flexible, strong and easy to use, such that it will answer most of the modern day internet-based concerns.

1.2 Project Scope and Goals

Our project, named IronCurtain, will implement an application level gateway for web filtering and access control. Our project's goals are as follows:

- Complete HTTP/1.1 support
- HTTPS support
- Decomposition of Entire Communication
- Plugin architecture for rules and actions
- Complete configuration over Web
- Secure User Authentication
- Logging and Accounting
- Multi-threaded (thread-pool) implementation
- Alerts (via plugins)
- Content identification (via plugins)

1.3 Usage Areas

IronCurtain will be flexible enough to satisfy needs of a great variety of entities. From large enterprises to home users, IronCurtain will have an answer to all of these entities' problems, thanks to the flexible and extendable plug-in mechanism.

- **Companies:** Modern day companies use internet access extensively to maintain theri operations and profitability. It is essential for companies to access internet. If the access is uncontrolled, many adverse effects of the internet give harm to the company overall. Some of these are: Productivity loss associated to worker distraction caused by uncontrolled web site access, maliciously designed web sites, leakage of sensitive company information, etc. By using IronCurtain, companies can enable appropriate filters to take precautions about these problems. Also IronCurtain would be a security layer between company servers an the internet, checking for possible security violations and attacks and preventing according to the rules. This would require no modification of the system, only writingthe appropriate rules will be sufficent. Also sensitive information about the company network may be blocked using IronCurtain. Ip adress of the computers can be replaced by the adress of the proxy.
- **Universities:** A university environment is usually more relaxed than a corporate one. But still administration of a university would need firm control on the network. Students and academics in the university might use up all the bandwith of the university. Or they might cause legal trouble by downloading illegal files. It would be possible to control bandwidth of the users and their internet activities using IronCurtain.
- **Other Areas:** Flexible plugin architecture of IronCurtain will enable its usage in a variety of networks, even a single computer network. Appropriate rules and plugins for the area of application will be chosen and IronCurtain will protect the network from inside or outside malicious traffic, and it will enforce other principles determined by the network's owners.

1.4 Design Constraints

Our main design constraints are as follows.

Time

Our fixed schedule is determined by our course syllabus. We have approximately six months remaining to finish the project completely. The design should be finished in one month. During the design we will also work on the prototype and it will be finished in one and a half months from the delivery of initial design report.

Language constraints

We decided to use the Python language as implementation language. It allows easy usage of plugin. In fact, python is used as a scripting language inside many applications. Most of the time we will be using Python's integrated libraries. But we may use some other language like PHP for the user interface, or we may use also Python as the interface language.

Performance and Network Latency

IronCurtain application level gateway aims to introduce lowest levels of latency to the network communication speed. The software will use threads to be more performant than a forking implementation. Also the software will reuse existing connections and will not try to open a new connection which takes CPU time and introduces latency. Gzip compression and chunked encoding feature of HTTP 1.1 standard will be used to reduce latency.

Maintenance

The IronCurtain will require minimal maintenance. After the administrator defines the plug-in, rules, general settings, per user and per group settings; there will be no need to check the operation of the gateway, other than the cases of; adding of a new rule and querying for statistics.

User Interface

IronCurtain is not a user-interface intensive application. Most of its operation happens behind the communication of the members of the network. The user interface is opened when settings are to be changed, or statistics are to be displayed. Other than these conditions, no user interface is required for normal operation. The user interface of the settings and statistics parts of the software will be easy and intuitive to use. The statistics should be displayed in a variety of easy to understand approaches.

1.5 Design Objectives

- **Portability**

  Our usage of Python language makes IronCurtain easily portable to any operating system that has a python run-time components; Linux, BSD, Mac OS X, Microsoft Windows, AIX, Amiga, AROS, AS/400, BeOS, OS/2, OS/390, Irix, Palm OS, Plan 9, PlayStation 2, Psion, QNX, RISC OS, Sharp Zaurus, Solaris, Symbian OS, VMS, VxWorks, Windows CE/Pocket PC, Xbox, z/OS. And also all of the hardware platforms that support above operating systems and the python run-time components.

  Because of this portability, users with any specified OS should be able to use IronCurtain, and if we decide to use another library, we will try to keep this platform independency.

- **Extendibility**

  The plug-in architecture of IronCurtain will enable very extendible and flexible operation. If there is a need for a new limitation/control/statistics options. There may be possible affordable solutions using IronCurtain. The easiest one being usage of existing plug-in and writing a new rule satisfying the needed options. Or a new plug-in can be written using the python programming language. Using plug-in any kind of behavior can be added to IronCurtain.

- **Maintainability**

  Maintainability is an important objective for IronCurtain. The plug-in system is very modularly designed, plug-in are independent of each other. Also functionality of IronCurtain is upon plug-in. Because of this modular design, we could change the internals of any plug-in without causing problems for the other untouched plug-ins and general operation of IronCurtain. This makes IronCurtain easy to maintain, because changes to one plug-in/functionality do not require the rewriting of other plug-in.

2. Data Design

The data of users, groups, their logging levels and rules will be stored in the database. The logs of users and rule actions will be written to database as well. In order to store the data in a structured form, the data objects will be used. In this section, we will look at the data objects, their relationships, the ER-diagram and the data dictionary to describe the data.

2.1 Data Objects

User

The User entity will store data associated with the users of the system. When they register with the system the data they enter will be stored as an instance of the User object and they can change the information at any time. The attributes of the entity will be:

- User_ID
- Real_Name
- Passwd
- Email
- Group_ID
- LogLevel_ID

The *User_ID* will be a string that will be used as the primary key as it will be unique to each user in the network. The *User_ID* and password will be used to log on during authentication. *Passwd* is the MD5 hash of the user password. *Group_ID* will be the reference to *Group* entity, which the user is associated with. The user does not have to be assigned to a Group. *LogLevel_ID* is the reference to *LogLevel* entity and it will define what to log for the user. *Email* will be the user's e-mail address. The user will be notified by e-mail if necessary.

Admin

Admin entity is just a User entity, but it will be used to identify the administrators. Admin entity will keep the *User_ID*'s of the users who has administrative rights. The users in this entity will be able to log in to administrative page of the system with their passwords.

Group

Group entity is the generalization of User entity. It will be used to group users to ease applying same rules to many users. The attributes of the entity will be:

- Group_ID
- LogLevel_ID
- Description

The *Group_ID* will be a string that will be used as the primary key as it will be unique to each *Group*. *LogLevel_ID* will define which information will be logged for the members of the users. *Description* will be just an info string.

LogLevel

The LogLevel entity will store the logging rules for users and groups. The attributes of the entity will be:

- LogLevel_ID
- Site
- Domain

- Bandwidth

LogLevel_ID is an integer and the primary key. The other attributes are of type boolean. These will be used to set if the related information will be logged or not. For *Site* attribute, it will be set to true if we want to set up a log level that logs the site names. *Domain* will be used to log the domain information of the visited site. *Bandwidth* will be used to enable or disable logging of data transfer size.

UserLog

UserLog entity will store the logs of the user actions according to users' logging levels.

- User_ID
- Site
- Domain
- Bandwidth
- Open_Time
- Close_Time

User_ID and the *Open_Time* together will be the primary key. *User_ID* is the foreign key and reference to *User* entity. *Site* is the string to hold the site URL that is visited. *Domain* is a string and it is the domain name of the visited site. *Bandwidth* is an integer that is the size of the data transfer. *Open_Time* and *Close_Time* are the action times. *Site, Domain, Bandwidth* will be stored according to user's logging level, *LogLevel_ID* attribute of the *User* or *Group* entity.

Plugin

The Plugin entity will store the identifiers of the plug-ins in the system. The plug-in schemas will be saved as xml files so we will just keep the ids of the plug-ins to relate them with the rules. The attributes of this entity will be:

- Plugin_ID
- Plug_File
- Schema_File

- Plug_Hash
- Schema_Hash
- Description

The *Plugin_ID* will be a string that will be used as the primary key as it will be unique to each Plug-in. This key will be read from the xml file. The *Plug_File* and *Schema_File* attributes will keep the paths of the Plug-in file and schema file of the Plug-in, respectively. The MD5 hashes of the files will be stored in *Plug_Hash* and *Schema_Hash*, accordingly. If any change occurs in these files, it will be checked from hashes. *Description* will be just an info string.

Rule

The Rule entity will store the identifiers of the rules that are generated from Plugins. Every rule derives from a Plug-in template. The attributes of the entity will be:

- Rule_ID
- Plugin_ID
- File
- File_Hash
- Description

Rule_ID is a string and primary key. *Rule_ID* is the identifier of the rule that is generated from a plugin schema and written to an xml file. The File attribute will keep the path of the xml file. *File_Hash* will be the MD5 hash of the file and make it possible to check whether any change occurs in the file. *Plugin_ID* is the foreign key and reference to *Plugin* entity. Description is just an info string.

RuleLog

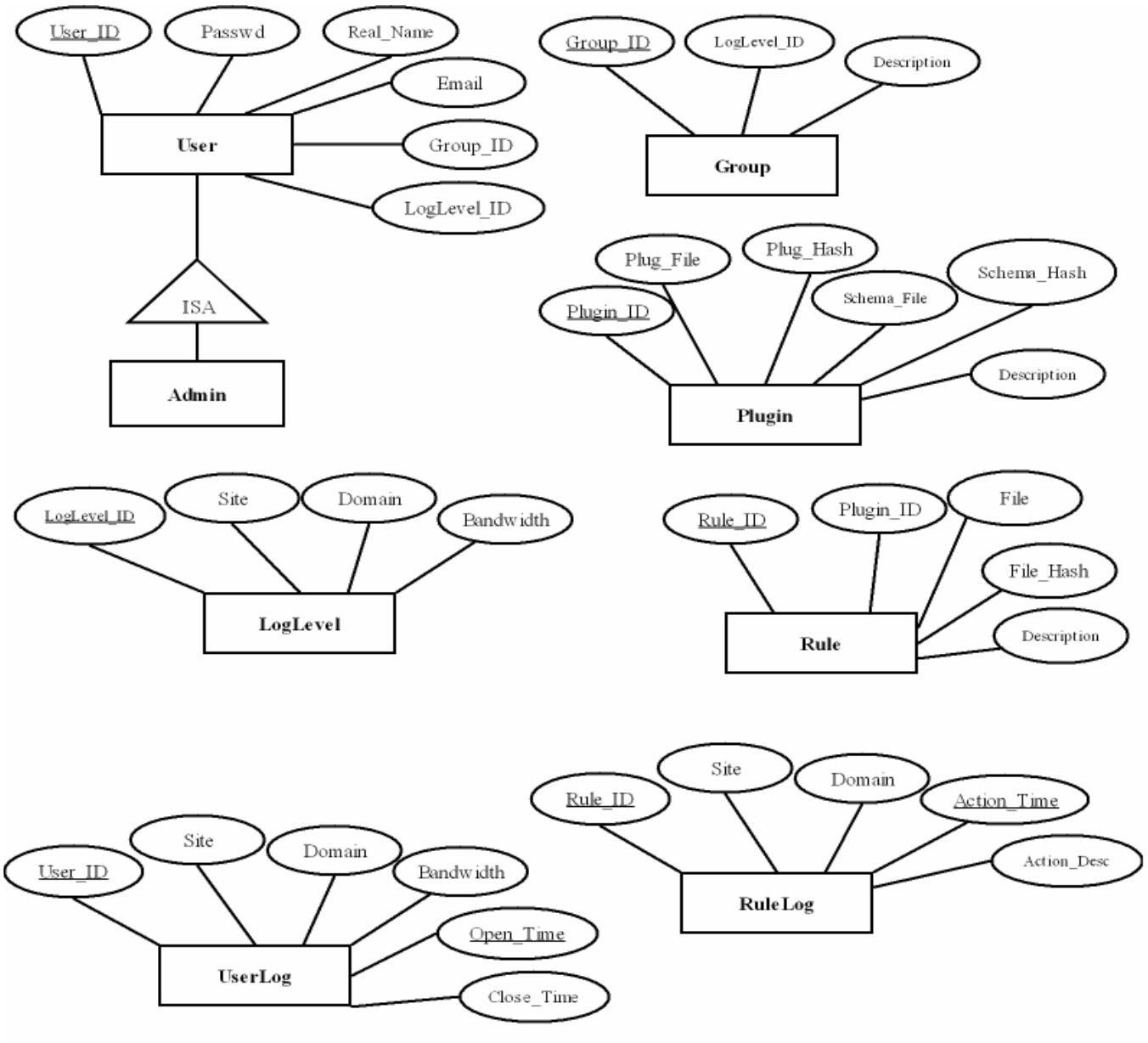
RuleLog entity will store the logs of rule actions. It has a similar schema to UserLog entity.

- Rule_ID
- Site
- Domain

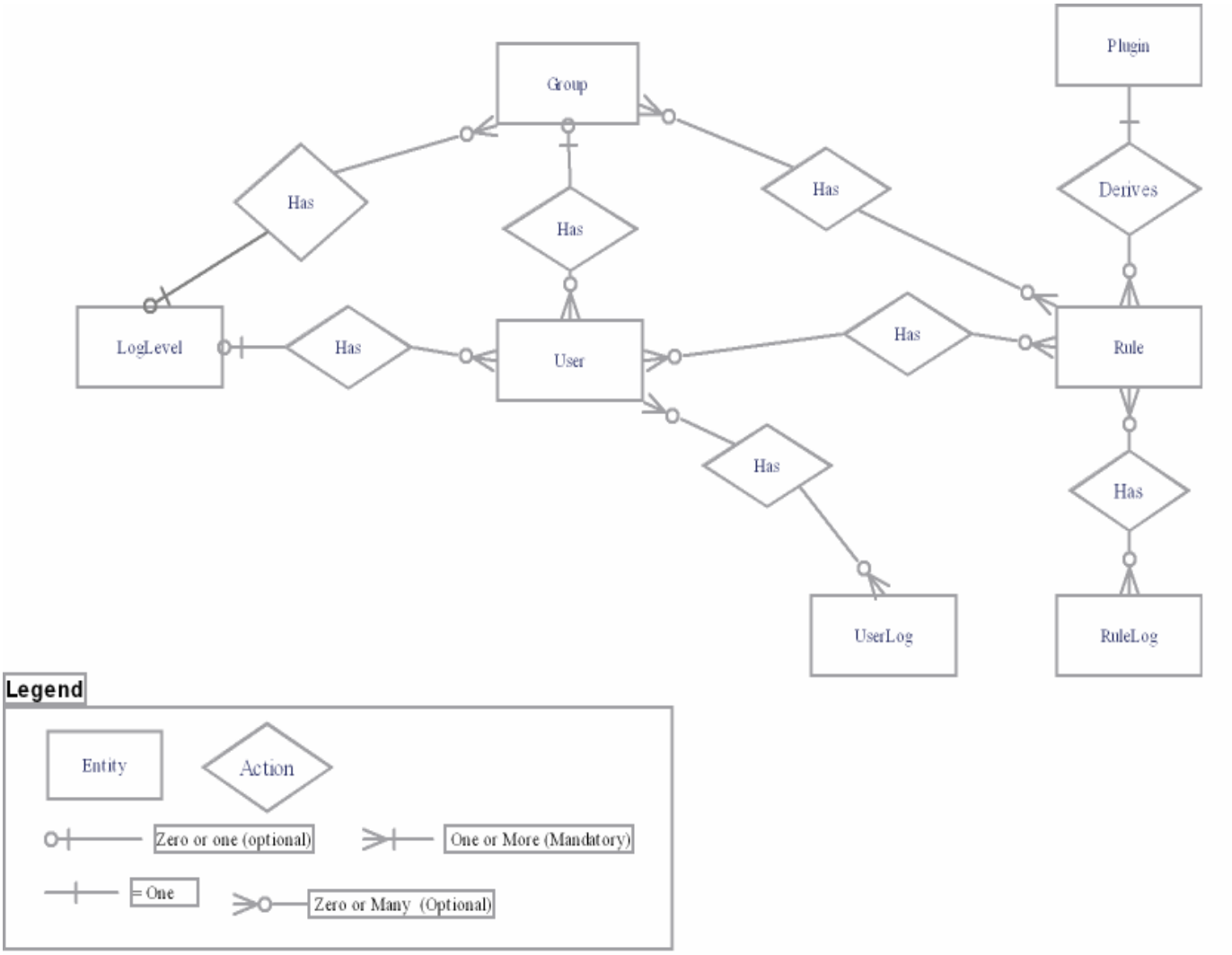
- Action_Time
- Action_Desc

Rule_ID and the *Action_Time* together will form the primary key. *Rule_ID* is the foreign key and reference to *Rule* entity. *Site* is the string to hold the site address that is visited. *Domain* is a string and it is the domain name of the visited site. *Action_Time* is the time of the action taken. Information about the action will be kept inside the *Action_Desc* attribute.

2.2 ER Diagrams



Data Objects



ER-Diagram

2.3 Data Dictionary

User

Name	User
Alias	-
Where / How used	The people that will use the system
Description	Every actor using the system is defined to be a user.

User_ID

Name User_ID

Alias -

Where / How used The users will enter their User_ID together with their passwords to log into the system. Administrators will have permission to access to control panel page with their passwords.

Description Every user has a unique User_ID.

Passwd

Name Passwd

Alias -

Where / How used While logging into the system

Description The password is to secure the system. Unauthorized users cannot access the control panel page. The user's password is converted to MD5 hash and checked with the one in the database.

Admin

Name Admin

Alias Administrator

Where / How used -

Description Admin is a special type of user who has all privileges. He/she will be able to access the control panel page. Then he/she will:

- add/remove users
- change users' and groups' logging and filtering rules
- check logs
- add/remove rules

Group

Name Group

Alias -

Where / How used Create Groups and add users to groups.

Description Groups will be created by the 'Admin' and they will be used to assign common logging levels and rules to many people at once. It will make the user management easier.

Plugin

Name	Plugin
Alias	-
Where / How used	While creating new rules.
Description	After a Plugin is written, it defines a schema and a proper template will be generated in order to create new rules from this Plugin.

Rule

Name	Rule
Alias	-
Where / How used	They are defined using an existing Plugin and assigned to users and groups.
Description	Admin defines the rule on the web control page and assign the rule to any user or group.

LogLevel

Name	LogLevel
Alias	-
Where / How used	While customizing the users' and groups' logging
Description	It will be possible to define logging levels by logging different items, which are site, site domain and bandwidth usage. Users' and groups' actions will be logged according to their logging level.

UserLog

Name	UserLog
Alias	-
Where / How used	While logging users actions
Description	UserLog contains log items: site as visited site's name; domain as site domain; bandwidth as data transfer size.

RuleLog

Name	RuleLog
Alias	-
Where / How used	When rules take action
Description	Rules will write their actions to database as 'RuleLog's. Visited site, site domain, action, time, action description will be saved.

2.4 Internal Software Data Structures

User

CREATE TABLE User
(
User_ID VARCHAR(15),
Group_ID VARCHAR(15),
LogLevel_ID INTEGER,
Passwd VARCHAR(32),
Real_Name VARCHAR(50),
Email VARCHAR(35),
PRIMARY KEY(User_ID),
FOREIGN KEY(Group_ID) REFERENCES Group,
FOREIGN KEY(LogLevel_ID) REFERENCES LogLevel
);

Data	***Type & Size***	***Format***
User_ID	Char-15	Text
Group_ID	Char-15	Text
LogLevel_ID	Integer	Number
Real_Name	Char-50	Text
Email	Char-35	Text
Password	Char–32	Text

CREATE TABLE Admin
(
User_ID VARCHAR(15),
PRIMARY KEY(User_ID),
FOREIGN KEY(User_ID) REFERENCES User
);

Data	***Type & Size***	***Format***
User_ID	Char-15	Text

CREATE TABLE Group
(
Group_ID VARCHAR(15),
LogLevel_ID INTEGER,
Description VARCHAR(255),
PRIMARY KEY(Group_ID),
FOREIGN KEY(LogLevel_ID) REFERENCES LogLevel
);

Data	*Type & Size*	*Format*
Group_ID	Char-15	Text
LogLevel_ID	Integer	Number
Description	Char-255	Text

```
CREATE TABLE LogLevel
(
LogLevel_ID INTEGER,
Site CHAR(1),
Domain CHAR(1),
Bandwidth CHAR(1),
PRIMARY KEY(LogLevel_ID)
);
```

Data	*Type & Size*	*Format*
LogLevel_ID	Integer	Number
Site	Yes/No	Yes/No
Domain	Yes/No	Yes/No
Bandwidth	Yes/No	Yes/No

```
CREATE TABLE UserLog
(
User_ID VARCHAR(15),
Site VARCHAR(255),
Domain VARCHAR(255),
Bandwidth INTEGER,
Open_Time VARCHAR(255),
Close_Time VARCHAR(255),
PRIMARY KEY(User_ID, Open_Time),
FOREIGN KEY(User_ID) REFERENCES User
);
```

Data	*Type & Size*	*Format*
User_ID	Char-15	Text
Site	Char-255	Text
Domain	Char-255	Text
Bandwidth	Integer	Number
Open_Time	Char-255	Text
Close_Time	Char-255	Text

```
CREATE TABLE Plugin
(
Plugin_ID VARCHAR(15),
Plug_File VARCHAR(255),
Schema_File VARCHAR(255),
Plug_Hash VARCHAR(32),
Schema_Hash VARCHAR(32),
Description VARCHAR(255),
PRIMARY KEY(Plugin_ID)
);
```

Data	*Type & Size*	*Format*
Plugin_ID	Char-15	Text
Plug_File	Char-255	Text
Schema_File	Char-255	Text
Plug_Hash	Char-32	Text
Schema_Hash	Char-32	Text
Description	Char-255	Text

```
CREATE TABLE Rule
(
Rule_ID VARCHAR(15),
Plugin_ID VARCHAR(15),
File VARCHAR(255),
File_Hash VARCHAR(32),
Description VARCHAR(255),
PRIMARY KEY(Rule_ID),
FOREIGN KEY(Plugin_ID) REFERENCES Plugin
);
```

Data	*Type & Size*	*Format*
Rule_ID	Char-15	Text
Plugin_ID	Char-15	Text
File	Char-255	Text
File_Hash	Char-32	Text
Description	Char-255	Text

```
CREATE TABLE RuleLog
(
Rule_ID VARCHAR(15),
Site VARCHAR(255),
Domain VARCHAR(255),
Action_Time VARCHAR(255),
Action_Desc VARCHAR(255),
PRIMARY KEY(Rule_ID, Action_Time),
FOREIGN KEY(Rule_ID) REFERENCES Rule
);
```

Data	*Type & Size*	*Format*
Rule_ID	Char-15	Text
Site	Char-255	Text
Domain	Char-255	Text
Action_Time	Char-255	Text
Action_Desc	Char-255	Text

```
CREATE TABLE UserRule
(
User_ID VARCHAR(15),
Rule_ID VARCHAR(15),
PRIMARY KEY(User_ID, Rule_ID),
FOREIGN KEY(User_ID) REFERENCES User,
FOREIGN KEY(Rule_ID) REFERENCES Rule
);
```

Data	*Type & Size*	*Format*
User_ID	Char-15	Text
Rule_ID	Char-15	Text

```
CREATE TABLE GroupRule
(
Group_ID VARCHAR(15),
Rule_ID VARCHAR(15),
PRIMARY KEY(Group_ID, Rule_ID),
FOREIGN KEY(Group_ID) REFERENCES Group,
FOREIGN KEY(Rule_ID) REFERENCES Rule
);
```

Data	*Type & Size*	*Format*
Group_ID	Char-15	Text
Rule_ID	Char-15	Text

2.5 Database Description

The database management system we are using for IronCurtain is SQLite. Tables will be created and be filled using Python's SQLite library. Database will store all of the information of the user's actions as logs. Also, the logging level, user and rule relation are stored in the database. When the system needs retrieving data, sql queries are used to get the necessary records.

2.6 Database Normalization

The database in our software is designed avoiding redundancy cases and we tried to suit them to the BCNF notation. To obey these rules we did some modifications over the real data tables. Instead of creating separate tables for each relation, we added a new attribute to one of the entities of the relation that is, a foreign key and reference to the other table. One of the most important modifications is assigning a "LogLevel_ID" and "Group_ID" to the "User" table. This was done to simplify the interaction between "User", "Group" and "LogLevel" tables. By this modification, we avoided redundancy of the tables. As a result, there are no insertion, update and deletion anomalies. Moreover, these will ease the queries for relations. Since more than one rule can be related with a user, we have to

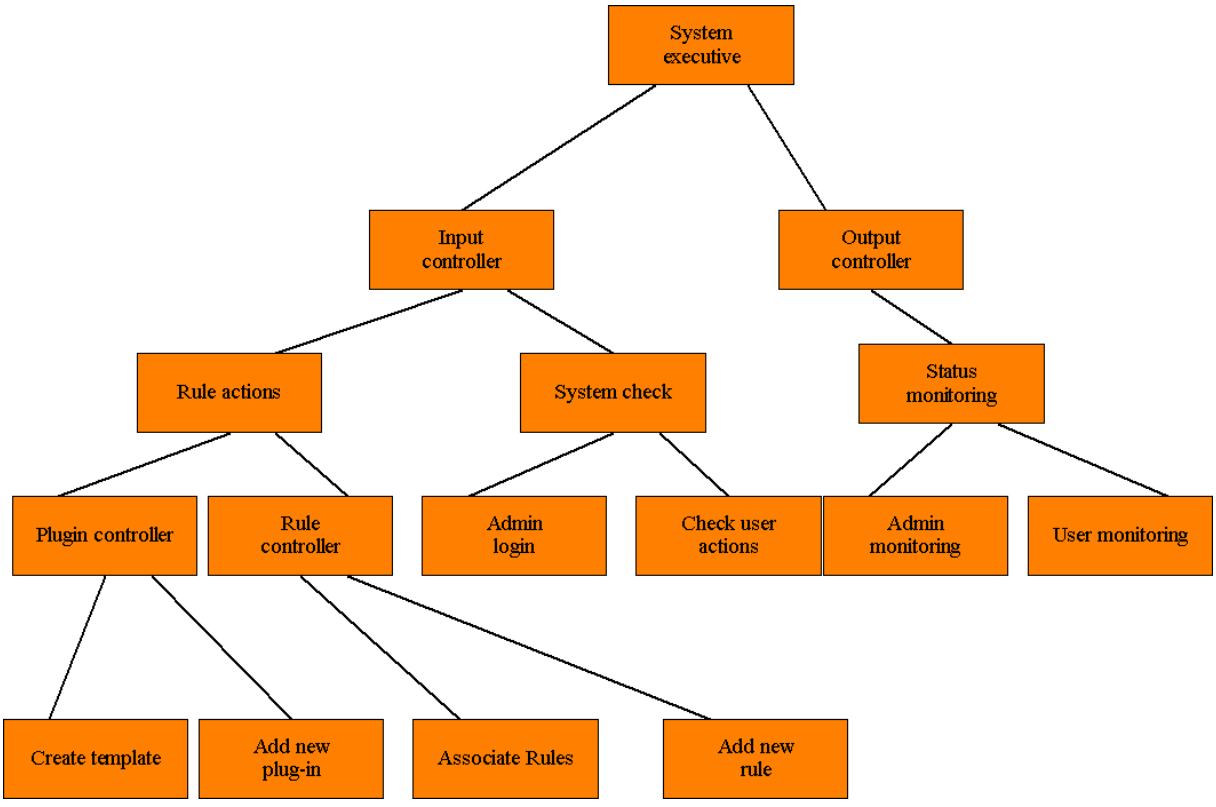
keep another table “UserRule”. The same approach is used for “Group” and “Rule” tables to form “GroupRule” table.

3. Architectural and Component-level Design

This section gives details about program structure, components, and software interface.

3.1 Structure Chart

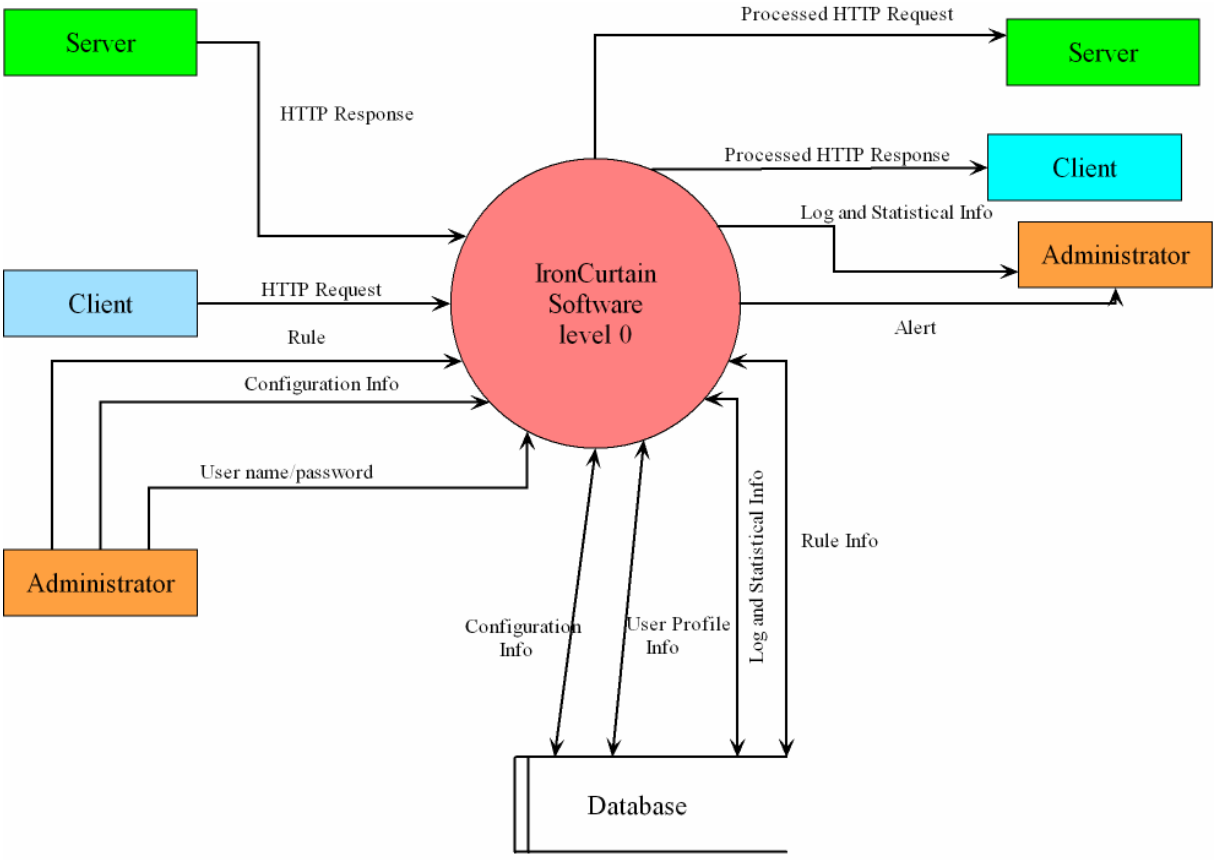
Our project has mainly into two modules which are named as input controller and output controller. Our system is also responsible of the appropriate coordination of these main parts and the system’s maintainability with the ultimate updates.



3.2 DFD

In this section, the functional model of IronCurtain is presented. It is composed of process specifications and DFD of the three levels (Level0, Level1 and Level2) of the system.

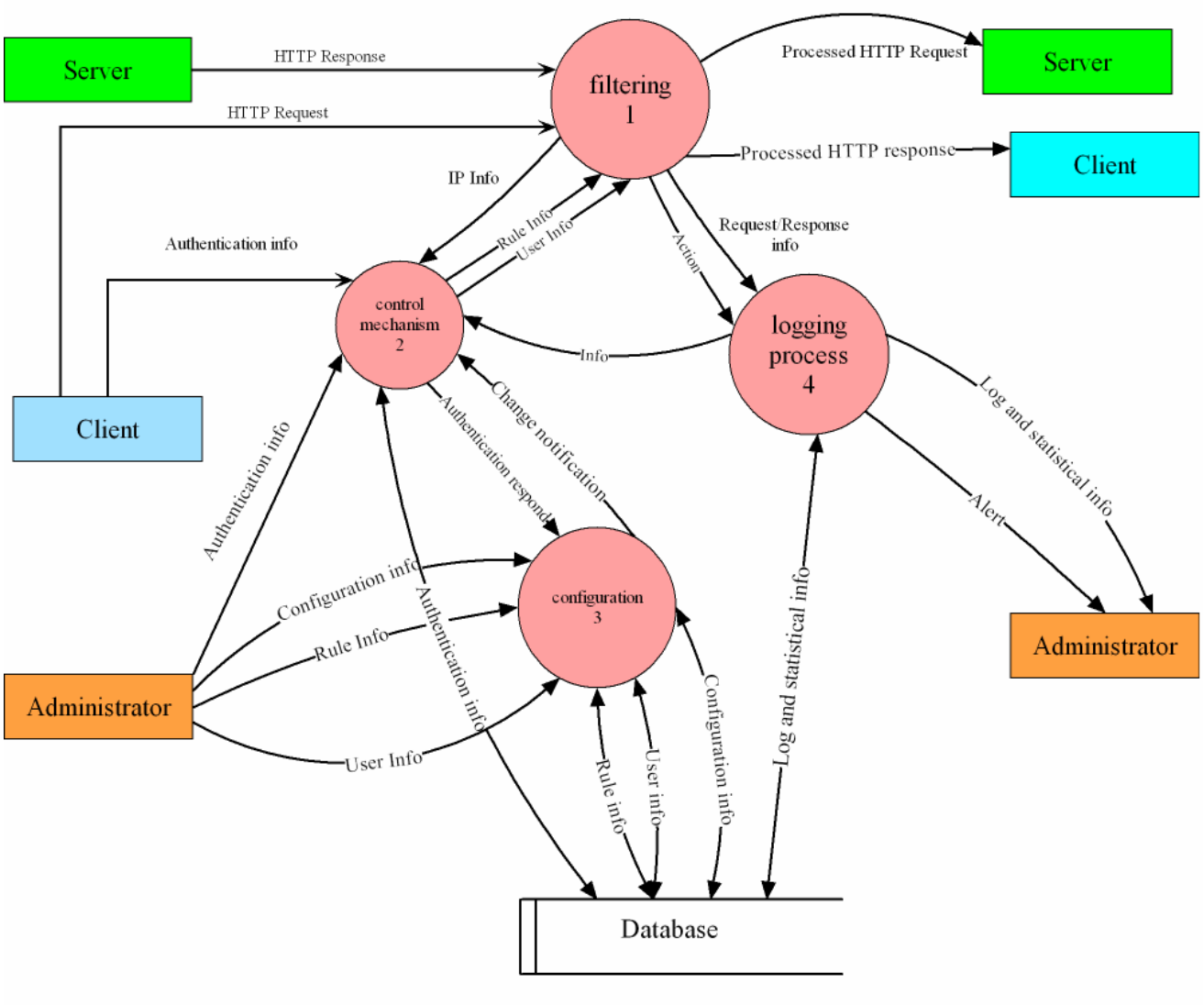
3.2.1 DFD Level 0



As seen from the diagram, we have three users: server, client and administrator. The main functionalities of the users are shown on the diagram. According to the requests IronCurtain writes some information to the database or process it and give a response to the users.

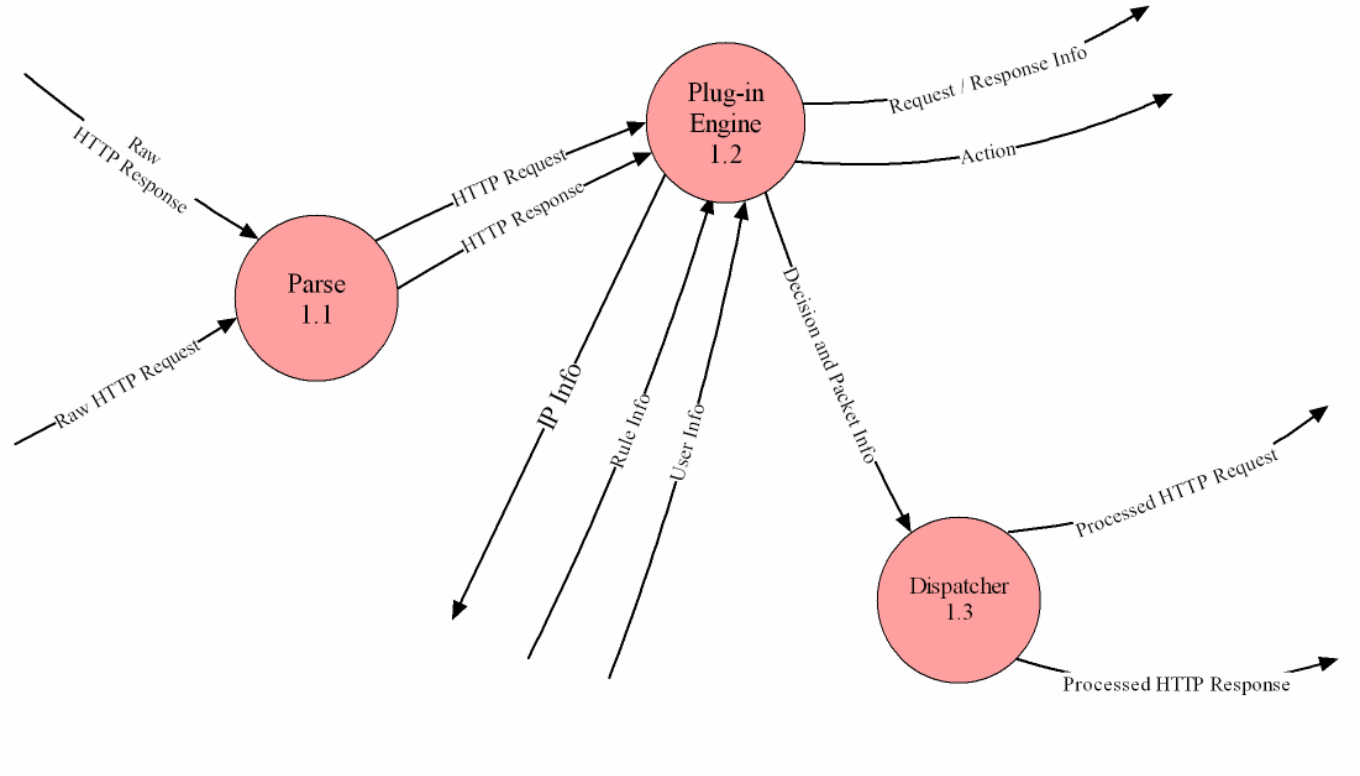
3.2.2 DFD Level 1

This is a more detailed diagram of IronCurtain. The main parts of IronCurtain are filtering, control mechanism, configuration and logging process. These parts communicate with users and each other.



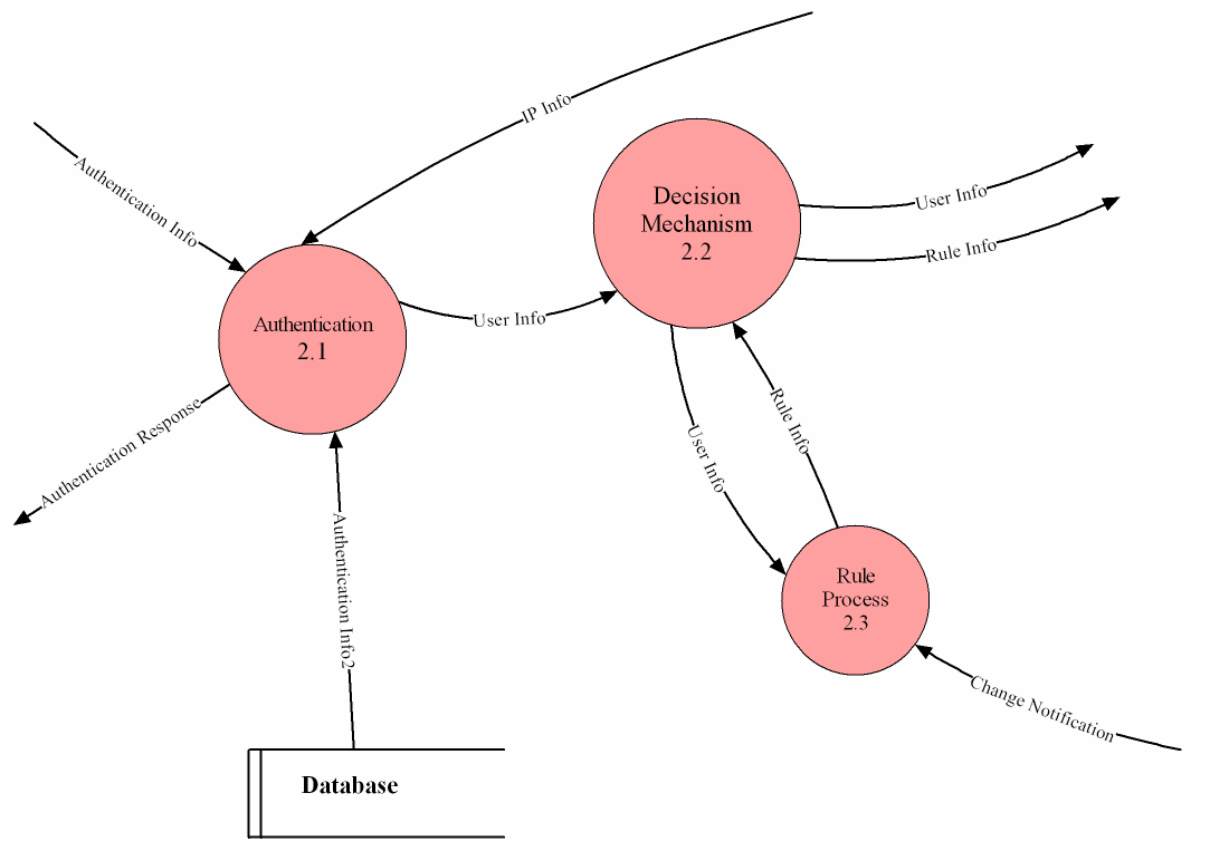
3.2.3 DFDs Level 2

Filtering



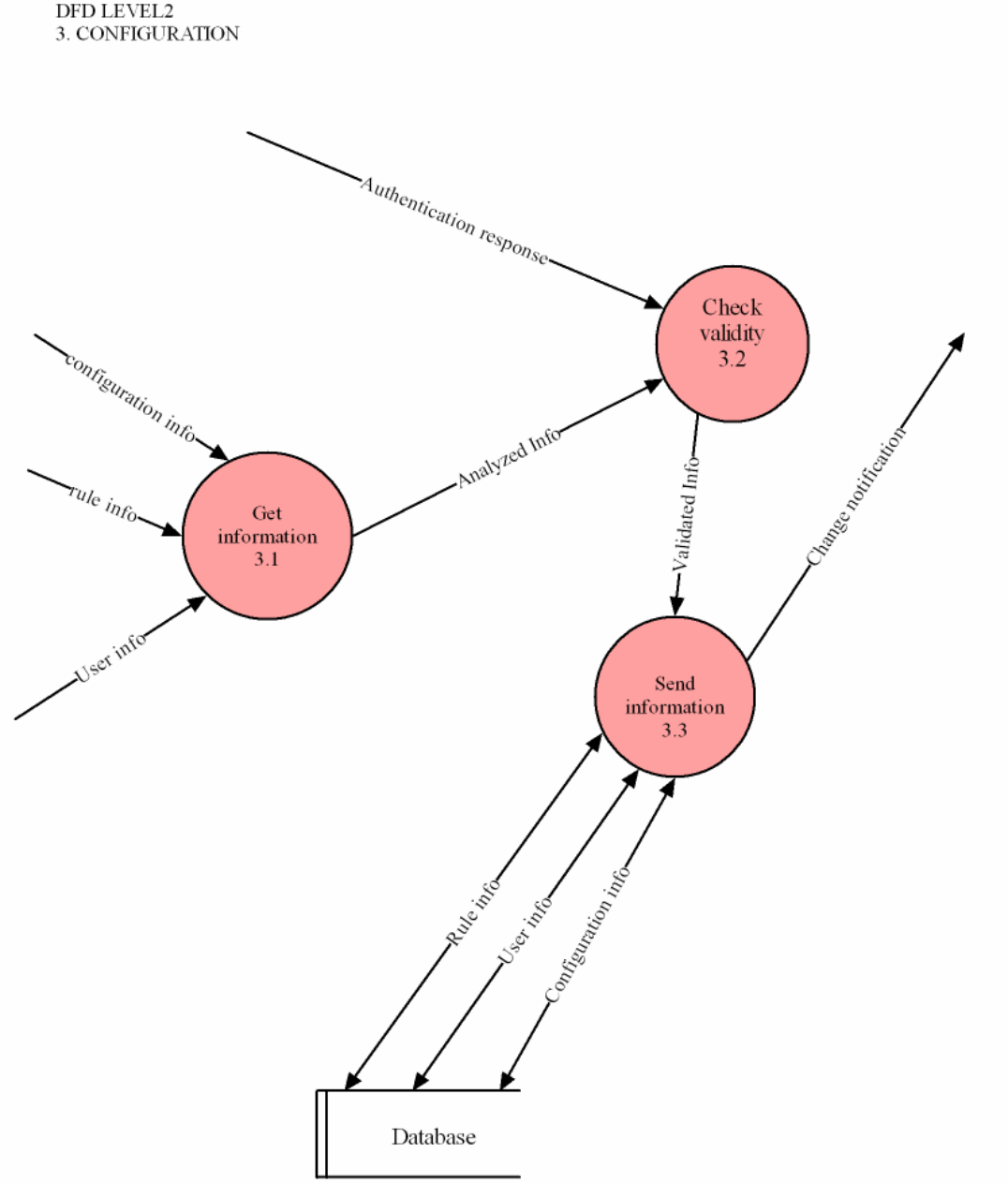
This is a more detailed diagram of the filtering part in IronCurtain. This is the most important part because all the requests and responses are processed here. First the requests and responses are parsed and then it is analyzed. In analyze part the validity of the request and response is checked by the control mechanism. After analyzing the processed request or response are sent by dispatcher to the users.

Control Mechanism



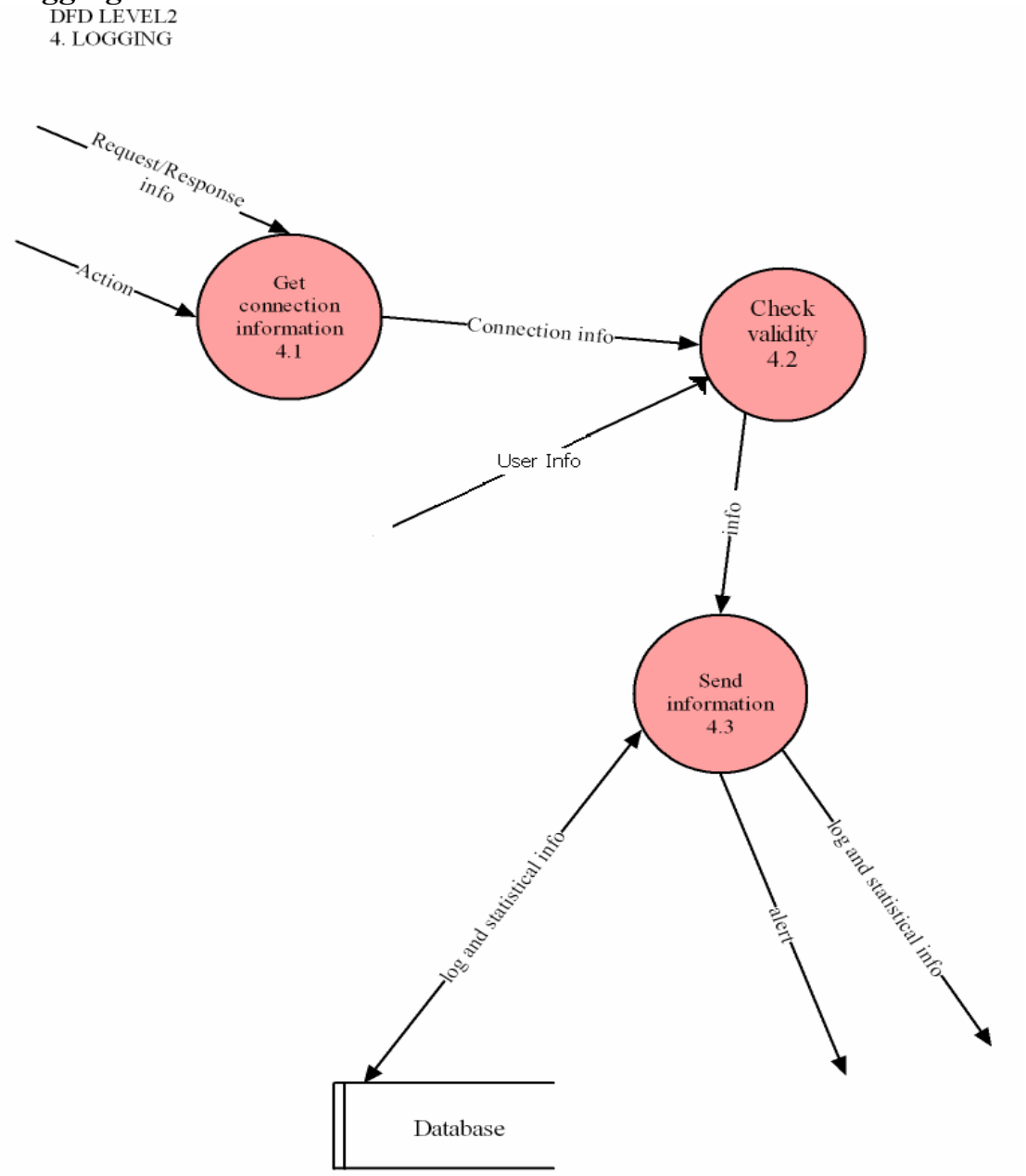
Control mechanism briefly checks the access rights and user authentication. It is connected to all the other modules. It gets the info from the other parts and checks whether this user have the right to do that action or not. According to that it sends an answer of approval/disapproval.

Configuration



Configuration part is used for changing the properties of users, rules and main configuration of the proxy. It checks the access rights of the user by using control mechanism.It writes the information to the database. Only administrator user can use that part of the IronCurtain.

Logging



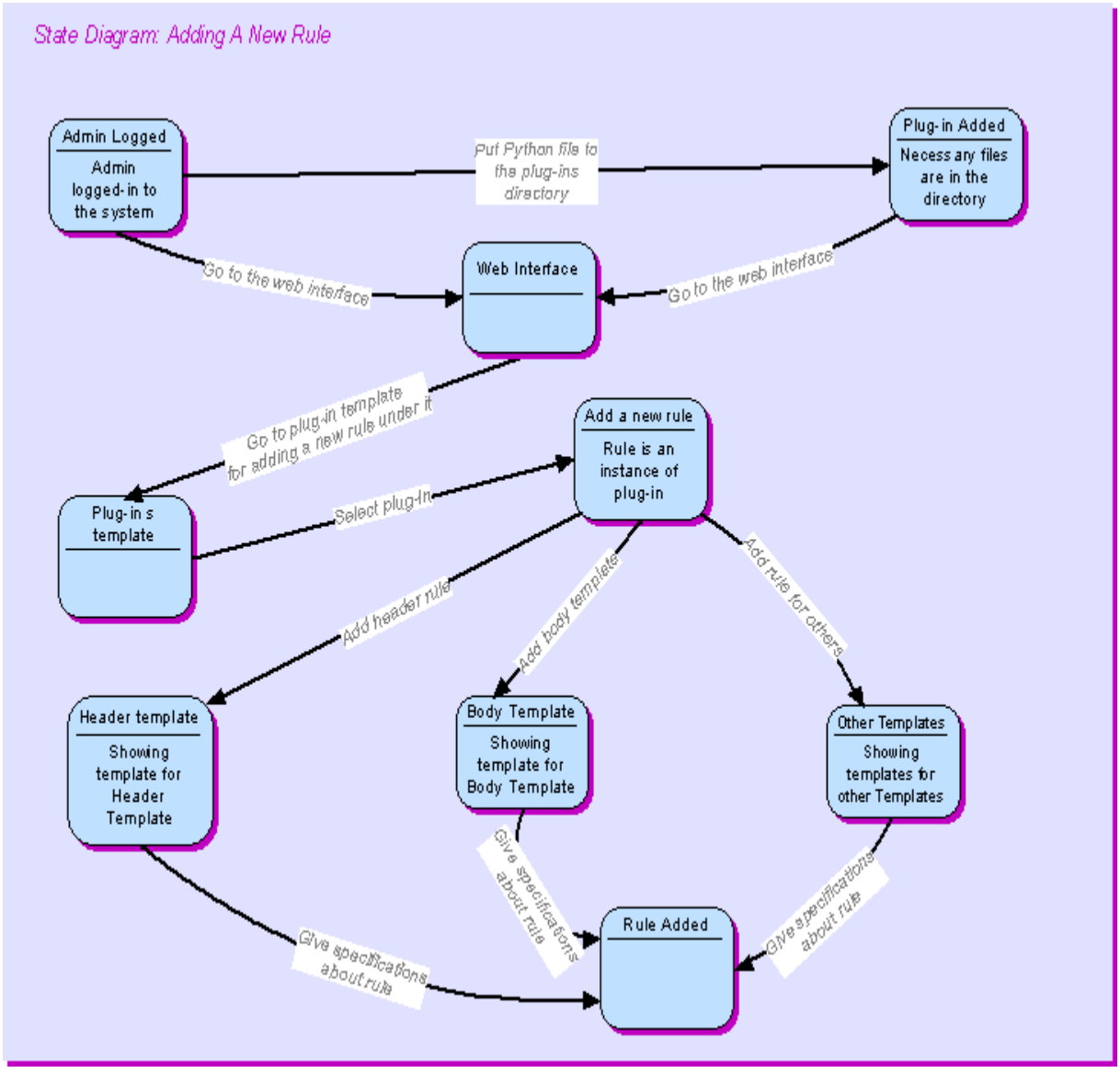
Logging process is used for logging all the activities of the user and also alerting if there is a violation of a predefined condition. All the request and response info is provided to logging process by filtering part. The logging process sends log and statistical info to the database and administrator user.

3.3 State Transition Diagrams

3.3.1 State Diagram Adding a Rule:

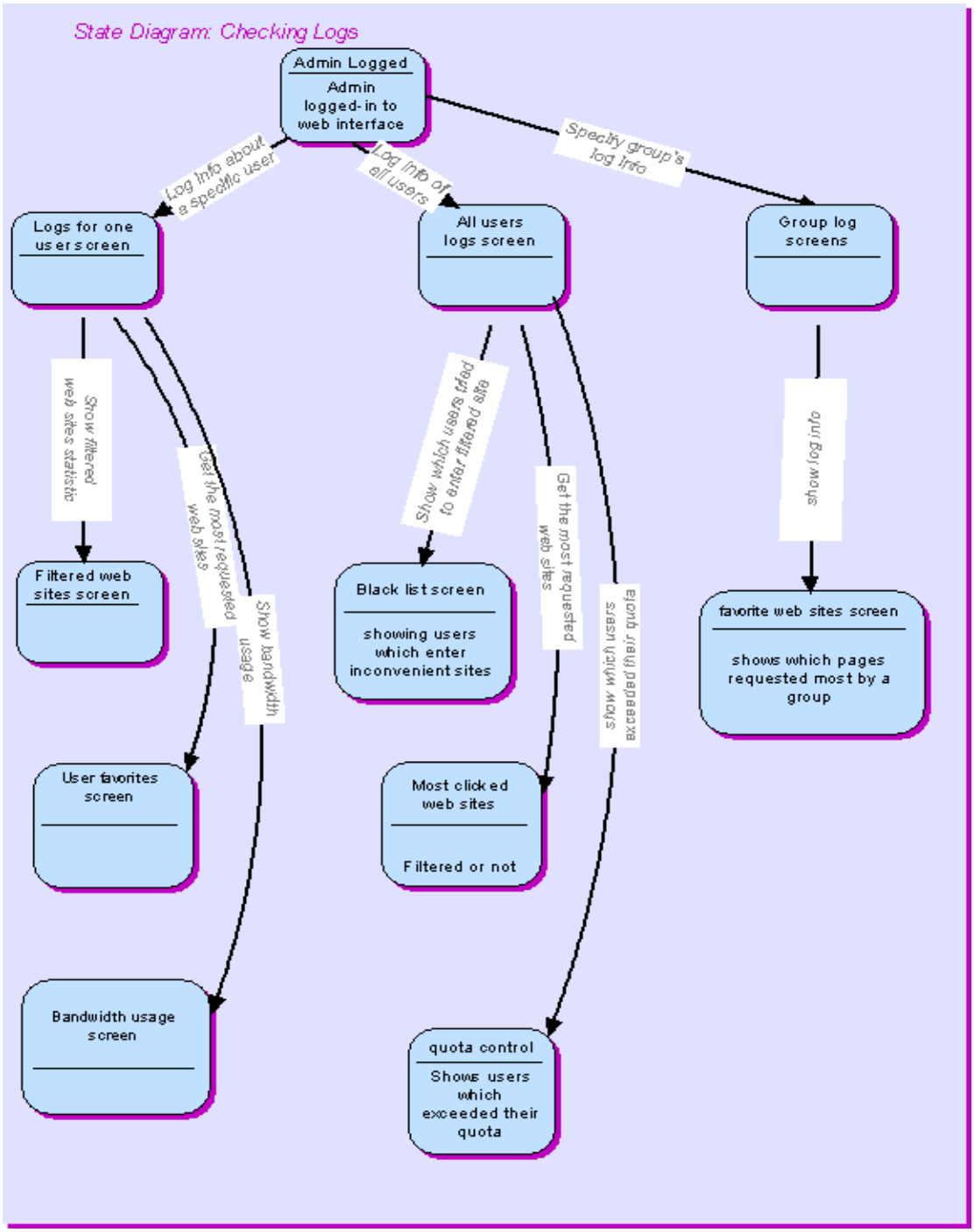
The figure given below explains how a rule is added to the system by the administrator by using web interface.
Adding a plug-in is represented below, too.

For a new rule to be added, there must exist a plug-in for it.

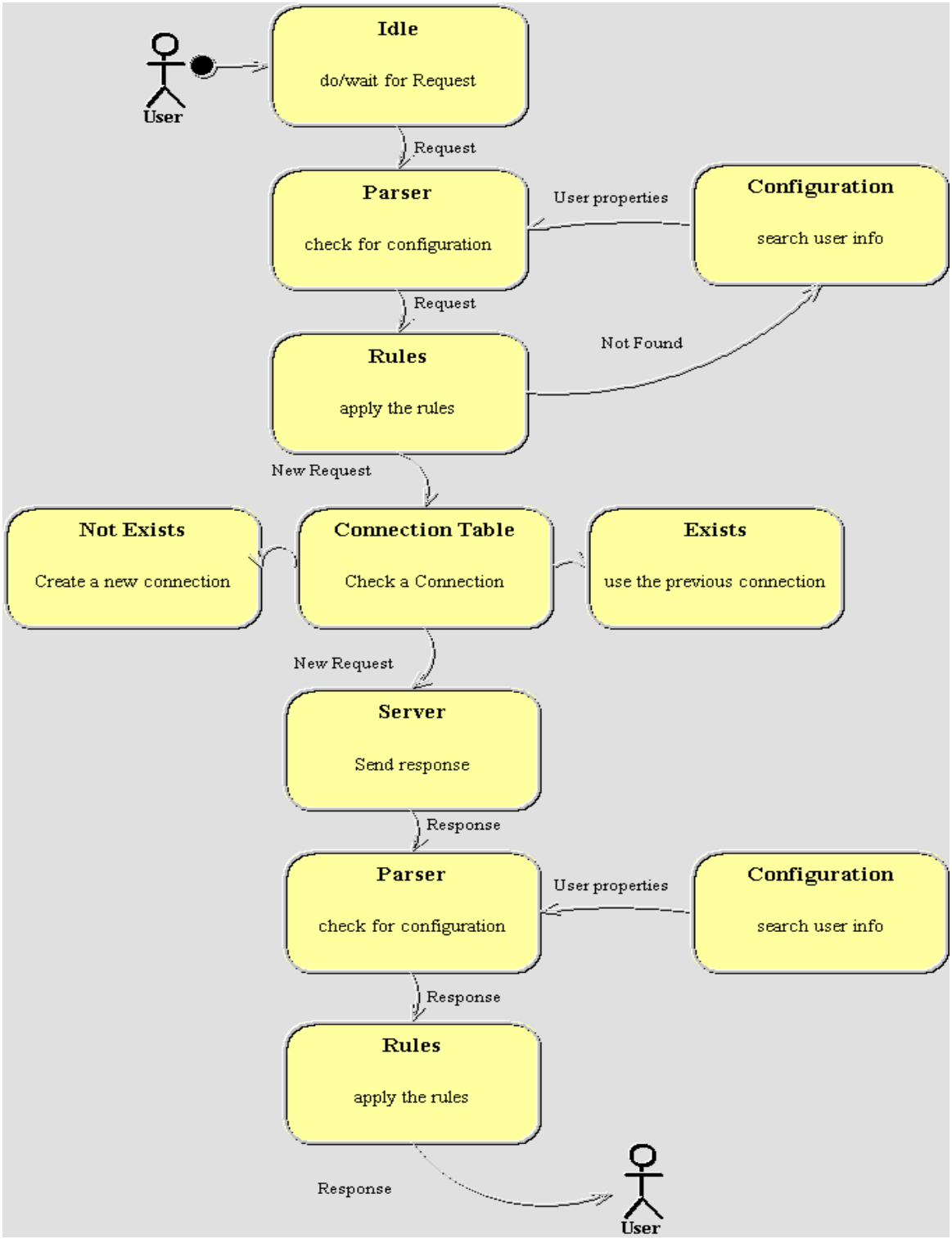


3.3.2 Checking Logs State Diagram:

The diagram below gives the state diagram for checking logs, starting from the administrator web interface.



3.3.3 Request – Response Diagram



- When a new Request comes, IronCurtain will create a new thread to our thread pool.
- Then it will send request to our parser.
- Parser will get information from configuration module. Information is about which rules to apply for this user? The bandwidth limit of the user. The time limit of the user.
- After getting information about user parser sends these to Rules. In rules module it will select the proper rules and apply them to the request in parallel.
- Then Rules will send the updated request to the connection table.
- There, the connection will be checked if it is opened before, it will use the previous connection. (HTTP 1.1 persistent connection property) If not it will create a new connection.
- Then the request will be sent to the server.
- Server will send the response.
- The response will be taken by Parser. It gets the information from configuration module.
- Then it will send the response to Rules.
- Rules will apply the proper operations to the response and then send the new response to the user.

3.4 Description of Components

3.4.1 Plug-in Architecture

Let's go over the process of creating a new rule.

We want to write a rule that changes the value of the “User-Agent” header in an HTTP request to “Protoxy/0.0.1” if the site is in a German or French domain. Let's assume that no plug-in has the necessary infrastructure for such a task (of course, in Protoxy there will be such an infrastructure), so we will also write a plug-in.

A plug-in is a python file with a few quirks. In the file, there will be one class with the same name as the file(case-sensitive), which defines at least one function act_on with no arguments, and a static variable called Args. Also, the constructor of the class must take one argument, a dictionary (also known as, associative array).

act_on: is the function that gets when the rule works. It can optionally return a constant value to indicate the course of action for IronCurtain.
The constants are:

ACTION_BLOCK: A blocked request is never answered; the browser's connection is closed and the browser reports an error or uses a "broken image" icon.
ACTION_ANSWER: An answered request is handled directly by the proxy. In a sense, the proxy acts as a web server.
ACTION_REDIRECT: A redirected request is sent to a location other than for what location it was originally intended.

ACTION_FORWARD: A forwarded request is sent to the web server for which it was originally intended.

If act_on function does not return a value, the action taken is ACTION_FORWARD.

Args: should be declared as a list of tuples that lists the arguments to the rule. For example;

Args = [(“MatchURLs”, “Match urls”, “array”, 1), (“Actions”, “Actions”, “enum”, (“Add”, “Replace”, “Delete”)), (“HeaderName”, “Header Name”, “string”), (“HeaderValue”, “Header Value”, “string”)]

Args is a matching between argument names and their types. An element in the list is a 3-tuple or a 4-tuple.
The first is the actual name of the argument.
The second is the text as it will look on the HTML Page.
The 3rd and 4th are indicators of data types. Only 4 structures are allowed: int, string, enum, and array. The first two (int and string) are obvious.
Of the rest enum is an n-tuple of strings.
Last, array is a list of strings. Two numbers follow array declaration to indicate its size. The first is the minimum size of array, the second is the maximum. If the second is omitted, it is assumed that maximum is infinity.

The argument to the constructor: is a dictionary. Params will contain the instances of arguments defined by Args. Following the previous example;

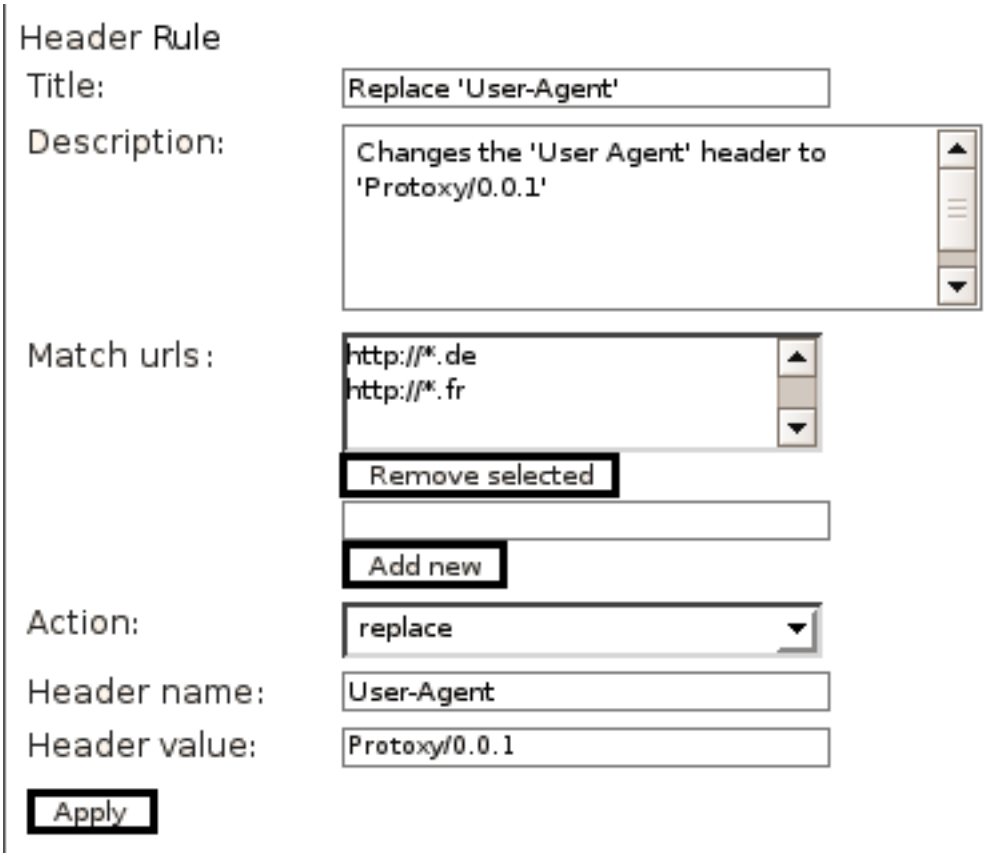
(Assuming such a declaration) def __init__(self, Params):

Params["Actions"] -> "Replace"
Params["HeaderName"] -> "User-Agent"
Params["HeaderValue"] -> "Protoxy/0.0.1"
len(Params["MatchURLs"]) -> 2
Params["MatchURLs"][0] -> "http://*.de"
Params["MatchURLs"][1] -> "http://*.fr"

Once we are happy with our code, we upload it as a new plug-in over IronCurtain's web interface. This creates a new directory under plugins/ directory with the name of the plug-in. Let's assume we named it HeaderPlugin. Under that directory
it puts the file HeaderPlugin.py and a hidden xml file with the name HeaderPlugin_$schema$.xml. The schema file is an explicit from the Args variable. Continuing our example, it would look like this;

```
<Plugin ID="1">
   <MatchURLs displayname="Match urls" type="array" minsize="1"/>
   <Actions displayname="Actions" type="enum" choices="3">
      <choice>
         Add
      </choice>
      <choice>
         Replace
      </choice>
      <choice>
         Delete
      </choice>
   <HeaderName displayname="Header Name" type=string/>
   <Heade_Value displayname="Header Value" type=string/>
</Plugin>
```

Once this file is generated, the web interface will show the new plug-in immediately. To add the rule we select the 'Add New Rule' tab and click on our previously defined plug-in as the template.



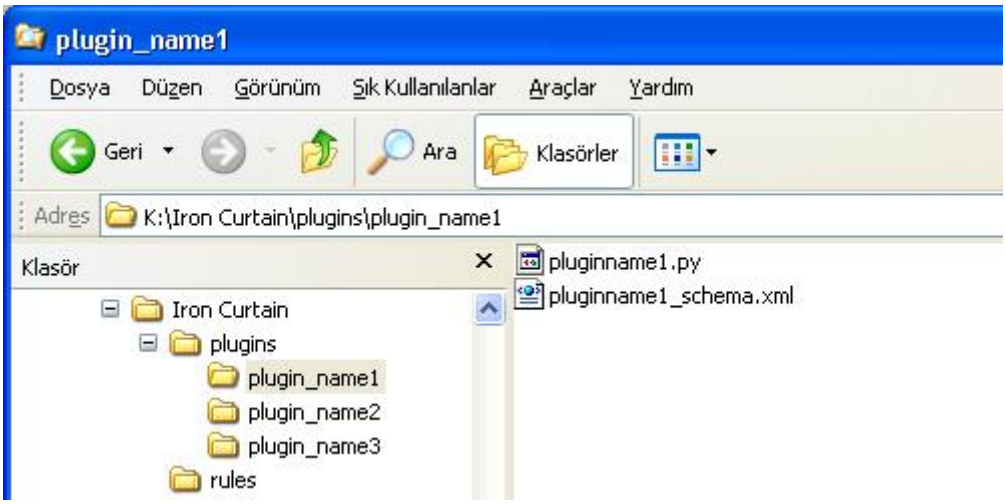
When we click the apply button, the web interface generates an XML document to store

the relevant data. The document is put in under rules/. Our rule would look like this:

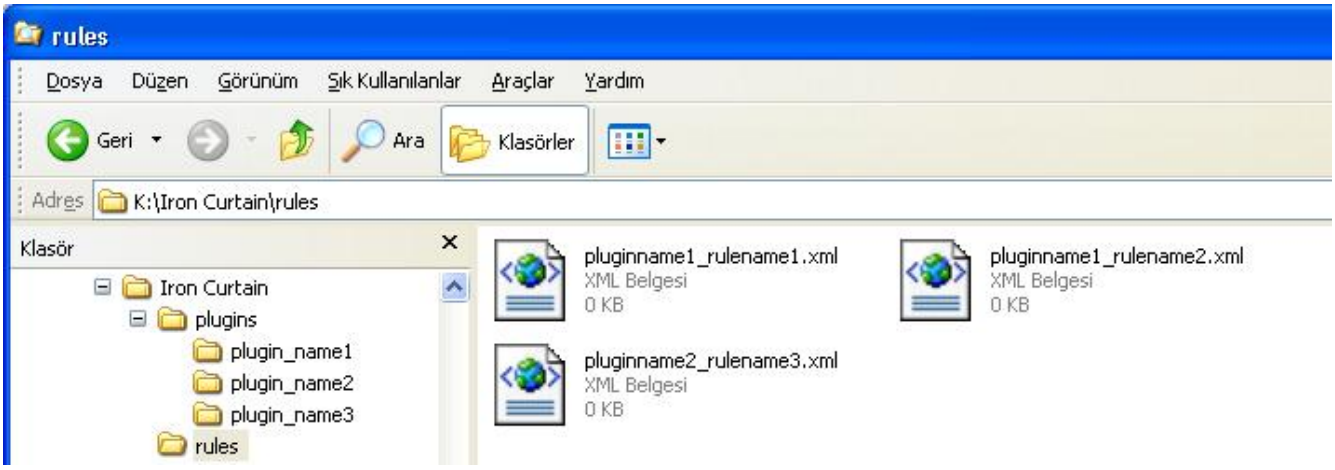
```
<Rule pluginID=”1” ID=”1”>
   <Title type="string">
      Replace 'User-Agent'
   </Title>
   <Description type="string">
      Replaces 'User-Agent' header to 'Protoxy/0.0.1
   </Description>
   <MatchURLs type="array" size="2">
      <element type="string">
         http://*.de
      </element>
      <element type="string">
         http://*.fr
      </element>
    </MatchURLs>
    <Actions type="string">
      Replace
    </Actions>
    <HeaderName type="string">
      User-Agent
    </HeaderName>
    <HeaderValue type="string">
      Protoxy/0.0.1
    </HeaderValue>
</Rule>
```

3.4.2 Plug-in and Rule directory structures

The directory structure of our plug-ins will be as follows.



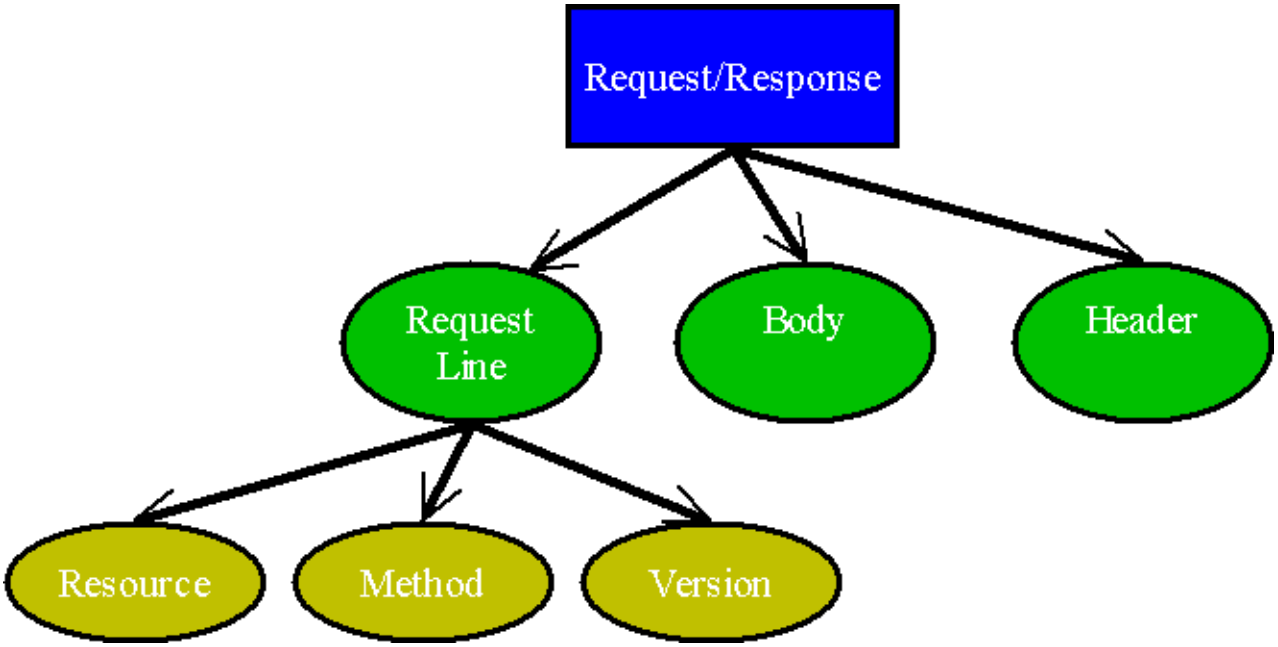
The files in a plug-in directory



The files in rules directory

3.4.3 Request / Response architecture

The structure of a request or response will be like as the following schema:



3.4.4 Data Dictionary

Name	Authentication Info
Aliases	Authentication Info2 User name/Password
Where used/ How used	Control Mechanism - Authentication 2.1 (input/output) Client (output) Administrator (output) Database (input/output)
Description	This data is the username and the password.
Format	An aggregate object consisting of User_ID and Passwd of type string.

Name	Authentication Response
Aliases	None
Where used/ How used	Control Mechanism - Authentication 2.1 (output) Configuration – Check Validity 3.2 (input)
Description	Result of the username, password and administrative rights checking
Format	An aggregate object consisting of User_ID and Passwd of type string and isAdmin of type boolean.

Name	HTTP Request
Aliases	Raw HTTP Request
Where used/ How used	Filtering – Parse 1.1 (input/output) Filtering – Plug-in Engine 1.2 (input) Client (output)
Description	Request coming from users is sent to Parser and Plug-in Engine to be filtered.
Format	An HTTP request object consisting of Line, Header and Body

Name	Processed HTTP Request
Aliases	None
Where used/ How used	Filtering – Dispatcher 1.3 (output) Server (input)
Description	Filtered HTTP Request
Format	An HTTP request object consisting of Line, Header and Body

Name	HTTP Response
Aliases	Raw HTTP Response
Where used/ How used	Filtering – Parse 1.1 (input/output) Filtering – Plug-in Engine 1.2 (input) Server (output)
Description	Response coming from server is sent to parser and plug-ins to be filtered.
Format	An HTTP response object consisting of Line, Header and Body

Name	Processed HTTP Response
Aliases	None
Where used/ How used	Filtering – Dispatcher 1.3 (output) Client (input)
Description	Filtered HTTP Response is sent to client.
Format	An HTTP response object consisting of Line, Header and Body.

Name	Request/Response Info
Aliases	None
Where used/ How used	Filtering – Plug-in Engine 1.2 (output) Logging – Get Connection Information 4.1 (input)
Description	Information to be logged after the analysis
Format	Request or Response object

Name	IP Info
Aliases	None
Where used/ How used	Filtering – Plug-in Engine 1.2 (output) Control Mechanism – Authentication 2.1 (input)
Description	To check User filtering rules Plug-in sends the user IP to Control Mechanism.
Format	IP is composed of 4 integers.

Name	User Info
Aliases	User Profile Info
Where used/ How used	Filtering – Plug-in Engine 1.2 (input) Control Mechanism – Decision 2.2 (input/output) Control Mechanism – Authentication 2.1 (output) Control Mechanism – Rule Process 2.3 (input) Configuration – Send information 3.3 (input/output) Configuration – Get information 3.1 (input) Logging – Check Validity 4.2 (input) Database (input/output)
Description	User's information will be passed through the modules
Format	User Object consisting of User_ID, LogLevel_ID, Group_ID of type string.

Name	Rule Info
Aliases	None
Where used/ How used	Filtering – Plug-in Engine 1.2 (input) Control Mechanism – Decision 2.2 (input/output) Control Mechanism – Rule Process 2.3 (output) Configuration – Send information 3.3 (input/output) Configuration – Get information 3.1 (input) Database (input/output)
Description	A new rule is added to the system over the web interface by the

	Administrator. Rules are stored in the DB and they are sent to 'Configuration Module' to get into action.
Format	An object containing the Rule attributes. The object may vary depending on the plug-in that the rule is created from.

Name	Decision and Packet Info
Aliases	None
Where used/ How used	Filtering – Plug-in Engine 1.2 (output) Filtering – Dispatcher 2.2 (input)
Description	Actions and filtered HTTP packets are sent to Dispatcher in order to deliver related client and server.
Format	HTTP Request/Response Object

Name	Change Notification
Aliases	None
Where used/ How used	Control Mechanism – Rule Process 2.3 (input) Configuration – Send Information 3.3 (output)
Description	When a new rule is added to the system, its information will be sent to the Rule Process to use the rule immediately.
Format	An object containing Rule identifier.

Name	Configuration Info
Aliases	None
Where used/ How used	Configuration – Get Information 3.1 (input) Configuration – Send Information 3.3 (output) Administrator (output) Database (input/output)
Description	Administrator makes changes to the system configuration. Admin can change logging levels, group, user and rule interrelations over the web. The changes are written to DB.
Format	Configuration change object

Name	Log and Statistical Info
Aliases	None
Where used/ How used	Logging Process – Send Information 4 (input/output) Administrator (input) Database (input/output)
Description	The logs and statistical information from these logs will be stored in the database and showed to Administrator.
Format	A Log object containing the User_ID/Rule_ID, Site, Domain, Bandwidth and Action_Time (Open_Time, Close_Time), Action_Desc attributes. These are explained in Data Objects part.

Name	Alert
Aliases	None
Where used/ How used	Logging Process – Send Information 4.3 (output) Administrator (input)
Description	On predefined conditions depending on the actions and the logs, the Logging Process module will send a notification e-mail to Administrator.
Format	A text that describes the action or situation briefly.

Name	Action
Aliases	None
Where used/ How used	Filtering – Plug-in Engine 1.2 (output) Logging Process – Get Connection Information 4.1 (output)
Description	One of the 4 predefined actions is returned by the Plugin according to the action performed.
Format	Enumerated value corresponds to the action taken: ACTION_BLOCK, ACTION_ANSWER, ACTION_REDIRECT, ACTION_FORWARD

Name	Analyzed Info
Aliases	None
Where used/ How used	Configuration – Get Information 3.1(output) Configuration – Check Validity 3.2(input)
Description	After the incoming information is analyzed, it is sent to be validated with the authentication info.
Format	One of the Configuration, User, Rule objects

Name	Validated Info
Aliases	None
Where used/ How used	Configuration – Send Information 3.3(input) Configuration – Check Validity 3.2(output)
Description	After the information is validated, it is transferred to Send Information module.
Format	One of the Configuration, User, Rule objects

Name	Connection Info
Aliases	None
Where used/ How used	Logging – Get Connection Information 4.1(output) Logging – Check Validity 4.2(input)
Description	Connection Information is sent to Check Validity 4.2 module
Format	Request or Response Object and User identifier.

Name	Info
Aliases	None
Where used/ How used	Logging – Send Information 4.3(input) Logging – Check Validity 4.2(output)
Description	After the information is validated, it is transferred to Send Information module.
Format	Request or Response Object and User identifier.

3.4.5 Algorithmic Model (PDL)

Main
input none
output none
Loop forever
If there is a new request then
Check if the thread pool has any threads available
If there is a thread available then
Associate the thread with the request and ServeRequest(request) (The thread does the function Serve)
Else enlarge the thread pool and associate the thread with the request and ServeRequest(request)
Once the ServerRequest function is done, put the thread back to thread pool

ServeRequest
input request
output none
Parse request, get Request Line, Headers and Body if exists
Get the user IP of request
Learn the user ID from user IP, also get the list of rules that apply to that user
Action = None
For each rule in list of rules
Run rule
If rule returns a block action then
Action=block
Block the requst
Else if rule returns an answer action
Action=answer
Else if rule returns redirect and Action is not answer
Action=redirect
If action is None then action=forward
DoAction(request, action)

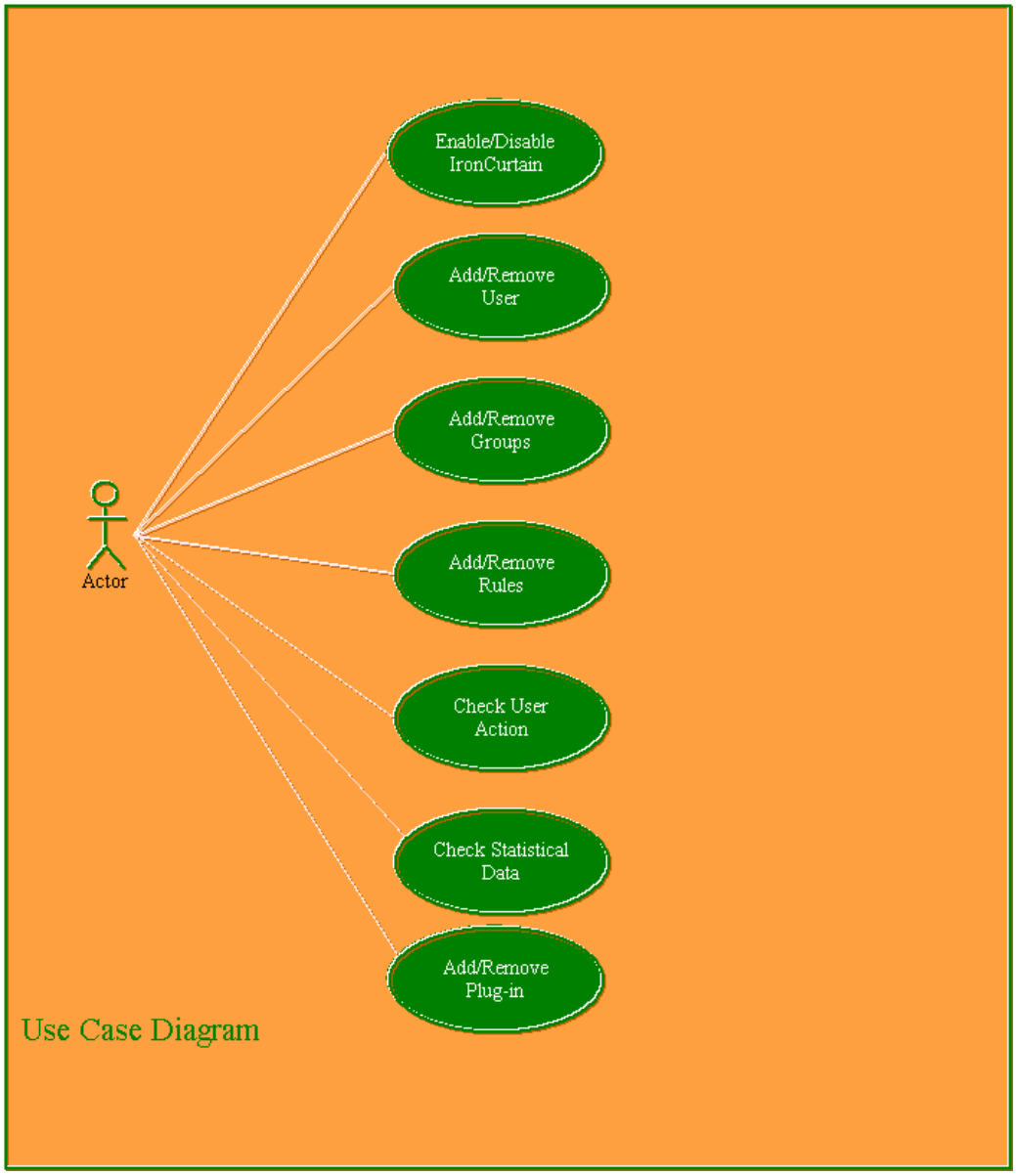
DoAction
input request, action
output none
If action is Forward then

```
            Check if a connection to the destination already exists
            If such a connection exists then
                  Send request to destination over that connection
            Else
                        Create a new connection to the destination
                        Send request to destination over that connection
      Else if action is redirect then
         Create a new request with uri being the destination
      Else if action is answer then
         generate response page and send to the client
      Loop forever
         If the response has arrived then
            break
      ServeResponse

ServeResponse
input none
output none
      Read the response
      Parse response, get Response Line, Headers and Body if exists
      action = None
      For each rule in list of rules
         Run rule
         If rule returns a block action then
            action=block
            Block the requst
         Else if rule returns an answer action
            action=answer
         Else if rule returns redirect and action is not answer
            action=redirect
      DoActionResponse(response, action)

DoActionResponse
input response, action
output none
      If action is Forward then
         Pass the response to the client
      Else if action is redirect then
         Create a new request with uri being the destination
      Else if action is answer then
          generate response page and send to the client
```

3.5 Use Case Diagram



4. User Interface Design

This section gives detailed information about user interfaces, interface design rules, components available and, UIDS.

4.1 Interface design rules

Interface design focuses on three areas of concern:

1. The design of interfaces between software modules;
2. The design of interfaces between the software and other nonhuman producers and consumers of information (i.e., other external entities);
3. The design of the interface between a human (i.e., the user) and the computer.

The important properties are the following;

- Easy to Learn
- Readability
- Easy navigate between interfaces

Our guidelines are provided below;

Data Display Guidelines

1. Consistency of display
2. Efficient assimilation of information by the user
3. Compatibility between data entry and display
4. Flexibility of user control

Menu Guidelines

1. Shallow, wide menus preferred over tall deep menus
2. User has access to all relevant items without referencing a manual
3. Logical item presentation sequence
    a. Numeric
    b. Alphabetic

4. Icons are harder to recognize than text during visual search
5. Ensure consistent navigation

Screen Formatting Guidelines

1. Focus on readability and user acceptability
2. Don’t clutter the screen (white space is free)
3. Choose pleasing color combinations
4. Use full width of the screen
5. Keep only relevant info on the screen

Guidelines for Effective Use of Color

1. Use color to group similar items
2. Limit the total number of colors
3. Watch out for bad color combinations (red/blue, blue/black)
4. Keep in mind people may view it in monochrome

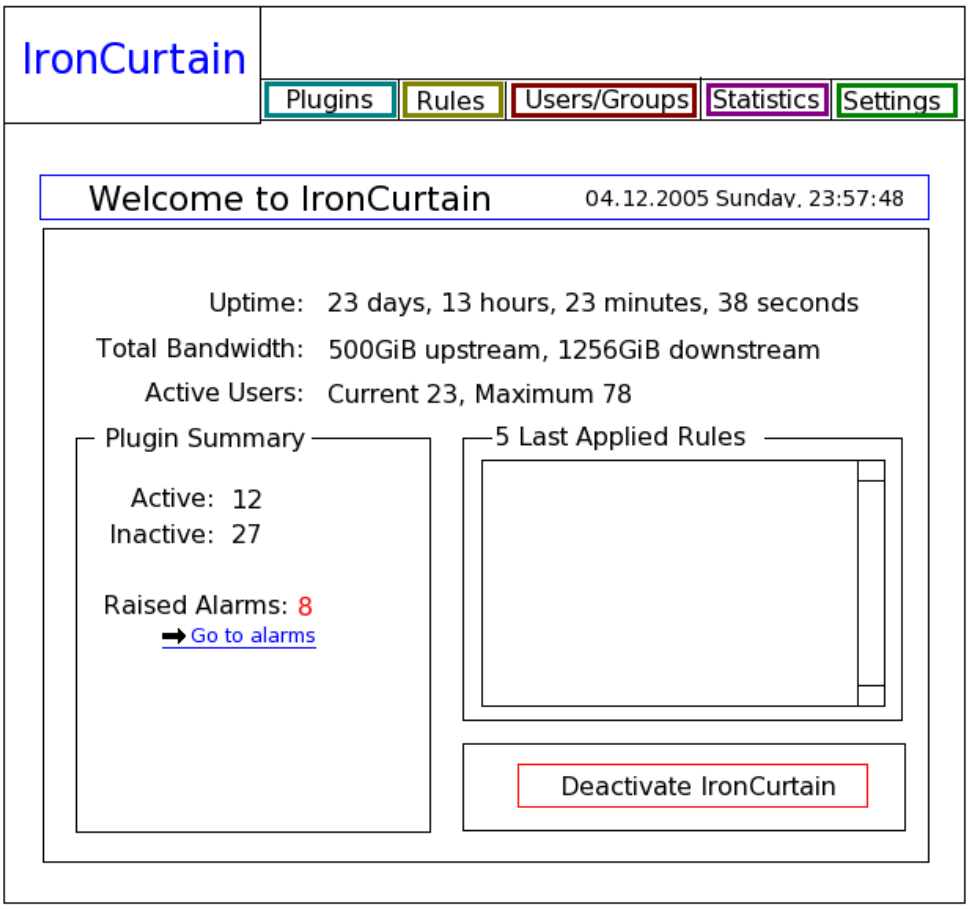
Prompt and Response Guidelines

1. Use upper/lower case letters (for emphasis as needed)
2. Tasks should be interruptible without loss
3. Give user a means of controlling multiple screens(or blocks in web page)
4. Give user a reasonable amount of time to respond
5. Compose in screen rather than lines
6. When long delays are inevitable, put up an indicator

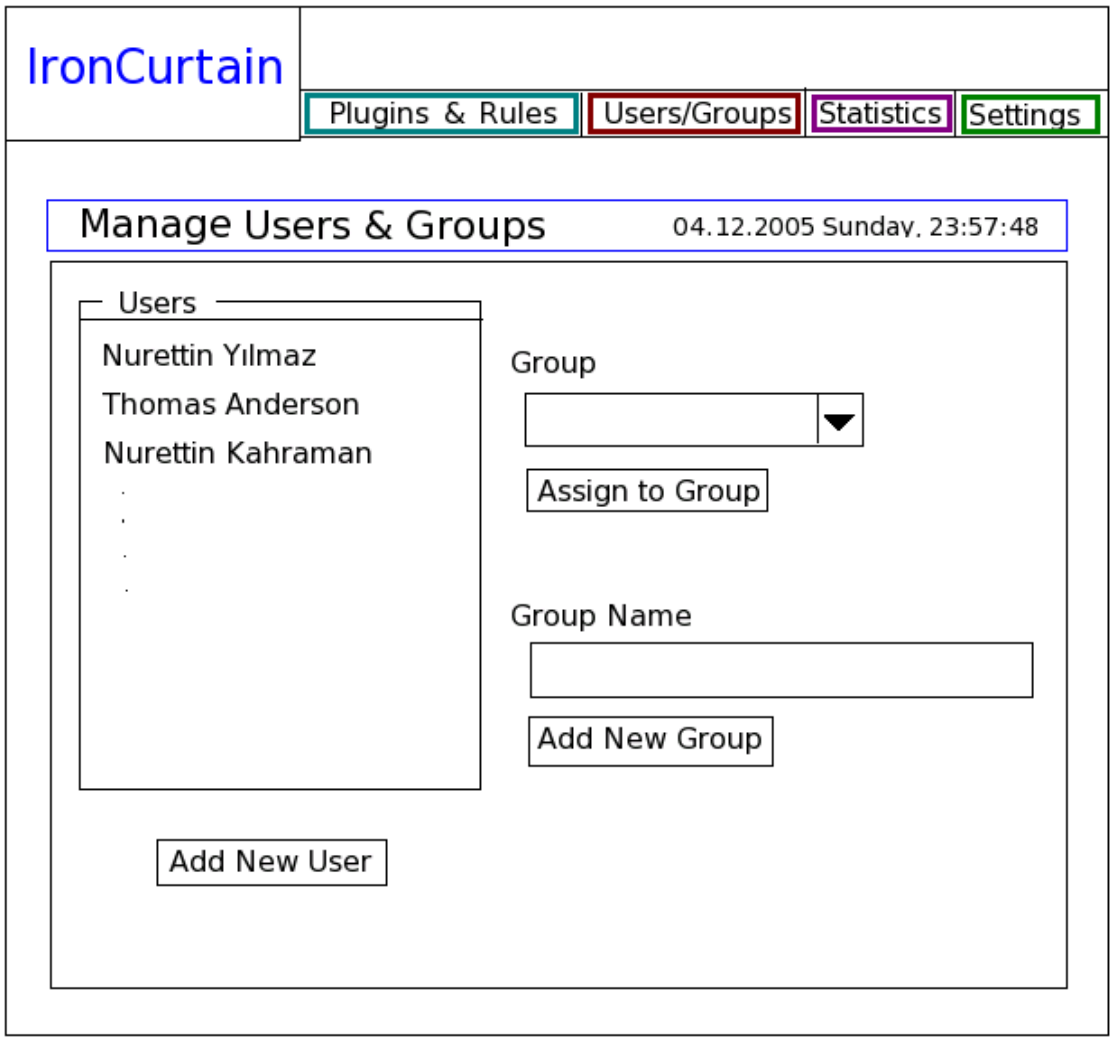
4.2 Screenshots

Our interface mockups are as follows.

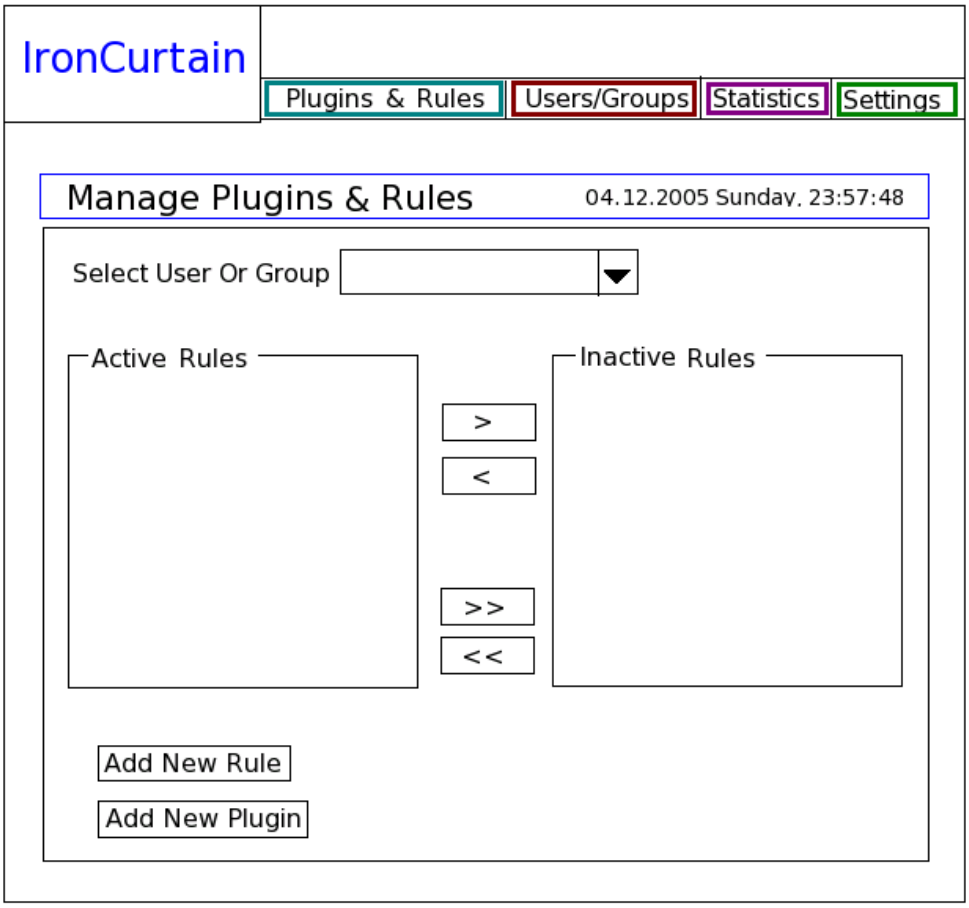
4.2.1 Summary View



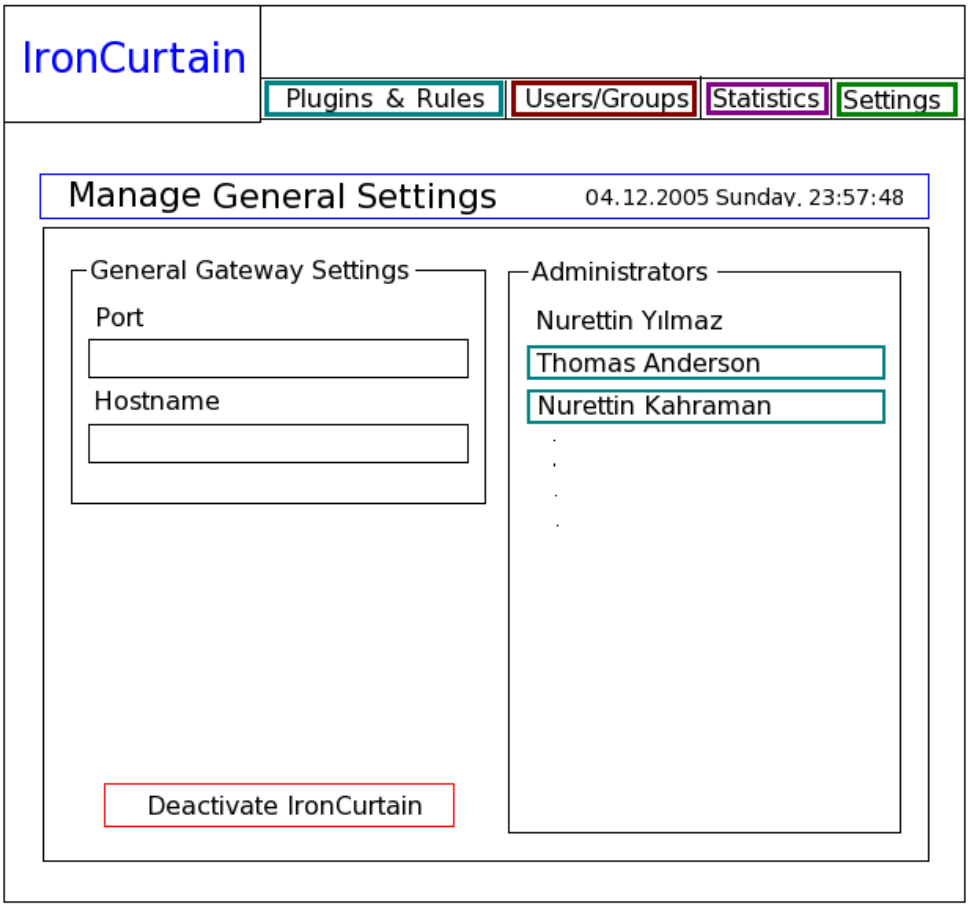
4.2.2 User and Group



4.2.3 Plug-in and Rule Management



4.2.4 Settings



5. Requirements

This section gives details about functional, non-functional and minimal software / hardware requirements.

5.1 Functional Requirements

Functional requirements can be listed as:

- **Enable or Disable IronCurtain**

In some cases, it may be necessary to suspend IronCurtain completely. There will be an option to disable the gateway actions. Then all connections will have direct access without any inspection. Anytime with the enable option, system will continue to control and secure the traffic

- **Web Content Control**

In Web pages, apart from text, there exist images, multimedia objects such as Flash, ActiveX, Java frames. These objects may contain insecure code fragments. To avoid them there will be options to filter the incoming pages. Ads, animated GIFs, Flash and ActiveX objects can be disabled by authorized user. Incoming or outgoing web content will be analyzed based on HTTP request and response headers, the HTML page, images, references to other sites. Basically IronCurtain will track and parse all the communication between the server and the client. Then it will expose the data to our plug-in environment.

In IronCurtain plug-in environment the proper module will perform on the parsed data, it will apply the pre-defined rules.

- **Add/Remove Users or Groups**

IronCurtain will allow adding and removing of users or groups. It will be possible to configure access and accounting per-user and per-group.

- **Control User Accounts**

Every user in this system will have an account saved in the database. There may be alternatives for identifying users;
It may be IP address or MAC based authentication, this would be more useful for a transparent proxy. It may use an existing authentication scheme for proxy authentication, Kerberos or NTLM.

- **Secure Login**

Administrator is required to enter his/her ID and password before using our proxy server. The entered login information is checked against the stored information: if they match, the user is logged on to the system; if they do not match, the system should print an error message and again present the login screen.

- **Add/Remove Rules**

Through the web interface, the administrator will be able to add arbitrary new rules, such as modifying HTTP headers or blocking images. The administrator will also be able to remove them from the system. When there is any change in the rules, the system will take action immediately.
IronCurtain will have a plug-in based architecture. Rule plug-ins, which will be written in our programming language or a simple script language, will define the mechanism. Rules, which will be defined over the web interface, will use plug-ins as their underlying template and will define the policy.

- **Check the bandwidth usage**

The bandwidths of the networks and Internet access are limited. So, too many unnecessary downloads or uploads of some users lead to slow connections, delays, denial of service problems for other users and programs. In order to avoid this, a proper logging

of bandwidth usage should exist in this system. For every user this logged bandwidth usage data will be displayed and there will be an option to limit the upload or download quota of a specific user. This is necessary for preventing users from exploiting the bandwidth and suffering other users on the net.

- **Check user actions**

There will be restrictions and some controls to prevent users from being distracted. In order to do this control all the user actions will be logged. Then the visited web sites and durations on these sites will be displayed. And it will be possible for authorized user to block any user's access to some sites. There will be also black lists. The sites in these lists will be unreachable by any user. The administrator will be able to change this list manually by adding or removing sites.

- **Check statistical data**

All of the user actions, or a part of it if the administrator configures as such, will be logged extensively.
Statistical data will be generated from the logs of users' actions, and it will be updated in real-time. A table will be displayed showing the web sites visited, a counter and a duration field. This information can be used for a general view of the Internet usage. The general problems such as bandwidth bottleneck can be detected. Moreover, it can provide critical information in case of hostile actions, viruses and worms. The links that nearly all the computers connect and a counter with a high value for that link may indicate a worm. It will be possible to take action by adding that link to black list, before its unwanted consequences. Statistical data for connections will also be displayed as diagrams.

5.2 Non-functional Requirements

Non-functional requirements can be listed as:

- **Scalability**

Since Web Access Control is an important issue for large companies too, our software must scale equally well to any number of users. Even, in large corporal networks, users should face minimal or no latency in their regular Web usage.

- **Maintenance**

Maintainability is required to be able to solve issues encountered before and after enabling the system. Later on it may be requested to add new functionality or enhance the system. Therefore, the source code must be maintainable.

- **User Interface**

The user interface will mainly be a website for the administrator to configure the system. The interface must be robust. Since, administrators will want to have a fine-grained control over their network, the web interface must enable the administrator to configure both macro details like the size of the thread pool or micro details like blocking a specific user from accessing a specific web site.

- **Security**

There are two aspects of security. First, the internal network which clients connect to our software may not be secure. Therefore, we must provide methods (such as, NT Lan Manager (http://en.wikipedia.org/wiki/NTLM) or Kerberos (http://web.mit.edu/kerberos/www/) to enable secure user authentication over insecure networks. Secondly, the Internet is, unfortunately, filled with people with malicious intents. IronCurtain must be safe from malicious code attacks like buffer overruns(which, we think that coding in a high-level language like Python will help enormously) itself, and also it must provide methods that allow administrators to define rules to block such known attacks.

Also the people that are being protected and controlled by the IronCurtain may try to bypass IronCurtain. The system must be resistant to such attempts, and should log and

alert the administrator also. To prevent bypassing IronCurtain, there are things an administrator must do; like blocking all outgoing HTTP connection attempts, an allowing only the IronCurtain's access to outside.

- **Platform Independent Use**

IronCurtain should be operational and functional across different platforms like UNIX, Windows, etc.

- **Reliability**

The system should be as bug free as possible. All sub components should work asynchronously, so that any delay caused by one of the components should not block other components' work flow.

5.3 Minimal Hardware Requirements

Minimal hardware requirements for our project are:
A PC with the following configuration will be needed:

Development

A PC with the following configuration will be needed:

- Intel Pentium IV 1 GHz
- 512 MB DDR RAM
- 40GB Hard Disk Space
- Internet Connection

End User

- Intel Pentium IV 2 GHz
- 1GB DDR RAM
- 100MB Hard Disk Space
- Local or Wide Area Network

5.4 Minimal Software Requirements

Software requirements for the project are divided into categories:

Development

- Recent Linux Distribution such as Ubuntu 5.10, or Windows XP or newer
- Full Installation of Python 2.4
- SPE Python IDE or Eclipse Installation with PyDev Plug-in
- Navicat (MySQL Manager)
- LaTeX Suite

End user

- Recent Linux Distribution such as Ubuntu 5.10, or Windows XP or newer
- Full Installation of Python 2.4
- MySQL Client
- Web Browser (Mozilla Firefox, Microsoft Internet Explorer)

6. Project Schedule

Project scheduling is one of the most important subtasks of project management. For this project, scheduling activities and tasks are given below.

6.1 Project Task Set

Framework Activities

- Customer Communication √
- Initial Design √
- Design
- Programming
- Testing

Task Set

- Requirements specification √
- Learning the languages and tools √

- Prototype construction
- Database construction
- Plug-in & Rule Construction
- Construction of Logging
- Interface construction
- Adaptation of modules
- Testing

List of deliverables

Documentation

- Project Proposal √
- Analysis Report √
- Initial Design Report √
- Design Report

Functional Decomposition of these tasks

Requirements Specification

Interviews with some software companies √

Internet Search √

Learning the languages and tools

Determining the proper language √

Determining the tutorials and manuals √

Studying the tutorials √

Prototype Construction

Database Construction

Initial Database Design

Initial Database Implementation

Complete Database Design

Complete Database Implementation

Plug-in & Rule Construction

- Plug-in & Rule design
- Plug-in & Rule initial implementation
- Plug-in & Rule complete implementation

Logging

- Logging and reporting

Interface Construction

- Administrator web interface

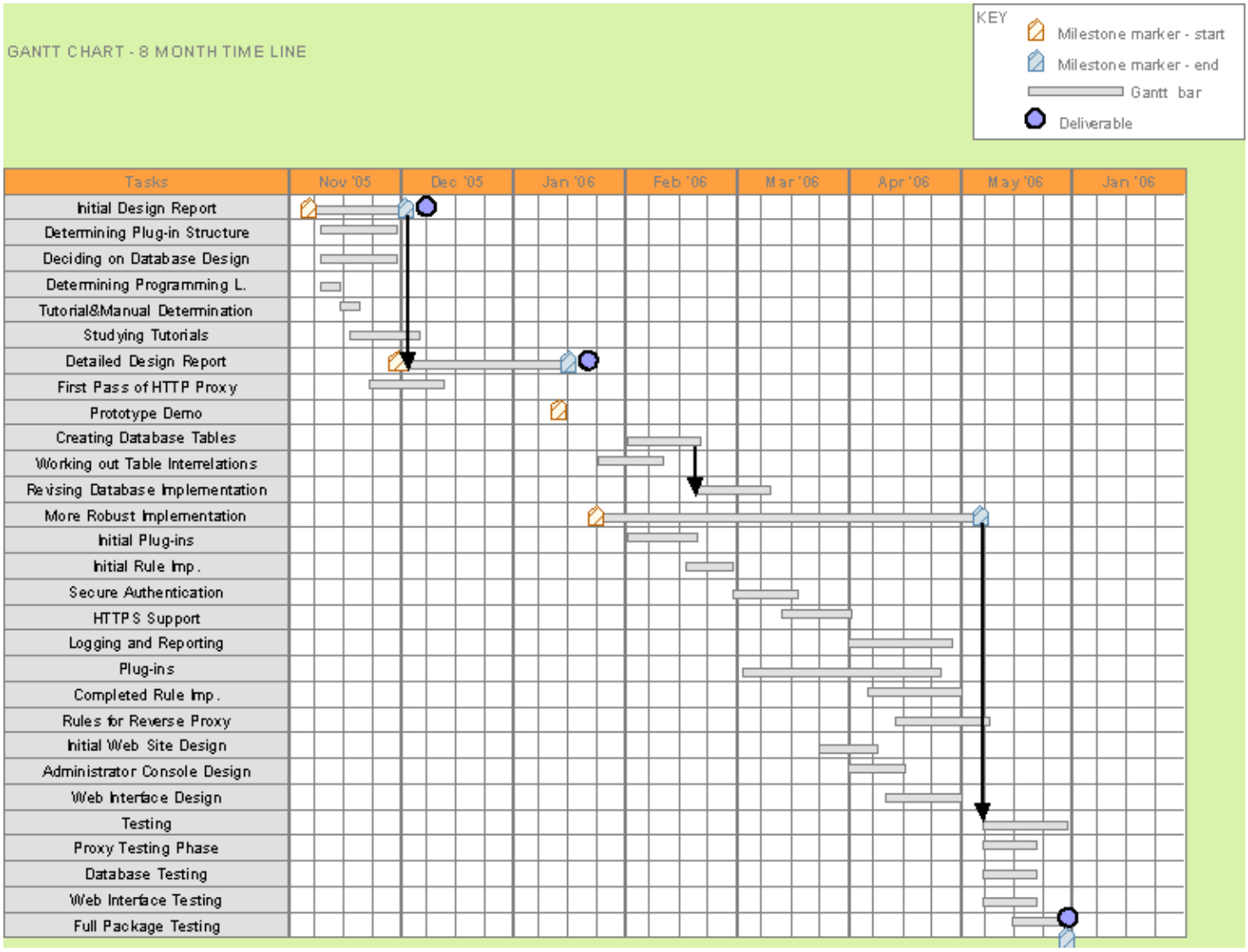
Adaptation of modules

- Database & rules
- Database & logging
- Full connection among all

Testing

- Web Interface testing
- Proxy testing
- Database testing
- Full package testing

6.2 Gannt Chart



7. Testing

Since this is initial design we will give just basic structure of testing strategies for our project.

7.1 Unit Testing

We are going to test our login

- Plug-in and module: Processing rules synchronously or threaded will be problematic. We are going to point at this, while testing.
- Database: We will use database mostly for accessing user information and reaching statistical information. Database structure is very important for our project since fast responding system is necessary for users to access internet. Testing for performance issues will be mostly related to the database.
- Web interface: Security issues will be important since web interface is just for the administrator and securing his/her information will be important.
- Log module: Statistical data will be gathered by this module. Database testing will be an important issue because we will record information on the database and also, get statistical information from the database.

7.2 Integration Testing

We are going to test the system as whole after combining different modules in IronCurtain. We will ensure synchronized processing of sub-modules is correct.

7.3 Higher Order Testing

Higher order testing is going to take place when complete integration is maintained on IronCurtain.

8. Issues to be Addressed in Final Design Report

There are some issues about IronCurtain design that is left to final design report. These need some extra technical research and elaboration on the topic. We may list them as follows.

- **Authentication:** We will research about NTLM and Kerberos.
- **Default Plug-ins:** We will detail the list of the plug-ins that we will ship by default, and their functions. These may include, among others, bandwidth management and cookie management.
- **Robustness of Plug-ins:** Currently, one rule who fails to return can lock-up the system. In addition to this, the rules work one by one which may introduce an undesirable latency. Also, the rules act on parsed complete requests or responses, there should be an option to allow incrementally decided actions. That is, as soon as a packet that forms a part of an HTTP request/response is received, it will be passed to plug-ins for processing. We plan to completely solve these issues in the final design report.