

University of Southampton  
Faculty of Engineering and Applied Science  
Department of Electronics and Computer Science

A project progress report submitted for the award of  
MEng Software Engineering

Project supervisor: Dr. Ulrich Ultes-Nitsche  
Second examiner: Dr. Les Carr

Network Management using  
Abstracted XML Specifications

by Simon Chudley

Jan 11, 2002

## **Abstract**

Establishing and managing network infrastructures between distributed nodes can be time consuming and risk prone. Such a statement is often re-enforced by the incorporation of new technologies, the merging of existing systems and the rotation of subsystems. However, the functional requirements of these enterprises remain constant, even though implementation specifics may differ greatly.

The driving force behind this project is to investigate the addition of an extra level of abstraction between the operation and implementation of such nodes within distributed environments. Such an abstraction has the advantage of being sufficiently withdrawn from specific node level details to allow subsystems to be interchanged, whilst maintaining the required behaviour. Utilising these high level descriptions, being specified in a generic syntax, a scope for simulation is introduced. Automated analysis could attempt to model the interactions between nodes and subsystems, leading onto validation of security, performance and scalability.

With the current architecture, abstracted XML descriptions of a firewall service can be created and manipulated. The service is specified using a sequence of firewall constructs, each containing a set of firewall rules. A translation method is in place to convert this abstracted description into a set of firewall rules, directly applicable to a firewall implementation commonly used under FreeBSD.

## Contents

1. Project Details .....	4
2. Project Goals .....	4
3. Background and Report of Literature Search .....	4
4. Report on Technical Progress .....	6
4.1 Analysing Firewall Rules .....	6
4.2 Creating the Abstracted Descriptions .....	7
4.3 Defining a System Structure .....	8
4.4 Translations – Specialisation.....	11
4.5 GUI Editor.....	12
5. Plan of Remaining Work .....	13
6. References.....	14
7. Appendix.....	15
7.1 XML Schema Abstracted Description of a Firewall .....	15
7.2 External Service Construct – Firewall IPSec Tunnel .....	20
7.3 Service Translation Description for Firewall to IPFW .....	22
7.4 Sample Node Configuration with a Firewall Service .....	24

## 1. Project Details

<b>Name</b>	Simon Chudley
<b>Email</b>	src299@ecs.soton.ac.uk
<b>Project Title</b>	Network Management using Abstracted XML Specifications
<b>1<sup>st</sup> Supervisor</b>	Dr. Ulrich Ultes-Nitsche
<b>2<sup>nd</sup> Supervisor</b>	Dr. Les Carr

## 2. Project Goals

- Provide a method of allowing users to gather and display structural details about nodes and links on their local area network.
- Allow the specification of network services and configuration for each required node on the network. The actual behaviour will be generalised so that it reflects the various different implementations and versions of that service available on different platforms.
- Store the state of the network layout and the service descriptions of each node to a series of XML files. These give an abstracted description of the system as a whole, allowing it to be restored and analysed at a later date.
- The XML node descriptions can then be parsed and converted into sets of configuration files specific to the required operating system and service types.

The main advantage of having the XML abstracted representation of the system and all nodes within it is that it separates the system level information, such as operating system and service version, from the higher level description of the behaviour of that node. This implies that nodes can be interchanged, and once the XML configuration of that node has been re-applied it will behave the same.

## 3. Background and Report of Literature Search

Modern networks consist of a variety of nodes chosen to provide network and user level services to peers. There are various solutions to publish these required services, often based on different operating systems due to performance and security issues. The system level configuration of these services is usually complex, and requires specific knowledge of both the service specification and the implementation being used.

Taking the firewall service as an example, configuration can differ between implementations, even though the required behaviour is the same. An example of which is IPFW [REF 2] and IPFilter [REF 3]. The following two configuration scripts both give the same functionality, to prevent packets from private networks coming in via their public network interface, tun0, as described in RFC 1918 [REF 4].

Example IPFilter configuration:

```
block in quick on tun0 from 192.168.0.0/16 to any
block in quick on tun0 from 172.16.0.0/12 to any
```

```
block in quick on tun0 from 10.0.0.0/8 to any
pass in all
```

Example IPFW configuration:

```
add deny all from 192.168.0.0/16 to any in recv tun0
add deny all from 172.16.0.0/12 to any in recv tun0
add deny all from 10.0.0.0/8 to any in recv tun0
add pass all from any to any
```

These two configurations differ purely on format; hence, they could be automatically generated from an abstracted description. Some example firewall configurations to emphasise this point are available for IPFW [REF 1] and IPFilter [REF 3]. In addition, comparisons of various packet filter firewall configurations can be seen on the Firewall Filter Language Compiler page [REF 5]. This project wishes to investigate creating such a conversion method that would be applicable to a wide range of network services.

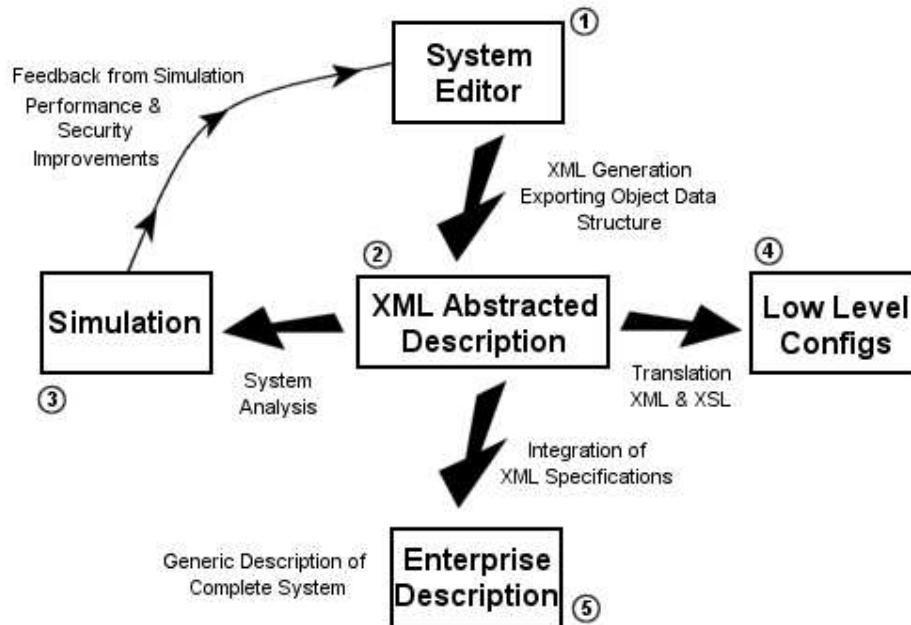


Figure 3.1: Conceptual view of the system

The proposed system is described in Figure 3.1. For each major element within the network, an abstracted XML description will be generated and stored centrally. Each network and user service running on a node will also be described using an abstracted XML description.

With these descriptions, a translation process will enable low-level configurations to be created for each specific service implementation. In addition, from stage 2, simulation and analysis could be performed on the abstracted descriptions, providing a feedback loop to the system editor, allowing for security and performance improvements.

Initial research was aimed towards developing a complete conversion for the firewall service from abstracted description through to various end level configurations. An analysis was carried out on IPFW [REF 7] and IPFilter [REF 3], two common firewall

implementation used within FreeBSD. Reference was also made to the specifications of the Internet Protocol v4 [REF 8] and Transport Control Protocol [REF 9], so that the various options supplied by each firewall implementation could be understood. Using these, a generic XML description of a firewall was generated, which acts as the storage element of this project. Another useful source of information was the firewall section in the FreeBSD handbook [REF 10].

An architecture was built to enable the XML service descriptions to be loaded, modified and saved back to XML. To enforce structural standards, XML schemas were used [REF 11]. Schemas were built to describe every major element of the system. The translation process was integrated into the system using XML Style sheets [REF 12], and hence allowed conversions to be defined external to the code base.

The Filter Compiler Language project [REF 5] has successfully implemented a conversion process from a set description to various firewall configurations. The approach taken uses the C pre-processor to execute the conversions. This allows the abstracted description to be generated using `if` statements, and variable mappings to be made. An example of this is as follows.

```
if ( in ) then {
    set protocol tcp
    if ( from host BAR and opening ) then {
        block .
    }
    if ( from foo and to host bar ) then {
        log body and block .
    }
    if ( to port 2049 ) then {
        log and block .
    }
    pass .
}
```

However, on closer examination, this approach would not be applicable to a wide range of services, as it relies on the fact that the configurations are rule based.

Another product was identified called the Firewall Builder [REF 6]. This uses a similar approach to that proposed by this project, where nodes are described using XML descriptions, and a GUI editor is used to configure the firewall. However, this is still specific to a single service, but will be useful as a comparison during the development of the firewall service within this project. In addition, the architecture for this project is going to be using Java and XML, to ensure that it remains as system independent as possible.

## 4. Report on Technical Progress

### 4.1 Analysing Firewall Rules

Initially this project focused on a single service, to enable the architecture to be designed and adapted. Once a complete translation was available to convert from a high-level to a low-level configuration, the project could be expanded to include other services. The first service to be implemented was a Firewall. Such a service has a good base for

development, as it is mainly rule based and there are many lower-level implementations for testing.

A single end level firewall implementation was chosen for the initial translation development. IPFW, the standard firewall application used on FreeBSD machines, was used to reverse-engineer the abstracted description. This was achieved by analysing the documentation [REF 10] and manual pages [REF 7] for IPFW, finding the main components and generating a description of each major element.

## 4.2 Creating the Abstracted Descriptions

It was decided that each service within the system was to be described and validated using an XML Schema. Within such descriptions, all elements will be defined formally, stating the possible values that they could take and other constraints. When the XML file for a node is parsed, each service will be validated against their respective XML Schema to ensure that they can be processed correctly.

Within the description of each element is a section to describe variables mappings. These allow the definition of some variables name to be mapped to a value. For example, defining the variable `foo` to equal `bar` at the level of a node will ensure that `bar` is available within all child elements of that node by accessing the value `foo`. The variable references are replaced by the XML generator before configurations files are produced. The use and requirement of variables mappings in this project are described in the next section.

Packet filtering firewalls, such as IPFW [REF 7], use sets of rules to determine how the traffic will be filtered. The first stage in development was to get the key elements that make up a single rule declaration. There is a finite amount of values for each element, hence the abstracted XML description of a firewall rule was straightforward to construct. Following is the definition of a firewall rule within IPFW:

```
index prob match-prob action log logamount number proto from src
      to dst interface options
```

The words in bold indicate keywords in the rule, the others represent option that can be specified. Each element has a finite set of values, such as the source and destination descriptions. For these, they can be made up of an IP (Internet Protocol) address, an address mask, and a sequence of port numbers. There could also be other specific keywords in the address declarations that have special meanings, such as `me` and `any`. Following is part of the XML Schema (XSD file) for a firewall, showing the definition of the source address of a packet.

```
<xsd:element name="src">
  <xsd:annotation>
    <xsd:documentation>The source of the
      packet</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:attribute name="type" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="any" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>
```

```
        <xsd:enumeration value="me" />
        <xsd:enumeration value="ip" />
    </xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="mask" type="xsd:string" />
<xsd:attribute name="address" type="xsd:string" />
<xsd:attribute name="negate">
    <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="true" />
            <xsd:enumeration value="false" />
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="ports" type="xsd:string" />
</xsd:complexType>
</xsd:element>
```

The XML Schema definition of an abstracted firewall is included in the appendix of this document, Section 7.1.

### 4.3 Defining a System Structure

One primary aim of this project was to provide an architecture that could generically be applied over a wide service base. It should be able to transparently support the addition of new translations from the abstracted description to implementation level configurations, and expand easily to cover new services.

The most versatile approach to use is a fully object oriented design. A major aspect of the entities' responsibilities will be to parse and generate the XML to represent them. The idea is to maintain maximum encapsulation within the service objects, so that new services can be added by the alteration of the minimum amount of external objects as possible.

During the early development of this project, it was established that commonly there is a large amount of data shared between services on nodes. For example, many services on a node would want to know that node's IP address and network interface names. Therefore, to allow for such information to be shared the concept of variable mapping was introduced. This infrastructure of variable mappings was extended throughout the entire architecture, as it aids in the overall management of the XML descriptions.

At each level within the architecture, such variable mappings can be defined. Once defined, they become available to all components lower down in the variable hierarchy, but can be redefined at these lower levels if needed.

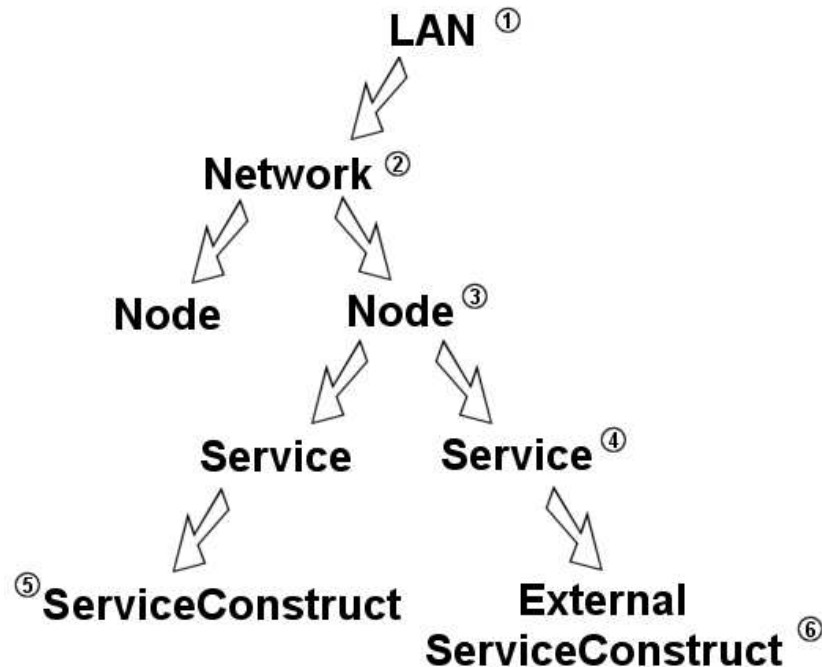


Figure 4.3.1: Variable mappings within the system

In the above diagram, Figure 4.3.1, variables can be defined at any of the levels. For example, variables defined in the LAN description can be accessible within the ServiceConstructs. At the LAN level, level 1, variables will be defined that apply to the whole system. This is currently the root element, but further expansion to the system may incorporate other elements higher than this. At this level definition of certain server IP addresses or names could be defined, such as the DNS server for that network. At level 2, the Network, typically the main mappings that will be defined are the network number and subnet mask. This may be 192.168.2.0 as the network number and 255.255.255.0 as the subnet for example.

At level 3, mappings that are unique to that Node are made. Commonly these will be such as the IP address of that node, and the network interfaces that are in use. These mappings will be available to all services running on that Node, and act as a good way to share configuration data between them. At level 4 Service level mappings are defined, a good example is with the firewall description where you could define the ports that will allow incoming connections to be made on. The final two levels are 5 and 6, these are within the ServiceConstructs themselves. These will only be available within the scope of those constructs, and are typically used mainly in the definition of ExternalServiceConstructs. Service constructs add another level of abstraction to the system and effectively allow a grouping of common service functionality together into a single construct, which can be treated as a single entity.

One major advantage of these constructs is that they can be defined externally, by registering them with a central repository. When a service wants to include the functionality provided by that construct, it includes a reference to it within the XML specification. As with all constructs, variable mappings are used to specify the behavior

of that segment of rules. This allows multiple references to the same external construct using different variables within a single service description. Following is an example of importing the functionality contained within the external construct 'Firewall::ExternalConstruct:IPSec Tunnel'. Variable mappings have been defined within the description, and these will only be available in the scope of the external service construct. This allows configuration information to be passed down to the other level. The actual external rule definition of the functionality imported in the following example is included in the appendix, Section 7.2.

```
<FirewallConstruct name="Tunnel" description="My Tunnel">
  <Mappings>
    <Variable name="remTunPubIP" value="217.1.2.3"/>
    <Variable name="remSubnet" value="192.168.0.0"/>
    <Variable name="remSubnetMask"
      value="255.255.255.255"/>
  </Mappings>
  <ExternalConstruct
    name="Firewall::ExternalConstruct:IPSec Tunnel"/>
</FirewallConstruct>
```

The reason for providing this functionality is that it allows the construction of abstracted service descriptions at a higher level. For example, users can specify that they want to include the functionality to create a tunnel between two nodes within the firewall rules. They need not be concerned with the rules needed to generate such functionality, as these are stored within the construct repository. However, when the end level configurations are generated, rules will be substituted in from the external sources and variable mappings made before the data is generated. In addition, as new constructs can be added easily it provides a useful library of specific service functionality.

The following diagram shows the relationships between the major elements within the system. Each element is responsible for its own XML parsing and generation. Such behaviour is inherited from the `NetworkElement` class, which is the parent to most entities within the system. This defines some basic operations performed to parse and generate XML data. This approach ensures that all entity functionality is encapsulated within a single object, hence allowing other objects within the system to reference it generically.

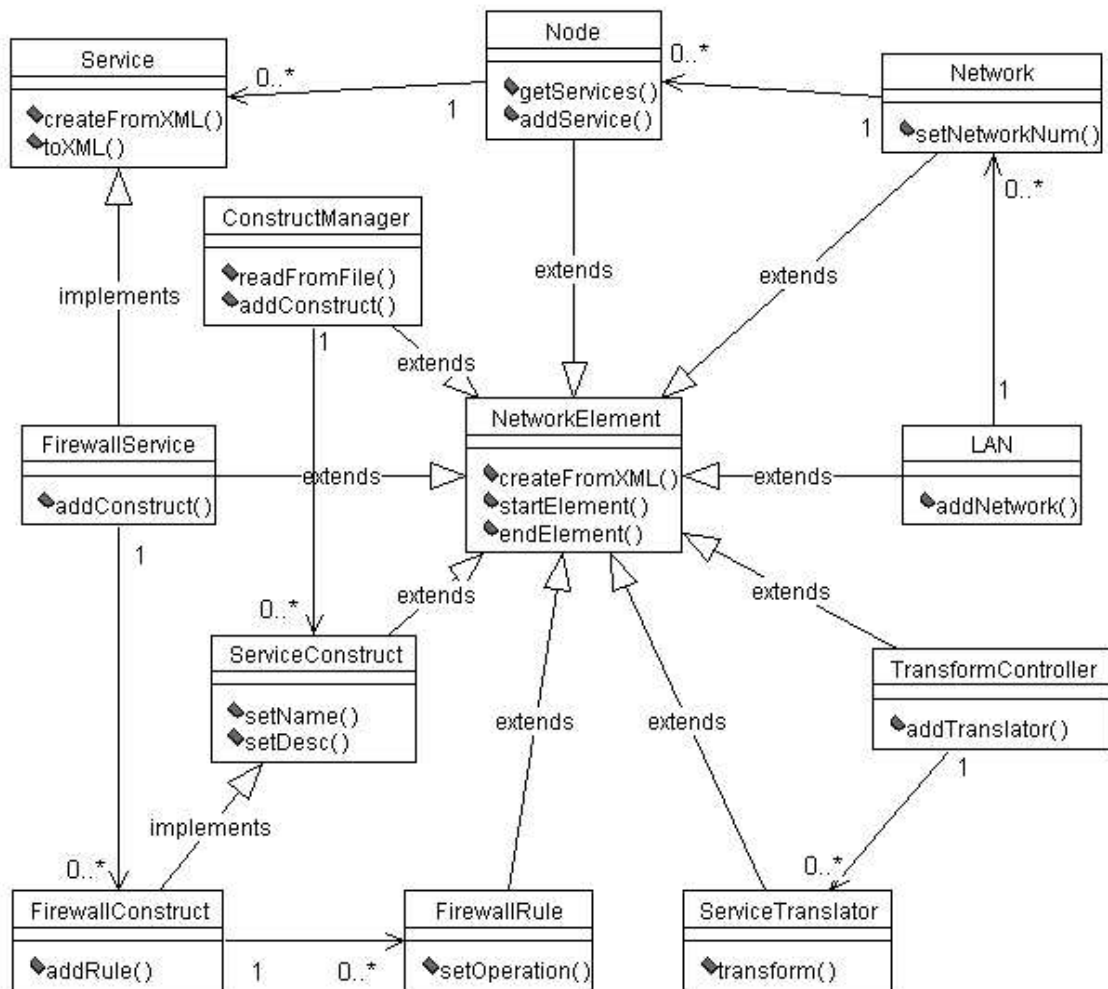


Figure 4.3.2: UML class diagram showing object relationships

#### 4.4 Translations – Specialisation

The process of converting the abstracted service descriptions to configuration files that can be applied is called translation. The translations describes how to convert on a rule-by-rule basis to the implementation level, and these can then be directly loaded into whatever service application is chosen.

Each translation process is controlled by a single `TranslationManager`, and translations can be obtained and invoked to generate the configuration files. A translation is described using an XML file and an XSL file [REF 12]. The XML file acts as a wrapper, providing information such as limitations of the translation, description and other functionality that cannot otherwise be represented in the XSL file. The XSL file (an XML style sheet) is used to achieve the translation. It provides mappings from constructs in the abstracted XML service description to elements within the final implementation's language. Following is an extract from the translation for a firewall into IPFW configuration rules. The details for converting a source address are given, and each element within that description has a possible translation to its IPFW equivalent.

```
<xsl:template match="src">
  <xsl:choose>
    <xsl:when test="@type='ip'">
      <xsl:value-of select="@address"/>/
      <xsl:value-of select="@mask"/>
      <xsl:if test="self::node()[@ports]">
        <xsl:text> </xsl:text>
        <xsl:value-of select="@ports"/>
      </xsl:if>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="@type"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

One major advantage of this choice of structure is that all translation functionality is contained within external XML and XSL files. No alteration needs to be made to the object structure or code as all translations are referenced using a generic interface within the system. This implies that for a user to create a translation to a new service implementation they only have to create some external files and register them with the `TranslationManager`.

The whole XSL translation file for converting the abstracted XML description to IPFW configuration files is given in the appendix to this document, Section 7.3.

#### 4.5 GUI Editor

The GUI editor is the element within the system that allows each service to be configured and implementation level configuration files to be generated. Each element within the system, as shown in Figure 4.3.2, is able to display an editing window to allow its state to be configured. The following screen shot, Figure 4.5.1 shows a node with the firewall configuration window open. It allows all variable mappings at that level to be changed and the service specific details to be configured. Selecting a service construct and using the `Edit` button will call the `displayEditor()` method on that particular construct. Using this process of encapsulating all configuration functionality within each individual object ensures that maintenance and expansion of the project is easier. If a user wishes to add support for a new service to the application, they can simply define their new classes and add the GUI configuration functionality to those classes directly. The higher-level node in this situation only refers to them by the `Service` interface, and hence does not need to be concerned with which service it is.

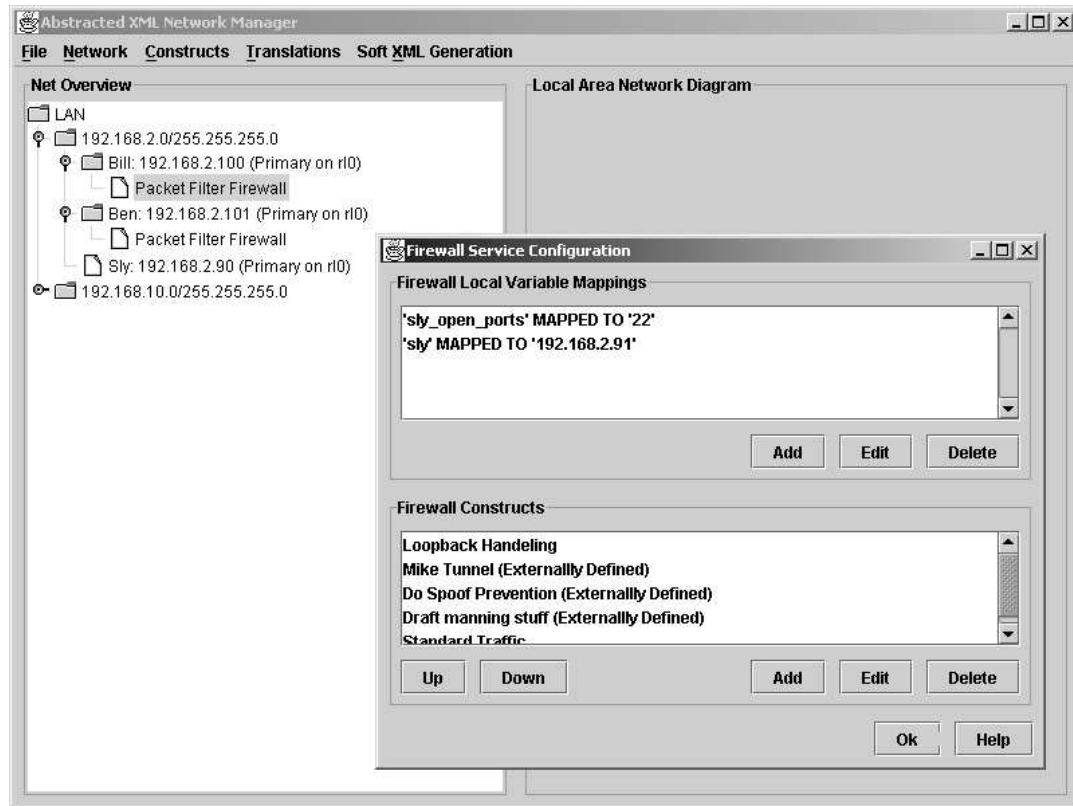


Figure 4.5.1: Screen shot of the firewall service configuration window

An XML description of a sample node is given in the appendix of this document, in Section 7.4. It shows the node configuration along with a firewall service. The service uses both local and externally defined service constructs.

## 5. Plan of Remaining Work

Currently, the system structure is in place to allow services to be loaded, modified and saved back to XML, and translations to be executed. As the initial development was undertaken using the firewall service, a generic description of a firewall can be loaded from XML, modified, saved back to XML and translated to IPFW configuration files.

External service constructs and variable mappings are functional, and some external constructs for the firewall service have been created. When XML files are generated from the application, variable mapping replacement can be toggled, allowing for the inclusion of external constructs rules into the actual node description. This would allow an XML node description to be loaded and used within a system that does not have certain external declarations in its own library.

The next stage in development is the addition of new services and translations. Support for other firewall implementations, such as IPFilter, will be added, and this will ensure that the abstracted description of a firewall can cover many different implementations. New services will be added to the system, such as DHCP, DNS and NAT, which will extend the scope of this project, and ensure that the architecture does easily cover

multiple services. The GUI editor also needs to be expanded to allow the configuration of all elements within the system.

If the above has been achieved then the project scope could be expanded to investigate into the possibility of network and service simulation. As XML descriptions of the whole network will be available, it should be possible to calculate how the overall system will behave. Such simulations may test to see what network packet data could get through a firewall service for example. The major advantage of a simulation component to the system is that it would allow for improvements in performance and security of the whole network, hence providing a feedback loop of new abstracted configurations.

## 6. References

- [Ref 1]:** IPFW example configuration  
<http://www.acme.com/firewall.html>
  
- [Ref 2]:** IPFW configuration details  
[http://www.onlamp.com/pub/a/bsd/2001/06/01/FreeBSD\\_Basics.html](http://www.onlamp.com/pub/a/bsd/2001/06/01/FreeBSD_Basics.html)
  
- [Ref 3]:** IPFilter Howto  
<http://www.obfuscation.org/ipf/ipf-howto.html>
  
- [Ref 4]:** Address allocation for private networks (RFC 1918)  
<http://www.faqs.org/rfcs/rfc1918.html>
  
- [Ref 5]:** Filter language compiler  
<http://coombs.anu.edu.au/~avalon/flc.html>
  
- [Ref 6]:** Firewall Builder  
<http://www.fwbuilder.org/>
  
- [Ref 7]:** IPFW Manual pages  
<http://www.gsp.com/cgi-bin/man.cgi?section=8&topic=ipfw>
  
- [Ref 8]:** Internet Protocol v4 (RFC 791)  
<http://www.faqs.org/rfcs/rfc791.html>
  
- [Ref 9]:** Transport Control Protocol (RFC 793)  
<http://www.faqs.org/rfcs/rfc793.html>
  
- [Ref 10]:** FreeBSD Handbook – Firewall Section  
[http://www.freebsd.org/doc/en\\_US.ISO8859-1/books/handbook/firewalls.html](http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls.html)
  
- [Ref 11]:** XML Schemas  
<http://www.w3.org/XML/Schema>
  
- [Ref 12]:** XML Style Sheets (XSL)  
<http://www.w3.org/Style/XSL/>

## 7. Appendix

### 7.1 XML Schema Abstracted Description of a Firewall

XML Schema to describe the abstracted behaviour of a firewall service running on a node within the system.

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema targetNamespace="XMLNetMan"
             xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
             xmlns:xmlnetman="XMLNetMan"
             elementFormDefault="qualified"
             attributeFormDefault="unqualified">

  <!-- Define the external schema definitions we need to suck in-->
  <xsd:import namespace="XMLNetMan"
             schemaLocation="ServiceConstruct.xsd"/>
  <xsd:import namespace="XMLNetMan" schemaLocation="Global.xsd"/>

  <!-- This element describes the behaviour of a firewall service-->
  <xsd:element name="Firewall">
    <xsd:annotation>
      <xsd:documentation>Describes the behaviour of a
        firewall service</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Mappings" minOccurs="0"/>
        <xsd:element ref="FirewallConstruct"
          minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <!-- A FirewallConstruct is a collection or block of related
  firewall rules, and is a ServiceConstruct. Defines common
  behaviour -->

  <xsd:element name="FirewallConstruct">
    <xsd:annotation>
      <xsd:documentation>Defines a ServiceConstruct for a
        section of firewall rules</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="Mappings" minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:choice>
          <xsd:element ref="Rule" minOccurs="0"
            maxOccurs="unbounded"/>
          <xsd:element ref="ExternalConstruct"
            minOccurs="0"/>
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string"
        use="required"/>
    </xsd:complexType>
  </xsd:element>

```

```

        <xsd:attribute name="description" type="xsd:string"
            use="optional"/>
    </xsd:complexType>
</xsd:element>

<!-- Defines a rule within the firewall script -->

<xsd:element name="Rule">
    <xsd:annotation>
        <xsd:documentation>A rule for a firewall
            command</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="action"/>
            <xsd:element ref="log" minOccurs="0"/>
            <xsd:element ref="protocol" minOccurs="0"/>
            <xsd:element ref="src" minOccurs="0"/>
            <xsd:element ref="dst" minOccurs="0"/>
            <xsd:element ref="options" minOccurs="0"/>
            <xsd:element ref="interface" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="RuleID" type="xsd:string"/>
        <xsd:attribute name="Probability" type="xsd:string"/>
        <xsd:attribute name="Desc" type="xsd:string"/>
    </xsd:complexType>
</xsd:element>

<!-- The action to take when matching this rule -->

<xsd:element name="action">
    <xsd:annotation>
        <xsd:documentation>The action to take on matching
            this value</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
        <xsd:attribute name="perform" use="required">
            <xsd:simpleType>
                <xsd:restriction base="xsd:NMTOKEN">
                    <xsd:enumeration value="pass"/>
                    <xsd:enumeration value="deny"/>
                    <xsd:enumeration value="unreach"/>
                    <xsd:enumeration value="reset"/>
                    <xsd:enumeration value="count"/>
                    <xsd:enumeration
                        value="check-state"/>
                    <xsd:enumeration value="divert"/>
                    <xsd:enumeration value="tee"/>
                    <xsd:enumeration value="fwd"/>
                    <xsd:enumeration value="pipe"/>
                    <xsd:enumeration value="queue"/>
                    <xsd:enumeration value="skipto"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="code">
            <xsd:simpleType>
                <xsd:restriction base="xsd:NMTOKEN">
                    <xsd:enumeration value="net"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
</xsd:element>

```

```

        <xsd:enumeration value="host" />
        <xsd:enumeration value="protocol" />
        <xsd:enumeration value="port" />
        <xsd:enumeration value="needfrag" />
        <xsd:enumeration value="srcfail" />
        <xsd:enumeration
            value="net-unknown" />
        <xsd:enumeration
            value="host-unknown" />
        <xsd:enumeration value="isolated" />
        <xsd:enumeration
            value="net-prohib" />
        <xsd:enumeration
            value="host-prohib" />
        <xsd:enumeration value="tosnet" />
        <xsd:enumeration value="toshost" />
        <xsd:enumeration value="filter-
            prohib" />
        <xsd:enumeration value="host-
            precedence" />
        <xsd:enumeration value="precedence-
            cutoff" />
    </xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="port" type="xsd:string" />
<xsd:attribute name="ipaddr" type="xsd:string" />
<xsd:attribute name="pipe_num" type="xsd:string" />
<xsd:attribute name="queue_num" type="xsd:string" />
<xsd:attribute name="number" type="xsd:string" />
</xsd:complexType>
</xsd:element>

<!-- Logging on rule match -->

<xsd:element name="log">
    <xsd:annotation>
        <xsd:documentation>Whether to log this rule
            match</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
        <xsd:attribute name="value" use="required">
            <xsd:simpleType>
                <xsd:restriction base="xsd:NMTOKEN">
                    <xsd:enumeration value="true" />
                    <xsd:enumeration value="false" />
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="logamount" type="xsd:string" />
    </xsd:complexType>
</xsd:element>

<!-- The protocol definition -->

<xsd:element name="protocol">
    <xsd:annotation>
        <xsd:documentation>The protocol to
            match</xsd:documentation>

```

```

</xsd:annotation>
<xsd:complexType>
  <xsd:attribute name="type" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="all"/>
        <xsd:enumeration value="ip"/>
        <xsd:enumeration value="other"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
</xsd:element>

```

```

<!-- The source address of the packet -->

```

```

<xsd:element name="src">
  <xsd:annotation>
    <xsd:documentation>The source of the
      packet</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:attribute name="type" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="any"/>
          <xsd:enumeration value="me"/>
          <xsd:enumeration value="ip"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="mask" type="xsd:string"/>
    <xsd:attribute name="address" type="xsd:string"/>
    <xsd:attribute name="negate">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="true"/>
          <xsd:enumeration value="false"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="ports" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>

```

```

<!-- The destination address of the packet -->

```

```

<xsd:element name="dst">
  <xsd:annotation>
    <xsd:documentation>The destination of the
      packet</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:attribute name="type" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="any"/>
          <xsd:enumeration value="me"/>

```

```

                <xsd:enumeration value="ip" />
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="mask" type="xsd:string" />
    <xsd:attribute name="address" type="xsd:string" />
    <xsd:attribute name="negate">
        <xsd:simpleType>
            <xsd:restriction base="xsd:NMTOKEN">
                <xsd:enumeration value="true" />
                <xsd:enumeration value="false" />
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="ports" type="xsd:string" />
</xsd:complexType>
</xsd:element>

<!-- Interface specifications -->

<xsd:element name="interface">
    <xsd:annotation>
        <xsd:documentation>Defines the interface
            specifications of the packet</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
        <xsd:attribute name="direction">
            <xsd:simpleType>
                <xsd:restriction base="xsd:NMTOKEN">
                    <xsd:enumeration value="in" />
                    <xsd:enumeration value="out" />
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="via" type="xsd:string" />
        <xsd:attribute name="xmit" type="xsd:string" />
        <xsd:attribute name="recv" type="xsd:string" />
    </xsd:complexType>
</xsd:element>

<!-- Packet options -->

<xsd:element name="options">
    <xsd:annotation>
        <xsd:documentation>Various tcp option
            flags</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
        <xsd:attribute name="keep-state" type="xsd:string" />
        <xsd:attribute name="bridged">
            <xsd:simpleType>
                <xsd:restriction base="xsd:NMTOKEN">
                    <xsd:enumeration value="yes" />
                    <xsd:enumeration value="no" />
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="frag">
            <xsd:simpleType>

```

```

        <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="yes"/>
            <xsd:enumeration value="no"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="ipoptions" type="xsd:string"/>
<xsd:attribute name="tcpoptions" type="xsd:string"/>
<xsd:attribute name="established">
    <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="yes"/>
            <xsd:enumeration value="no"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="setup">
    <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="yes"/>
            <xsd:enumeration value="no"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="tcpflags" type="xsd:string"/>
<xsd:attribute name="icmptypes" type="xsd:string"/>
<xsd:attribute name="uid" type="xsd:string"/>
<xsd:attribute name="gid" type="xsd:string"/>
</xsd:complexType>
</xsd:element>

</xsd:schema>

```

## 7.2 External Service Construct – Firewall IPSec Tunnel

Each firewall construct is a set of rules, with various external variables that must be defined when they are included. To use these, you must define the variables somewhere in the variable namespace. Hence, if a construct defines that `$ut_if` needs to be defined, you will have to define it somewhere, either within the parent Service or Node.

The actual description allows tunnel access between two trusted sites using IPSec, with ESP encryption and IPv4 in IPv4. Also allows IKE on UDP port 500 between the two sites. Need to define `$remTunPubIP` as the public IP of the remote end of the tunnel, `$remSubnet` as the remote trusted subnet, `$remSubnetMask` to be their netmask, `$trusted_net` to be the trusted network, `$trusted_mask` to be the trusted netmask and `$untrusted_if` to be the untrusted network interface.

```

<ServiceConstructs xmlns="XMLNetMan"
    xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-
instance"
    xsi:schemaLocation="XMLNetMan ServiceConstruct.xsd">

    <FirewallConstruct
        name="Firewall::ExternalConstruct:IPSec Tunnel"
        description="Allows IPSec tunnel traffic between two given
sites">

```

```
<Rule Desc="Allow ESP Protocol 4 IPv4 encapsulated in IPv4
out">
  <action perform="pass"/>
  <protocol name="ipencap" type="other"/>
  <src type="me"/>
  <dst type="ip" mask="255.255.255.255"
    address="$remTunPubIP"/>
  <interface direction="out" xmit="$untrusted_if"/>
</Rule>
<Rule Desc="Allow ESP protocol 50 traffic out">
  <action perform="pass"/>
  <protocol name="esp" type="other"/>
  <src type="me"/>
  <dst type="ip" mask="255.255.255.255"
    address="$remTunPubIP"/>
  <interface direction="out" xmit="$untrusted_if"/>
</Rule>
<Rule Desc="Allow IKE out">
  <action perform="pass"/>
  <protocol name="udp" type="other"/>
  <src type="me"/>
  <dst type="ip" mask="255.255.255.255"
    address="$remTunPubIP" ports="500"/>
  <interface direction="out" xmit="$untrusted_if"/>
</Rule>
<Rule Desc=
  "Allow ESP Protocol 4 IPv4 encapsulated in IPv4 in">
  <action perform="pass"/>
  <protocol name="ipencap" type="other"/>
  <src type="ip" mask="255.255.255.255"
    address="$remTunPubIP"/>
  <dst type="me"/>
  <interface direction="in" rcv="$untrusted_if"/>
</Rule>
<Rule Desc="Allow ESP protocol 50 traffic in">
  <action perform="pass"/>
  <protocol name="esp" type="other"/>
  <src type="ip" mask="255.255.255.255"
    address="$remTunPubIP"/>
  <dst type="me"/>
  <interface direction="in" rcv="$untrusted_if"/>
</Rule>
<Rule Desc="Allow IKE in">
  <action perform="pass"/>
  <protocol name="udp" type="other"/>
  <src type="ip" mask="255.255.255.255"
    address="$remTunPubIP" ports="500"/>
  <dst type="me"/>
  <interface direction="in" rcv="$untrusted_if"/>
</Rule>
<Rule Desc=
  "Allow trusted network to talk to remote subnet">
  <action perform="pass"/>
  <protocol type="all"/>
  <src address="$trusted_net" mask="$trusted_mask"
    type="ip"/>
  <dst mask="$remSubnetMask" address="$remSubnet"
    type="ip"/>
</Rule>
<Rule Desc=
```

```

        "Allow remote subnet to talk to trusted network">
        <action perform="pass"/>
        <protocol type="all"/>
        <src address="$remSubnet" mask="$remSubnetMask"
            type="ip"/>
        <dst mask="$trusted_net" address="$trusted_mask"
            type="ip"/>
    </Rule>
</FirewallConstruct>

</ServiceConstructs>

```

### 7.3 Service Translation Description for Firewall to IPFW

XST document to describe the translation from a generic firewall implementation into IPFW rules.

```

<?xml version="1.0"?>

<xsl:stylesheet xmlns:xmlnetman="XMLNetMan"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<!-- PROCESS EACH FIREWALL CONSTRUCT SECTION -->
<xsl:template match="FirewallConstruct">
#####
# <xsl:value-of select="@description"/>
  <xsl:text>&#xa;</xsl:text>
  <xsl:apply-templates select="Rule"/>
  <xsl:text>&#xa;</xsl:text>
</xsl:template>

<!-- CONVERT EACH FIREWALL RULE -->
<xsl:template match="Rule">
# <xsl:value-of select="@Desc"/><xsl:text> add </xsl:text>
  <xsl:if test="@RuleID != ''"><xsl:value-of
select="@RuleID"/><xsl:text> </xsl:text></xsl:if>
  <xsl:apply-templates select="action"/><xsl:text> </xsl:text>
  <xsl:apply-templates select="protocol"/>
  <xsl:if test="self::node()[src]">
    <xsl:text> from </xsl:text>
    <xsl:apply-templates select="src"/>
  </xsl:if>
  <xsl:if test="self::node()[dst]">
    <xsl:text> to </xsl:text>
    <xsl:apply-templates select="dst"/>
  </xsl:if>
  <xsl:if test="self::node()[interface]">
    <xsl:apply-templates select="interface"/>
  </xsl:if>
  <xsl:if test="self::node()[options]">
    <xsl:apply-templates select="options"/>
  </xsl:if>
  <xsl:text> </xsl:text>

```

```
<xsl:text>&#xa;</xsl:text>
</xsl:template>

<xsl:template match="action">
<xsl:value-of select="@perform"/>
</xsl:template>

<xsl:template match="protocol">
  <xsl:choose>
    <xsl:when test="@type='other'"> <xsl:value-of select="@name"/>
      </xsl:when>
    <xsl:otherwise> <xsl:value-of select="@type"/> </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="src">
  <xsl:choose>
    <xsl:when test="@type='ip'"> <xsl:value-of
      select="@address"/><xsl:value-of select="@mask"/>
    <xsl:if test="self::node()[@ports]"><xsl:text>
</xsl:text><xsl:value-of select="@ports"/></xsl:if>
    </xsl:when>
    <xsl:otherwise> <xsl:value-of select="@type"/> </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="dst">
  <xsl:choose>
    <xsl:when test="@type='ip'"> <xsl:value-of
      select="@address"/><xsl:value-of select="@mask"/>
    <xsl:if test="self::node()[@ports]"><xsl:text>
      </xsl:text><xsl:value-of select="@ports"/></xsl:if>
    </xsl:when>
    <xsl:otherwise> <xsl:value-of select="@type"/> </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="interface">
  <xsl:if test="@direction = 'in'"><xsl:text>in </xsl:text></xsl:if>
  <xsl:if test="@direction = 'out'"><xsl:text>out </xsl:text></xsl:if>
  <xsl:if test="self::node()[@via]"><xsl:text>via
</xsl:text><xsl:value-of
  select="@via"/></xsl:if>
  <xsl:if test="self::node()[@xmit]"><xsl:text>xmit
</xsl:text><xsl:value-
  of select="@xmit"/></xsl:if>
  <xsl:if test="self::node()[@recv]"><xsl:text>recv
</xsl:text><xsl:value-
  of select="@recv"/></xsl:if>
</xsl:template>

<xsl:template match="options">
  <xsl:if test="self::node()[@keep-state]"><xsl:text>keep-state
</xsl:text><xsl:value-of select="@keep-state"/></xsl:if>
```

```

    <xsl:if test="@bridged = 'yes'"><xsl:text>bridged
</xsl:text></xsl:if>
    <xsl:if test="@frag = 'yes'"><xsl:text>frag </xsl:text></xsl:if>
    <xsl:if test="self::node()[@ipoptions]"><xsl:text>ipoptions
</xsl:text><xsl:value-of select="@ipoptions" /></xsl:if>
    <xsl:if test="self::node()[@tcpoptions]"><xsl:text>tcpoptions
</xsl:text><xsl:value-of select="@tcpoptions" /></xsl:if>
    <xsl:if test="@established = 'yes'"><xsl:text>established
</xsl:text></xsl:if>
    <xsl:if test="@setup = 'yes'"><xsl:text>setup </xsl:text></xsl:if>
    <xsl:if test="self::node()[@tcpflags]"><xsl:text>tcpflags
</xsl:text><xsl:value-of select="@tcpflags" /></xsl:if>
    <xsl:if test="self::node()[@icmptypes]"><xsl:text>icmptypes
</xsl:text><xsl:value-of select="@icmptypes" /></xsl:if>
    <xsl:if test="self::node()[@uid]"><xsl:text>uid
</xsl:text><xsl:value-of
    select="@uid" /></xsl:if>
    <xsl:if test="self::node()[@gid]"><xsl:text>gid
</xsl:text><xsl:value-of
    select="@gid" /></xsl:if>
</xsl:template>
</xsl:stylesheet>

```

## 7.4 Sample Node Configuration with a Firewall Service

This file gives a sample description of a node. It has a firewall service definition using various internal and externally defined service constructs.

```

<?xml version="1.0" encoding="UTF-8"?>

<Node xmlns="XMLNetMan" xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-
instance" xsi:schemaLocation="XMLNetMan
Node.xsd">

    <Mappings>
        <Variable name="trusted_if" value="r10"/>
        <Variable name="trusted_ip" value="192.168.2.100"/>
        <Variable name="untrusted_if" value="tun0"/>
    </Mappings>

    <Firewall>

        <Mappings>
            <Variable name="sly" value="192.168.2.91"/>
            <Variable name="sly_open_ports" value="22"/>
        </Mappings>

        <!-- Defines a local construct, body collect of rules -->
        <!-- we don't explicitly define the class of construct-->
        <FirewallConstruct
            name="Loopback Handeling" description="Does
            stuff with the loopback device">
            <Rule Desc="Allow loopback" RuleID="10">
                <action perform="pass"/>
                <protocol type="all"/>
                <src type="any"/>
                <dst type="any"/>
                <interface via="lo0"/>
            </Rule>
        </FirewallConstruct>
    </Firewall>
</Node>

```

```

</Rule>
<Rule Desc=
  "Prevent spoofing of loopback" RuleID="50">
  <action perform="deny"/>
  <log logamount="0" value="true"/>
  <protocol type="all"/>
  <src type="any"/>
  <dst mask="255.0.0.0" address="127.0.0.1"
    type="ip"/>
  <interface/>
</Rule>
<Rule Desc="Prevent spoofing of internal private ip
range" RuleID="51">
  <action perform="deny"/>
  <protocol type="all"/>
  <src type="ip" negate="true" address="$net_num"
    mask="$net_mask"/>
  <dst type="any"/>
  <interface direction="in" recv="$trusted_if"/>
</Rule>
</FirewallConstruct>

<!-- We want to include the IPSec tunnel between -->
<FirewallConstruct name="Mike Tunnel"
description="Tunnel to Mike">
  <Mappings>
    <Variable name="remTunPubIP"
      value="217.1.2.3"/>
    <Variable name="remSubnet"
      value="192.168.0.0"/>
    <Variable name="remSubnetMask"
      value="255.255.255.255"/>
  </Mappings>
  <ExternalConstruct
    name="Firewall::ExternalConstruct:IPSec
    Tunnel"/>
</FirewallConstruct>

<!-- Define we want to include the spoof protection -->
<!-- Sucks in a construct from the ConstructManager -->
<FirewallConstruct name="Do Spoof Prevention"
description="">
  <ExternalConstruct
    name="Firewall::ExternalConstruct:Spoof
    Prevention"/>
</FirewallConstruct>

<!-- Body collect of rules -->
<FirewallConstruct name="Standard Traffic"
description="Standard communications to let through">

  <Mappings>
    <Variable name="psyco" value="192.168.2.93"/>
  </Mappings>

  <Rule Desc="Divert all traffic through NAT">
    <action perform="divert" port="natd"/>
    <protocol type="all"/>
    <src type="any"/>
    <dst type="any"/>
    <interface via="$untrusted_if"/>
  </Rule>

```

```
</Rule>
<Rule Desc=
  "Allow SSH connection into internal machine">
  <action perform="pass"/>
  <protocol type="other" name="tcp"/>
  <src type="any"/>
  <dst type="ip" address="$sly" mask="$net_mask"
    ports="$sly_open_ports"/>
  <options setup="yes"/>
  <interface recv="$untrusted_if"/>
</Rule>
<Rule Desc="Allow subset of icmp comming in">
  <action perform="pass"/>
  <protocol name="icmp" type="other"/>
  <src type="any"/>
  <dst type="any"/>
  <options icmptypes="3,4,11,12"/>
</Rule>
<Rule Desc=
  "Allow outgoing icmp information requests">
  <action perform="pass"/>
  <protocol name="icmp" type="other"/>
  <src type="ip" address="$net_num"
    mask="$net_mask"/>
  <dst type="any"/>
  <options icmptypes="8,13,15,17"/>
  <interface recv="$trusted_if"/>
</Rule>
<Rule Desc=
  "Allow incomming icmp information replies">
  <action perform="pass"/>
  <protocol name="icmp" type="other"/>
  <src type="any"/>
  <dst type="ip" address="$trusted_ip"
    mask="$net_mask"/>
  <options icmptypes="0,14,16,18"/>
</Rule>
<Rule Desc="Check against dynamic rule set">
  <action perform="check-state"/>
</Rule>
<Rule Desc="Allow any traffic from firewall ip to any
going via untrusted interface">
  <action perform="pass"/>
  <protocol type="ip"/>
  <src type="me"/>
  <dst type="any"/>
  <options keep-state=""/>
  <interface direction="out"
    via="$untrusted_if"/>
</Rule>
<Rule Desc=
  "Allow any traffic from local network to any
passing via internal interface">
  <action perform="pass"/>
  <protocol type="ip"/>
  <src type="ip" address="$net_num"
    mask="$net_mask"/>
  <dst type="any"/>
  <options keep-state=""/>
  <interface via="$untrusted_if"/>
</Rule>
```

```
        <Rule Desc="Deny all other traffic and log it"
              RuleID="65435">
          <action perform="deny"/>
          <log value="true" logamount="0"/>
          <protocol type="all"/>
          <src type="any"/>
          <dst type="any"/>
        </Rule>
      </FirewallConstruct>
    </Firewall>
  </Node>
```