

# Verigraph Special Project Report

Flavio Lorenzo

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Preliminary Notes</b>	<b>2</b>
2.1	Folder Structure . . . . .	2
<b>3</b>	<b>Verigraph XML Schema Documentation</b>	<b>4</b>
3.1	Differences with the previous version . . . . .	4
3.2	Verigraph XML Schema . . . . .	4
<b>4</b>	<b>Verigraph JSON Schemas Documentation</b>	<b>19</b>
4.1	Graph JSON Schema . . . . .	19
4.2	Verification Results JSON Representation . . . . .	22
<b>5</b>	<b>Rest API Description</b>	<b>24</b>
5.1	Resource Design . . . . .	24
5.2	Operation Design . . . . .	25
<b>6</b>	<b>Conclusions</b>	<b>28</b>

# 1 Introduction

Verigraph is a verification service (based on JSON) that can receive a description of a network topology and of security network function (NSF) configurations and that can analyze the given input in order to check if some reachability policies (e.g. the possibility for packets of a certain flow generated by a certain node of the topology graph to reach another node) are satisfied or not.

In this project, the main contribution to the Verigraph system has been the extension of the policy verification service. Based on the work of the netgroup, the service has been improved by allowing the user to define his own custom reachability policy and by providing a more in depth analysis of the chains of service functions that satisfy (or don't satisfy) the given policies.

This required both the extension of the existing data format representations (XML and JSON), the design and the implementation of new resources for the restful API, and the modification of the existing web client used to communicate with the server.

Furthermore, the existing XML and JSON representations has been upgraded by adding additional constraint and by improving the reuse of existing elements.

## 2 Preliminary Notes

### 2.1 Folder Structure

The files and folders described here are not the full set of documents that compose the Verigraph folder, but simply the ones that has been modified during the project.

- examples/ - Example graph configurations that can be loaded into the web client to test the reachability policies
- jsonschemas/ - JSON schemas that are used to validate the information received by the web client or loaded from a file
- schema/ - XML schema that is used to generate the JAXB classes used to model the different elements of the service
- src/ - Java classes
  - it/polito/neo4j/ - Packages used for handling the storage of the graph into neo4j or to retrieve the graph information from neo4j
  - it/polito/verigraph/deserializer - Deserialize a JSON representation to a model object
  - it/polito/verigraph/model - Model the elements of the service in the java application
  - it/polito/verigraph/resources - Model the resources available through the REST API
  - it/polito/verigraph/serializer - Serialize model objects to JSON representations
  - it/polito/verigraph/service - Implement the services requested to resource objects
  - it/polito/verigraph/solver - Classes used to provide the data structures and functions necessary for the policy verification
  - it/polito/verigraph/validation - Correctly validate some elements represented as JsonNode objects
- gen-src/it/polito/neo4j/jaxb - Classes automatically generated from the xml schema
- target/ - Folder of the war file
- webapp/ - Web client used to access the REST API

- build.xml — Ant script to automate the compiling and the deployment
- tomcat-build.xml — Ant script to control tomcat

## 3 Verigraph XML Schema Documentation

### 3.1 Differences with the previous version

In this work the original XML schema was improved in several ways. Here we list the changes and improvements that were made:

Improvements	Details
New constraints	The original XML schema did not include most of the reference constraints required to check the existence of a node that was referred by another node. For instance, a NAT or a Firewall could be targetting a node that did not exist, and a vpnaccess element could be referring to a node that was not a vpnexit element.
PacketType	In the original XML schema there were two elements (Endhost and FieldModifier) that used the same attributes to describe a subset of the fields that composed a packet. In the new version, this redundancy has been removed by introducing a new complex type PacketType that centralizes the description of the subset of the supported fields. Moreover, the same complex type can also be used to characterize a traffic flow.
Policies, Policy, Restrictions	New elements and types have been introduced to model the different policies that can be applied to a graph.
Firewall	The firewall's ACL has been extended to include also the source/destination ports and the transport protocol.

### 3.2 Verigraph XML Schema

Listing 1: XML Example

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <graphs xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
  xml_components_v2.xsd">
3   <graph id="0">
4     <node functional_type="FIREWALL" id="0" name="fw">
5       <neighbour id="0" name="nodeSrc"/>
6       <neighbour id="1" name="nodeDest"/>
7       <configuration description="Simple Description" id="0" name="conf1">
8         <firewall>
9           <elements>
10            <source>nodeSrc</source>
11            <destination>nodeDest</destination>
12            <srcPort>30</srcPort>
13            <destPort>30</destPort>
14            <protocol>TCP</protocol>
15          </elements>
16        </firewall>
17      </configuration>
18    </node>
19    <node functional_type="ENDPOINT" id="1" name="nodeSrc">
20      <neighbour id="0" name="fw"/>
21      <configuration description="Simple description" id="0" name="conf2">
22        <endhost />
23      </configuration>
24    </node>
25    <node functional_type="ENDPOINT" id="2" name="nodeDest">
26      <neighbour id="0" name="fw"/>
27      <configuration description="Simple description" id="0" name="conf3">
28        <endhost />
29      </configuration>
30    </node>
31  </graph>
```

```

32 <policies>
33   <policy destination="nodeDest" id="0" name="policyExample" source="nodeSrc">
34     <trafficflow protocol="HTTP_REQUEST"/>
35     <restrictions type="SELECTION">
36       <genericFunction type="FIREWALL"/>
37     </restrictions>
38   </policy>
39 </policies>
40 </graph>
41 </graphs>

```


## Graphs

Graphs is the root element of the XML schema. It contains a list of **Graph** elements.

### Graph

A Graph is a set of services that will be deployed in the network. It is contained inside a list of Graphs. Graph is characterised by

- A unique **ID**
- A list of **Node** elements
- (Optionally) A **Policies** element that contains a list of **Policy**

 **Warning:** You must define a Graph that has at least one Node.

Listing 2: Graphs Example

```

1 <graphs xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
  xml_components_v2.xsd">
2   <graph id="0">
3     <node functional_type="FIREWALL" id="0" name="fw">
4       ...
5     </node>
6     <node functional_type="ENDPOINT" id="1" name="nodeSrc">
7       ...
8     </node>
9     <node functional_type="ENDPOINT" id="2" name="nodeDest">
10      ...
11    </node>
12
13    <policies>
14      <policy destination="nodeDest" id="0" source="nodeSrc">
15        ...
16      </policy>
17    </policies>
18  </graph>
19 </graphs>

```

Listing 3: Graphs schema code snippet

```

1 <xsd:element name="graphs">
2   <xsd:complexType>
3     <xsd:sequence>
4       <xsd:element ref="graph" maxOccurs="unbounded"

```

```


5         minOccurs="0" />
6     </xsd:sequence>
7 </xsd:complexType>
8 <xsd:unique name="uniqueGraph">
9     <xsd:selector xpath="graph" />
10    <xsd:field xpath="@id" />
11 </xsd:unique>
12</xsd:element>
13<xsd:element name="graph">
14    <xsd:complexType>
15        <xsd:sequence>
16            <xsd:element ref="node" maxOccurs="unbounded"/>
17            <xsd:element ref="policies" minOccurs="0"/>
18        </xsd:sequence>
19        <xsd:attribute name="id" type="xsd:long" use="optional" />
20    </xsd:complexType>
21    ...
22</xsd:element>

```

## Node

A Node is a logical network element that correspond to a Network Function. A node is characterised by:

- A unique **Name**
- A **Functional Type**
- A list of **Neighbour Node Names**
- A **Configuration** for the Functional Type

 **Warning:** Pay attention when you define the neighbours of a node. The neighbour node names must exist within the specific graph.

Listing 4: Node Example

```

1 <node functional_type="ENDPOINT" id="1" name="nodeSrc">
2   <neighbour id="0" name="fw"/>
3   <configuration ...>
4     ...
5   </configuration>
6 </node>

```

Listing 5: Node schema code snippet

```

1 <xsd:element name="graph">
2   ...
3   <xsd:unique name="uniqueNodeId">
4       <xsd:selector xpath="node" />
5       <xsd:field xpath="@id" />
6   </xsd:unique>
7   ...
8 </xsd:element>
9 <xsd:element name="node">
10    <xsd:complexType>
11        <xsd:sequence>
12            <xsd:element ref="neighbour" maxOccurs="unbounded"

```

```

13         minOccurs="0" />
14     <xsd:element ref="configuration" maxOccurs="1"
15         minOccurs="1" />
16 </xsd:sequence>
17 <xsd:attribute name="id" type="xsd:long" use="optional" />
18 <xsd:attribute name="name" type="xsd:string"
19     use="required" />
20 <xsd:attribute name="functional_type" type="functionalTypes"
21     use="required" />
22 </xsd:complexType>
23 ...
24 </xsd:element>

```

## Functional Type

A Node can be a:

- FIREWALL
- ENDMETHOD
- ENDPOINT
- ANTISPAM
- CACHE
- DPI
- MAILCLIENT
- MAILSERVER
- NAT
- VPNACCESS
- VPNEXIT
- WEBCLIENT
- WEBSERVER
- FIELDMODIFIER

## Configuration

In this section we describe the different type of configurations that can be provided. A configuration is characterized by an *unique* name and by an *optional* description.

### Firewall

A Firewall Configuration contains a list of ACLs (elements). The ACL defines a tuple of:


- Source node ID
- Destination node ID
- Source port

- Destination port
- The **Transport Protocol Type**

This tuple represents the connection that will be blocked.

The **Transport Protocol Type** can be:

- TCP
- UDP

 **Warning:** Source and destination nodes must exist in the same graph.

Listing 6: Firewall Configuration Example

```

1 <configuration description="Simple Description" id="0" name="conf1">
2   <firewall>
3     <elements>
4       <source>nodeSrc</source>
5       <destination>nodeDest</destination>
6       <srcPort>30</srcPort>
7       <destPort>30</destPort>
8       <protocol>TCP</protocol>
9     </elements>
10  </firewall>
11 </configuration>

```

Listing 7: Firewall schema code snippet

```

1 <xsd:element name="graph">
2   ...
3   <xsd:keyref name="keyRefNodeSrc" refer="keyNode">
4     <xsd:selector xpath="node/neighbour" />
5     <xsd:field xpath="@name" />
6   </xsd:keyref>
7   <!-- Firewall -->
8   <xsd:keyref name="keyRefFirewallSrc" refer="keyNode">
9     <xsd:selector xpath="node/configuration/firewall/elements" />
10    <xsd:field xpath="source" />
11  </xsd:keyref>
12  <xsd:keyref name="keyRefFirewallDest" refer="keyNode">
13    <xsd:selector xpath="node/configuration/firewall/elements" />
14    <xsd:field xpath="destination" />
15  </xsd:keyref>
16  ...
17 </xsd:element>
18 <xsd:element name="firewall">
19   <xsd:complexType>
20     <xsd:sequence>
21       <xsd:element ref="elements" minOccurs="0" maxOccurs="unbounded" />
22     </xsd:sequence>
23   </xsd:complexType>
24 </xsd:element>
25 <xsd:element name="elements">
26   <xsd:complexType>
27     <xsd:sequence>
28       <xsd:element name="source" type="xsd:string" />
29       <xsd:element name="destination" type="xsd:string" />
30       <xsd:element name="srcPort" type="xsd:positiveInteger" />
31       <xsd:element name="destPort" type="xsd:positiveInteger" />

```



```

32     <xsd:element name="protocol" type="transportProtocolTypes" />
33   </xsd:sequence>
34 </xsd:complexType>
35 </xsd:element>

```

## EndHost

An EndHost Configuration contains a description of the traffic flow that is generated from/received by this node.

Listing 8: EndHost Configuration Example

```

1 <configuration description="Simple Description" id="0" name="conf1">
2   <endhost body="lorem ipsum"/>
3 </configuration>

```

Listing 9: EndHost schema code snippet

```

1 <xsd:element name="endhost" type="packetType" />

```

## FieldModifier

A Field Modifier Configuration contains the fields of the traffic flow that are modified by this node.

Listing 10: FieldModifier Configuration Example

```

1 <configuration description="Simple Description" id="0" name="conf1">
2   <fieldmodifier body="This will be changed"/>
3 </configuration>

```

Listing 11: Fieldmodifier schema code snippet

```

1 <xsd:element name="fieldmodifier" type="packetType"/>

```

## Packet Type

Each packet and traffic flow can be characterized by the following attributes (all of them are optional):

- Body
- Sequence
- Protocol
- Email From
- URL
- Options
- Destination

Listing 12: PacketType schema code snippet


```

1 <xsd:complexType name="packetType">
2   <xsd:attribute name="body" type="xsd:string" />
3   <xsd:attribute name="sequence" type="xsd:integer" />
4   <xsd:attribute name="protocol" type="protocolTypes" />
5   <xsd:attribute name="email_from" type="xsd:string" />
6   <xsd:attribute name="url" type="xsd:string" />
7   <xsd:attribute name="options" type="xsd:string" />
8   <xsd:attribute name="destination" type="xsd:string" />
9 </xsd:complexType>

```

## AntiSpam

An Antispam Configuration contains a list of source nodes that represent the blacklisted mail clients and servers.

 **Warning:** The nodes to which the Antispam refers must exist in the same graph.

Listing 13: Antispam Configuration Example

```

1 <configuration description="Simple Description" id="0" name="conf1">
2   <antispam>
3     <source name="node1" />
4   </antispam>
5 </configuration>

```

Listing 14: Antispam schema code snippet


```

1 <xsd:element name="graph">
2   ...
3   <xsd:keyref name="keyRefNodeSrc" refer="keyNode">
4     <xsd:selector xpath="node/neighbour" />
5     <xsd:field xpath="@name" />
6   </xsd:keyref>
7   ...
8   <!-- Antispam -->
9   <xsd:keyref name="keyRefAntispam" refer="keyNode">
10    <xsd:selector xpath="node/configuration/antispam/source" />
11    <xsd:field xpath="@name" />
12  </xsd:keyref>
13  ...
14 </xsd:element>
15 <xsd:element name="antispam">
16   <xsd:complexType>
17     <xsd:sequence minOccurs="1" maxOccurs="unbounded">
18       <xsd:element name="source" type="addressType" />
19     </xsd:sequence>
20   </xsd:complexType>
21 </xsd:element>

```

## Cache

A Cache Configuration contains a list of resources. A resource is a node internal to the cache, and it is characterized by the name of the node it refers to.

 **Warning:** The nodes to which the cache refers must exist in the same graph.

Listing 15: Cache Configuration Example

```
1 <configuration description="Simple Description" id="0" name="conf1">
2   <cache>
3     <resource name="node1"/>
4     <resource name="node2"/>
5   </cache>
6 </configuration>
```

Listing 16: Cache schema code snippet

```
1 <xsd:element name="graph">
2   ...
3   <xsd:keyref name="keyRefNodeSrc" refer="keyNode">
4     <xsd:selector xpath="node/neighbour" />
5     <xsd:field xpath="@name" />
6   </xsd:keyref>
7   ...
8   <!-- Cache -->
9   <xsd:keyref name="keyRefCache" refer="keyNode">
10    <xsd:selector xpath="node/configuration/cache/resource" />
11    <xsd:field xpath="@name" />
12  </xsd:keyref>
13  ...
14 </xsd:element>
15 <xsd:element name="cache">
16   <xsd:complexType>
17     <xsd:sequence minOccurs="1" maxOccurs="unbounded">
18       <xsd:element name="resource" type="addressType" />
19     </xsd:sequence>
20   </xsd:complexType>
21 </xsd:element>
```

## DPI

A DPI Configuration contains a list of notAllowed elements, that defines the strings that can't be present inside a packet otherwise it will be dropped.

Listing 17: DPI Configuration Example


```
1 <configuration description="Simple Description" id="0" name="conf1">
2   <dpi>
3     <notAllowed>Some String</notAllowed>
4   </dpi>
5 </configuration>
```

Listing 18: DPI schema code snippet

```
1 <xsd:element name="dpi">
2   <xsd:complexType>
3     <xsd:sequence minOccurs="1" maxOccurs="unbounded">
4       <xsd:element name="notAllowed" type="xsd:string" />
5     </xsd:sequence>
6   </xsd:complexType>
7 </xsd:element>
```

## MailClient

A Mail Client Configuration contains the Mail Server name.

 **Warning:** The Mail Server to which this node refers must exist within the specific graph.

Listing 19: MailClient Configuration Example

```
1 <configuration description="Simple Description" id="0" name="conf1">
2   <mailclient mailserver="ServerName"/>
3 </configuration>
```

## MailServer

A Mail Server Configuration contains the Mail Server name.

Listing 20: MailServer Configuration Example

```
1 <configuration description="Simple Description" id="0" name="conf1">
2   <mailserver>
3     <name>ServerName</name>
4   </mailserver>
5 </configuration>
```

Listing 21: MailClient and MailServer schema code snippet

```
1 <xsd:element name="graph">
2   ...
3   <!-- WebClient / WebServer -->
4   <xsd:key name="keyWebServer">
5     <xsd:selector xpath="node/configuration/webserver" />
6     <xsd:field xpath="name" />
7   </xsd:key>
8   <xsd:keyref name="keyRefWebClient" refer="keyWebServer">
9     <xsd:selector xpath="node/configuration/webclient" />
10    <xsd:field xpath="@nameWebServer" />
11  </xsd:keyref>
12  <!-- WebClient / WebServer -->
13  <xsd:key name="keyMailServer">
14    <xsd:selector xpath="node/configuration/mailserver" />
15    <xsd:field xpath="name" />
16  </xsd:key>
17  <xsd:keyref name="keyRefMailClient" refer="keyMailServer">
18    <xsd:selector xpath="node/configuration/mailclient" />
19    <xsd:field xpath="@mailserver" />
20  </xsd:keyref>
21  ...
22 </xsd:element>
23 <xsd:element name="mailclient">
24   <xsd:complexType>
25     <xsd:attribute name="mailserver" type="xsd:string"
26       use="required" />
27   </xsd:complexType>
28 </xsd:element>
29 <xsd:element name="mailserver">
30   <xsd:complexType>
31     <xsd:sequence>
32       <xsd:element name="name" type="xsd:string" />
```


```

33     </xsd:sequence>
34 </xsd:complexType>
35 </xsd:element>

```

## NAT

A NAT Configuration contains a list of internal nodes. The **source** element define a node in the graph.

 **Warning:** The nodes to which the NAT refers must exist in the same graph.

Listing 22: NAT Configuration Example

```

1 <configuration description="Simple Description" id="0" name="conf1">
2   <nat>
3     <source name="node1"/>
4     <source name="node2"/>
5   </nat>
6 </configuration>

```

Listing 23: NAT schema code snippet


```

1 <xsd:element name="graph">
2   ...
3   <xsd:keyref name="keyRefNodeSrc" refer="keyNode">
4     <xsd:selector xpath="node/neighbour" />
5     <xsd:field xpath="@name" />
6   </xsd:keyref>
7   ...
8   <!-- NAT -->
9   <xsd:keyref name="keyRefNAT" refer="keyNode">
10    <xsd:selector xpath="node/configuration/nat/source" />
11    <xsd:field xpath="@name" />
12  </xsd:keyref>
13  ...
14 </xsd:element>
15 <xsd:element name="nat">
16   <xsd:complexType>
17     <xsd:sequence minOccurs="1" maxOccurs="unbounded">
18       <xsd:element name="source" type="addressType" />
19     </xsd:sequence>
20   </xsd:complexType>
21 </xsd:element>

```

## VPNAccess

A VpnAccess Configuration contains the VpnExit name.

 **Warning:** The VPNExit name must exist within the same graph.

Listing 24: VPNAccess Configuration Example


```

1 <configuration description="Simple Description" id="0" name="conf1">
2   <vpnaccess vpnexit="vpe1">
3     <name>vpa1</name>
4   </vpnaccess>
5 </configuration>

```

## VPNExit

A VpnExit Configuration contains the VpnAccess name.

 **Warning:** The VPNAccess name must exist within the same graph.

Listing 25: VPNExit Configuration Example


```
1 <configuration description="Simple Description" id="0" name="conf1">
2   <vpnexit vpnaccess="vpa1">
3     <name>vpe1</name>
4   </vpnexit>
5 </configuration>
```

Listing 26: VPNAccess and VPNExit schema code snippet

```
1 <xsd:element name="graph">
2   ...
3   <!-- VPN -->
4   <xsd:key name="keyVPNAccess">
5     <xsd:selector xpath="node/configuration/vpnaccess" />
6     <xsd:field xpath="name" />
7   </xsd:key>
8   <xsd:keyref name="keyRefVPNAccess" refer="keyVPNAccess">
9     <xsd:selector xpath="node/configuration/vpnexit" />
10    <xsd:field xpath="@vpnaccess" />
11  </xsd:keyref>
12  <xsd:key name="keyVPNExit">
13    <xsd:selector xpath="node/configuration/vpnexit" />
14    <xsd:field xpath="name" />
15  </xsd:key>
16  <xsd:keyref name="keyRefVPNExit" refer="keyVPNExit">
17    <xsd:selector xpath="node/configuration/vpnaccess" />
18    <xsd:field xpath="@vpnexit" />
19  </xsd:keyref>
20  ...
21 </xsd:element>
22 <xsd:element name="vpnaccess">
23   <xsd:complexType>
24     <xsd:sequence>
25       <xsd:element name="name" type="xsd:string" />
26     </xsd:sequence>
27     <xsd:attribute name="vpnexit" type="xsd:string"
28       use="required" />
29   </xsd:complexType>
30 </xsd:element>
31 <xsd:element name="vpnexit">
32   <xsd:complexType>
33     <xsd:sequence>
34       <xsd:element name="name" type="xsd:string" />
35     </xsd:sequence>
36     <xsd:attribute name="vpnaccess" type="xsd:string"
37       use="required" />
38   </xsd:complexType>
39 </xsd:element>
```

## WebClient

A Web Client Configuration contains the Web Server name.

 **Warning:** The Web Server to which this node refers must exist within the specific graph.

Listing 27: WebClient Configuration Example

```
1 <configuration description="Simple Description" id="0" name="conf1">
2   <webclient nameWebServer="ServerName"/>
3 </configuration>
```

## WebServer

A Web Server Configuration contains the Web Server name.

Listing 28: WebServer Configuration Example

```
1 <configuration description="Simple Description" id="0" name="conf1">
2   <webserver>
3     <name>ServerName</name>
4   </webserver>
5 </configuration>
```

Listing 29: WebClient and WebServer schema code snippet


```
1 <xsd:element name="graph">
2   ...
3   <!-- WebClient / WebServer -->
4   <xsd:key name="keyWebServer">
5     <xsd:selector xpath="node/configuration/webserver" />
6     <xsd:field xpath="name" />
7   </xsd:key>
8   <xsd:keyref name="keyRefWebClient" refer="keyWebServer">
9     <xsd:selector xpath="node/configuration/webclient" />
10    <xsd:field xpath="@nameWebServer" />
11  </xsd:keyref>
12  <xsd:key name="keyMailServer">
13    <xsd:selector xpath="node/configuration/mailserver" />
14    <xsd:field xpath="name" />
15  </xsd:key>
16  <xsd:keyref name="keyRefMailClient" refer="keyMailServer">
17    <xsd:selector xpath="node/configuration/mailclient" />
18    <xsd:field xpath="@mailserver" />
19  </xsd:keyref>
20  ...
21 </xsd:element>
22 <xsd:element name="webclient">
23   <xsd:complexType>
24     <xsd:attribute name="nameWebServer" type="xsd:string"
25       use="required" />
26   </xsd:complexType>
27 </xsd:element>
28 <xsd:element name="webserver">
29   <xsd:complexType>
30     <xsd:sequence>
31       <xsd:element name="name" type="xsd:string" />
32     </xsd:sequence>
33   </xsd:complexType>
34 </xsd:element>
```

## Policy

A Policy contains a reachability policy that we want to verify in a network graph. It is contained inside a list of Policies.

A Policy is characterised by

- A unique **ID**
- The **Source Node** name
- The **Destination Node** name
- The **Traffic Flow** to which this policy refers.
- A **Restrictions** element containing the restrictions to be checked.

 **Warning:** The source and destination names must exist within the same graph of the policy.

Listing 30: Policies Example

```
1 <policies>
2   <policy destination="node2" source="node1" id="1" name="example">
3     <trafficflow body="Lorem Ipsum"/>
4     <restrictions ...>
5       ...
6     </restrictions>
7   </policy>
8 </policies>
```

Listing 31: Policy schema code snippet

```
1 <xsd:element name="graph">
2   ...
3   <!-- Integrity constraints on the policies that must refer to existing nodes -->
4   <xsd:keyref name="keyRefPolicySrc" refer="keyNode">
5     <xsd:selector xpath="policies/policy" />
6     <xsd:field xpath="@source" />
7   </xsd:keyref>
8   <xsd:keyref name="keyRefPolicyDest" refer="keyNode">
9     <xsd:selector xpath="policies/policy" />
10    <xsd:field xpath="@destination" />
11  </xsd:keyref>
12  <xsd:keyref name="keyRefPolicyFunction" refer="keyNode">
13    <xsd:selector xpath="policies/policy/restrictions/exactFunction" />
14    <xsd:field xpath="name" />
15  </xsd:keyref>
16  ...
17 </xsd:element>
18 <xsd:element name="policies">
19   <xsd:complexType>
20     <xsd:sequence>
21       <xsd:element ref="policy" maxOccurs="unbounded"/>
22     </xsd:sequence>
23   </xsd:complexType>
24   <xsd:unique name="uniquePolicy">
25     <xsd:selector xpath="policy" />
26     <xsd:field xpath="@id" />
27   </xsd:unique>
```



```

28 </xsd:element>
29 <xsd:element name="policy">
30   <xsd:complexType>
31     <xsd:sequence>
32       <xsd:element name="trafficflow" type="packetType"/>
33       <xsd:element ref="restrictions"/>
34     </xsd:sequence>
35     <xsd:attribute name="id" type="xsd:long"
36       use="required" />
37     <xsd:attribute name="name" type="xsd:string"
38       use="required" />
39     <xsd:attribute name="source" type="xsd:string"
40       use="required" />
41     <xsd:attribute name="destination" type="xsd:string"
42       use="required" />
43   </xsd:complexType>
44 </xsd:element>

```

## Restrictions

A Restrictions element is composed of a type and a set of service functions which the specified traffic must pass through.

The possible types of a Restrictions element can be:

- **SELECTION**
- **SET**
- **SEQUENCE**
- **LIST**

The service functions can be of two types:

- **genericFunction**: It describes a generic service function, without specifying the name of the exact node to which it refers.
- **exactFunction**: It describes a specific service function that **must exist** within the same graph of the policy.

Listing 32: Restrictions Example

```

1 <restrictions type="SELECTION">
2   <genericFunction type="FIREWALL" />
3   <exactFunction>
4     <name>node1</name>
5   </exactFunction>
6   <genericFunction type="NAT" />
7 </restrictions>

```

Listing 33: Restrictions schema code snippet

```

1 <xsd:element name="restrictions">
2   <xsd:complexType>
3     <xsd:choice maxOccurs="unbounded">
4       <xsd:element ref="genericFunction"/>
5       <xsd:element ref="exactFunction" />
6     </xsd:choice>
7     <xsd:attribute name="type" type="restrictionTypes"

```

```
8         use="required" />
9     </xsd:complexType>
10 </xsd:element>
11 <xsd:element name="genericFunction">
12     <xsd:complexType>
13         <xsd:attribute name="type" type="functionalTypes" use="required"/>
14     </xsd:complexType>
15 </xsd:element>
16 <xsd:element name="exactFunction">
17     <xsd:complexType>
18         <xsd:sequence>
19             <xsd:element name="name" type="xsd:string" />
20         </xsd:sequence>
21     </xsd:complexType>
22 </xsd:element>
```

## 4 Verigraph JSON Schemas Documentation

The interaction between the server and the web client is based on data in json format. Because of this, different json schemas and representations are used for the following reasons:

- Describe a graph within the web interface and allow the exchange of data about the graph between client and server.
- Server validation of the data received by the web client.
- Provisioning of the results of a verification request to the client.

In this chapter we summarize how the json schemas has been changed to support both policies and the improved version of existing elements. For a more detailed description of the elements involved, please refer to the chapter on the XML schema documentation.

### 4.1 Graph JSON Schema

Listing 34: JSON Example

```
1 {
2   "id": 0,
3   "nodes": [
4     {
5       "name": "source",
6       "functional_type": "endpoint",
7       "neighbours": [],
8       "configuration": []
9     },
10    {
11      "name": "dest",
12      "functional_type": "endpoint",
13      "neighbours": [],
14      "configuration": []
15    }
16  ],
17  "policies": [
18    {
19      "policyName": "example",
20      "source": "source",
21      "target": "dest",
22      "trafficFlow": [],
23      "restrictions": {
24        "type": "selection",
25        "functions": []
26      }
27    }
28  ]
29 }
```

#### Graph

The JSON representation of a graph has been extended to include the list of policies that were defined for that graph.

#### Policy

A Policy is an object characterized by the following properties:

- **policyName**: The unique name of the policy
- **source**: The ID of the source node
- **target**: The ID of the destination node
- **trafficFlow**: The traffic flow to which this policy refers.
- **restrictions**: The element containing the restrictions to be checked.

## Restrictions

A Restrictions element is composed of a type (selection, set, sequence or list) and a set of service functions which the specified traffic must pass through. Each service function is characterized by two parameters:

- **funcType**: The type of service function (generic, exact)
- **funcName**: The name of the referred function

Listing 35: Restrictions Example

```

1 "restrictions": {
2     "type": "selection",
3     "functions": [
4         {
5             "funcType": "generic",
6             "funcName": "firewall"
7         }
8     ]
9 }
```

A schema was defined to correctly validate the array of functions of a restrictions element.

Listing 36: Restrictions' functions schema

```

1 {
2     "$schema": "http://json-schema.org/draft-04/schema#",
3     "title": "Functions'",
4     "description": "Polito restrictions to policy",
5     "type": "array",
6     "items": {
7         "type": "object",
8         "properties": {
9             "funcType" : { "type": "string"},
10            "funcName" : { "type": "string"}
11        }
12    },
13    "minItems": 0
14 }
```

## PacketType

The endhost configuration schema used for validating endhost nodes has been extended to be used for validating fieldmodifier nodes and the traffic flow of a policy.

Listing 37: PacketType Example

```

1 "trafficFlow": [
2   {
3     "body": "lorem ipsum",
4     "destination": "dest"
5   }
6 ]

```

Listing 38: PacketType schema

```

1 {
2   "$schema": "http://json-schema.org/draft-04/schema#",
3   "title": "PacketType",
4   "description": "General description of a subset of the fields that characterize a packet or
5     a traffic flow.",
6   "type": "array",
7   "items": {
8     "type": "object",
9     "properties": {
10      "body": {
11        "description": "HTTP body",
12        "type": "string"
13      },
14      "sequence": {
15        "description": "Sequence number",
16        "type": "integer"
17      },
18      "protocol": {
19        "description": "Protocol",
20        "type": "string",
21        "enum": ["HTTP_REQUEST", "HTTP_RESPONSE", "POP3_REQUEST", "POP3_RESPONSE"]
22      },
23      "email_from": {
24        "description": "E-mail sender",
25        "type": "string"
26      },
27      "url": {
28        "description": "URL",
29        "type": "string"
30      },
31      "options": {
32        "description": "Options",
33        "type": "string"
34      },
35      "destination": {
36        "description": "Destination node",
37        "type": "string"
38      }
39    },
40    "additionalProperties": false
41  },
42  "maxItems": 1
43 }

```

## Firewall

The schema used for validating the configuration of firewall nodes has been extended with the properties describing the source and destination ports, as well as with the property that describes the protocol type.

Listing 39: Firewall example

```

1 {
2   "name": "firewall",
3   "functional_type": "firewall",
4   "neighbours": [
5     ...
6   ],
7   "configuration": [
8     {
9       "source_id": "nat",
10      "destination_id": "dpi",
11      "source_port": 10,
12      "destination_port": 30,
13      "protocol": "TCP"
14    }
15  ]
16 }

```

Listing 40: Firewall schema

```

1 {
2   "$schema": "http://json-schema.org/draft-04/schema#",
3   "title": "Firewall",
4   "description": "Polito Firewall",
5   "type": "array",
6   "items": {
7     "type": "object",
8     "properties": {
9       "source_id": { "type": "string"},
10      "destination_id": { "type": "string"},
11      "source_port" : { "type": "integer"},
12      "destination_port" : { "type": "integer"},
13      "protocol" : { "type": "string"}
14    }
15  },
16  "minItems": 0,
17  "uniqueItems": true
18 }

```

## 4.2 Verification Results JSON Representation

This is the output of the Verigraph's PolicyVerifier resource. It contains the following information:

- **result**: It summarizes the outcome of the policy's verification request. The possible outcomes are: **SAT** (the policy is satisfied), **UNSAT** (the policy is unsatisfied), **UNKNOWN**.
- **comment**: It gives additional information about the outcome of the policy's verification request.
- **tests**: The list of the chains that were evaluated while verifying the policy. The content of the list depends on the specific verification request that was made by the user (see chapter about the rest API).

Furthermore, each chain is characterized by the following information:

- **result**: It summarizes the outcome of the policy's verification request for this specific chain. The possible outcomes are: **SAT** (the chain satisfies the policy), **UNSAT** (the chain doesn't satisfy the policy), **UNKNOWN**.

- **comment:** It gives additional information about the outcome of the policy's verification request. If the outcome is UNSAT because a function in the chain is blocking the traffic specified in the policy, then this property will report such function to the user.
- **path:** The list of the functions in the chain.

Listing 41: VerificationResults example

```
1 {
2   "result": "SAT",
3   "comment": "There is at least one path 'source' can use to reach 'dest'. See all the
4     available paths below",
5   "tests": [
6     {
7       "result": "SAT",
8       "comment": "'source' is able to reach 'dest'",
9       "path": [
10        ...
11      ],
12    },
13    {
14      "result": "UNSAT",
15      "comment": "'source' is unable to reach 'dest' because of node 'firewall'",
16      "path": [
17        ...
18      ]
19    }
20  ]
21 }
```

## 5 Rest API Description

Note: The documentation is also available as Swagger documentation, which can be accessed through the Verigraph web client.

### 5.1 Resource Design

Resources	URLs	XML Repr	Meaning
Graphs	/graphs	graphs	Set of all graphs
Graph	/graphs/{graphId}	graph	Single graph
Nodes	/graphs/{graphId} /nodes	[node]	Set of all nodes of a specific graph
Node	/graphs/{graphId} /nodes/{nodeId}	node	Single node
Configuration	/graphs/{graphId} /nodes/{nodeId} /configuration	configuration	Configuration applied to a node
Neighbours	/graphs/{graphId} /nodes/{nodeId} /neighbours	[neighbour]	Set of neighbours of a specific node
Neighbour	/graphs/{graphId} /nodes/{nodeId} /neighbours/{neighbourId}	neighbour	Single neighbour
Paths	/graphs/{graphId} /paths	paths	Set of nodes traversed to reach a destination from a source
Policies	/graphs/{graphId} /policies	policies	Set of all policies belonging to a specific graph
Policy	/graphs/{graphId} /policies/{policyId}	policy	Single policy
Restrictions	/graphs/{graphId} /policies/{policyId} /restrictions	restriction	Restrictions applied to the policy
PolicyVerifier	/graphs/{graphId} /policyVerifier		Verify the selected policy

The policyVerifier resource is designed to replace the policy resource (/graphs/graphId/policy) cur-



rently available in the existing service. Further details are listed in the next section.

## 5.2 Operation Design

Resources	Method	Query Param	Req. Body	Status	Resp. Body	Meaning
Graphs	GET			200	Graphs	Return an array of all the available graphs
	POST	name:string id:long email:string	Graph	201	Graph	Graph successfully created
Graph	GET			200	Graph	Returns a single graph
	PUT		Graph	200	Graph	Graph successfully edited
	DELETE			204		Graph successfully deleted
Nodes	GET			200	Nodes	Return an array of the nodes belonging to {graphId}
	POST		Node	201	Node	Node successfully created
Node	GET			200	Node	Returns a single node
	PUT		Node	200	Node	Node successfully edited
	DELETE			204		Node successfully deleted
Configuration	PUT		Configuration	200	Configuration	Configuration successfully added/updated for the requested node
Neighbours	GET			200	Neighbours	Returns an array of neighbours of a given node belonging to graph {graphId}
	POST		Neighbour	200	Neighbour	Adds single neighbour to a given node belonging to a given graph
Neighbour	GET			200	Neighbour	Returns a single neighbour
	PUT		Neighbour	200	Neighbour	Edits a single neighbour of a given node belonging to a given graph
	DELETE			204		Neighbour successfully deleted
Paths	GET	source:string destination:string		200	[Paths]	Retrieve all paths between two nodes
Policies	GET			200	Policies	Return all the policies related to {graphId}
	POST		Policy	201	Policy	Policy successfully created
Policy	GET			200	Policy	Returns a single policy
	PUT		Policy	200	Policy	Policy successfully edited
	DELETE			204		Policy successfully deleted
Restrictions	PUT		Restrictions	200	Restrictions	Restriction successfully updated for the requested policy
PolicyVerifier	GET	type:string policyId:long source:string destination:string		200	VerificationResult	Policy successfully verified

Depending on the specific operation that was requested, different errors can occur:

Status	Operations	Response Body	Meaning
400	POST, PUT	ErrorMessage	Invalid graph / node / policy / restrictions element supplied
403	GET, POST, PUT, DELETE	ErrorMessage	Invalid graph / node / policy id supplied
404	GET, POST, PUT, DELETE	ErrorMessage	Graph / node / policy not found
500	GET, POST, PUT, DELETE	ErrorMessage	Server Error

The policyVerifier resource improves and extends the behavior of the existing policy resource. Here is the full list of requests that can be made to the policyVerifier resource:

- **Reachability:** It checks if there is at least one chain such that the traffic flow sent from the source node reaches the destination node. It returns the result of the verification, along with the list of the chains that were evaluated and, for each list, if the path satisfies the request.
- **Isolation:** It checks if there is at least one chain such that the traffic flow sent from the source node reaches the destination node without traversing the middlebox node. It returns the result of the verification, along with the list of the chains that were evaluated and, for each list, if the path satisfies the request.
- **Traversal:** It checks if there is at least one chain such that the traffic flow sent from the source node reaches the destination node by traversing the middlebox node. It returns the result of the verification, along with the list of the chains that were evaluated and, for each list, if the path satisfies the request.
- **VOne:**It verifies if at least one chain exists that enforces the requirements in the selected policy. It returns the result of the verification, along with the list of the chains that satisfy this constraint (if the verification is successful) or the list of the chains that don't satisfy this constraint (otherwise).
- **VOnly:**It verifies if only one chain exists that enforces the requirements in the selected policy. It returns the result of the verification, along with the list of the chains that satisfy this constraint (if the verification is successful or if the chains that enforce the requirements are more than one) or the list of the chains that don't satisfy this constraint (otherwise).
- **VMore:**It verifies if more than one chain exists that enforces the requirements in the selected policy. It returns the result of the verification, along with the list of the chains that satisfy this constraint (if the verification is successful) or the list of the chains that don't satisfy this constraint (otherwise).
- **VAll:**It verifies if all chains enforce the requirements in the selected policy. It returns the result of the verification, along with the list of the chains that satisfy this constraint (if the verification is successful) or the list of the chains that don't satisfy this constraint (otherwise).
- **VNone:**It verifies if there isn't any chain that enforces the requirements in the selected policy. It returns the result of the verification, along with the list of the chains that satisfy this constraint (empty if the verification is successful).

- **VOthers:** It returns the result of the verification, along with the list of the chains that don't satisfy this constraint (if the verification is successful) or the list of the chains that satisfy this constraint (otherwise).

## 6 Conclusions

In this project, the techniques seen in the Distributed Programming II course have been employed to enhance and extend the design and functionality of the Verigraph service, both from the point of view of the data format representations and of the resources available through the REST API.

As a final note, we list here some improvements that could be made in future works.

### Generalization of endhost functions

In the chapter about the Verigraph XML schema we distinguished between several types of configuration, all with similar functionalities (endhost, webclient, webserver, mailclient, mailserver). These configuration types could be generalized in a single endhost configuration that could be used to centralize the representation of the packets sent through the network. Even though a possible representation has been provided, this has not been implemented because it would have required several changes in both the Verigraph client and server. Moreover, this generalization would have required changes in the interaction with the Z3 prover.

Listing 42: Examples of generalized endhost configurations

```
1 <endhost>
2   <host />
3   <packet body="" destination="" email_from="" options="" protocol="HTTP_REQUEST" sequence="0"
4     url="" />
5 </endhost>
6
7 <endhost>
8   <webserver>
9     <name>Webserver</name>
10    </webserver>
11    <packet body="" destination="" email_from="" options="" protocol="HTTP_REQUEST" sequence="0"
12      url="" />
13  </endhost>
14
15 <endhost>
16   <webclient webserver="Webserver" />
17   <packet body="" destination="" email_from="" options="" protocol="HTTP_REQUEST" sequence="0"
18     url="" />
19 </endhost>
20
21 <endhost>
22   <webserver>
23     <name>Mailserver</name>
24    </webserver>
25    <packet body="" destination="" email_from="" options="" protocol="HTTP_REQUEST" sequence="0"
26      url="" />
27  </endhost>
28
29 <endhost>
30   <mailclient mailserver="Mailserver" />
31   <packet body="" destination="" email_from="" options="" protocol="HTTP_REQUEST" sequence="0"
32     url="" />
33 </endhost>
```

Listing 43: Proposal for a generalized endhost data representation

```
1 <xsd:element name="endhost">
2   <xsd:complexType>
3     <xsd:sequence>
4       <xsd:choice>
5         <xsd:element ref="host" />
```

```

6         <xsd:element ref="webserver" />
7         <xsd:element ref="webclient" />
8     <xsd:element ref="mailclient" />
9     <xsd:element ref="mailserver" />
10    </xsd:choice>
11    <xsd:element name="packet" type="packetType"/>
12    </xsd:sequence>
13    </xsd:complexType>
14</xsd:element>
15<xsd:element name="host">
16    <xsd:complexType />
17</xsd:element>
18<xsd:element name="mailclient">
19    <xsd:complexType>
20        <xsd:attribute name="mailserver" type="xsd:string"
21            use="required" />
22    </xsd:complexType>
23</xsd:element>
24<xsd:element name="mailserver">
25    <xsd:complexType>
26        <xsd:sequence>
27            <xsd:element name="name" type="xsd:string" />
28        </xsd:sequence>
29    </xsd:complexType>
30</xsd:element>
31<xsd:element name="webclient">
32    <xsd:complexType>
33        <xsd:attribute name="webserver" type="xsd:string"
34            use="required" />
35    </xsd:complexType>
36</xsd:element>
37<xsd:element name="webserver">
38    <xsd:complexType>
39        <xsd:sequence>
40            <xsd:element name="name" type="xsd:string" />
41        </xsd:sequence>
42    </xsd:complexType>
43</xsd:element>

```