Verifoo RestAPI and XML Docs

Antonio Varvara, Raffaele Sommese

Contents

1 Preliminary Information

1.1 Folder Structure

- docs/ – Documentation of the code (including javadoc)
  - VerifooDocs.pdf — Documentation of the web service and other useful information
  - verigraph_doc.pdf — Verigraph documentation for further details
- lib/ — All the external library (e.g. Z3 library)
  - junit/ — Library for running tests
  - lib4j/ — Library for manage the logging operations
- log/ — All the logs (for debugging purposes)
- resources/ —
  - log4j2.xml — Settings of the logs
- src/ — Java classes (for further information see the javadoc)
  - it/polito/verifoo/components/ — Basical Verifoo classes
  - it/polito/verifoo/rest/app/ — Classes to start the Rest application
  - it/polito/verifoo/rest/common/ — Classes that retrieve the informations from the XML and pass them to Verifoo
  - it/polito/verifoo/rest/jaxb/ — Automatically generated JAXB classes
  - it/polito/verifoo/rest/logger/ — Classes that handle the logging operations
  - it/polito/verifoo/rest/main/ — Main class for debugging purposes
  - it/polito/verifoo/rest/test/ — Test classes
  - it/polito/verifoo/rest/webservice/ — WebService classes
  - it/polito/verifoo/test — Simple examples on how Verifoo works
  - it/polito/verigraph/* — Basical Verigraph classes
- target/ — Folder for the war file
- testfile/ — XML files that are used to test the application
- WebContent/ — Files needed in order to deploy the service
- xsd/ — XML schemas needed for the application
  - errorSchema.xsd – XML schema of Rest error response
  - nfvInfo.xsd – XML schema of Verifoo
  - xml_components.xsd – XML schema of Verigraph (used into verifoo)
- build.xml — Ant script

1.2 Z3 Install Note

For the correct functioning of the application, you must provide the Z3 native library and include it to Java Library Path. The most convenient way to do this is add the path that the library to the dynamic linking library path.

- In Linux is LD_LIBRARY_PATH
- In MacOS is DYLD_LIBRARY_PATH
- In Windows is PATH

Make sure that you have download the correct version of Z3 according to your OS and your JVM endianness. In any case a mechanism for automatically adding the Z3 library to the path when it is deployed to a WebServer, is provided. It is tested on Tomcat and on Websphere and it works for the following distribution:

- Ubuntu x32
- Ubuntu x64
- Debian x64

1.3 Install and Testing Note

This project came with an Ant Script for compiling and testing purpose. The service is packaged into the WAR archive by issuing the command:

```
$ ant war
```

To test the internal component (Verifoo Proxy and Unmarshaller):

```
$ ant test
```

To test the WebService:

```
$ ant testWS
```

☞ **Warning**: You must have Tomcat installed and you must set $CATALINA_HOME accrodingly.

To start Tomcat with provided Ant Script.

```
$ ant start-tomcat
```

To deploy application and start Tomcat.

```
$ ant deploy
```

To deploy application to a running instance of Tomcat.

```
$ ant redeployWS
```

☞ **Warning**: You must configure tomcatUsername, tomcatPassword, tomcatPort and tomcatUrl in tomcat-build.xml accodling to your configuration.

2 XML Schemas

2.1 Verifoo XML Schema

Listing 1: XML Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<NFV xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    ↪ noNamespaceSchemaLocation="nfvInfo.xsd">
  <graphs>
    <graph id="0">
      <node functional_type="FIREWALL" name="node1">
        <neighbour name="nodeA"/>
        <neighbour name="node2"/>
        <configuration description="A simple description" name="conf1">
          <firewall>
            <elements>
              <source>nodeC</source>
              <destination>nodeD</destination>
            </elements>
          </firewall>
        </configuration>
      </node>
    </graph>
  </graphs>
 <CapacityDefinition>
    <CapacityForNode node="node1" capacity="10"/>
 </CapacityDefinition>
 <PropertyDefinition>
    <Property graph="0" name="IsolationProperty"/>
 </PropertyDefinition>
 <Hosts>
  <Host diskStorage="10" name="host1" type="CLIENT"/>
  <Host diskStorage="10" name="host2" type="MIDDLEBOX"/>
  <Host diskStorage="10" name="host3" type="SERVER"/>
 </Hosts>
 <Connections>
  <Connection sourceHost="host1" destHost="host2" avgLatency ="-1"/>
  <Connection sourceHost="host1" destHost="host3" avgLatency ="-10"/>
 </Connections>
 <ParsingString></ParsingString>
</NFV>
```

NFV

NFV is the root element of the XML schema, it must contain:

- A **Graphs** element that contains a list of **Graph**
- A list of **Capacity Definition**
- A list of **Property Definition** (one or more)
- An **Hosts** element that contains a list of **Host**
- A **Connections** element that contains a list of **Connection** between hosts
- An optional **Parsing String** used for the converter service.

Graph

A Graph is a chain of service that will be deployed in the network, it is contained inside a list of Graphs.
Verifoo can check and deploy multiple graphs.
Graph is characterised by

- A *unique* **ID**
- A list of **Nodes**

☞ **Warning**: You must define a graph that have at least 1 Client and 1 Server otherwise an exception will be thrown.

Listing 2: Graphs Example

```
<graphs>
   <graph id="0">
     <node ....>
   </graph>
   <graph id="1">
     ....
   </graph>
   <graph id="2">
     ....
   </graph>
</graphs>
```

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 Types

☞ **Warning**: Pay attention when you define the neighbours of a node, remember the graph must be a chain otherwise an exception will be thrown.

Listing 3: Node Example

```
<node functional_type="FIREWALL" name="node1">
   <neighbour name="node2"/>
   <configuration ...>
      ....
   </configuration>
</node>
```

Functional Type

A Node can be a:

- **FIREWALL**
- **ENDHOST**
- ~~**ENDPOINT**~~
- **ANTISPAM**
- **CACHE**
- **DPI**
- **MAILCLIENT**
- **MAILSERVER**
- **NAT**
- **VPNACCESS**
- **VPNEXIT**
- **WEBCLIENT**
- **WEBSERVER**
- **FIELDMODIFIER**

☞ **Warning**: ENDPOINT is not implemented in Verifoo.

☞ **Warning**: You cannot have in the same graph more than one server or more than one client, or a server and a client that use different protocols. If you are not compliant with this condition an exception will be thrown.

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. (for further details, please refer to the Verigraph documentation)

Firewall

A Firewall Configuration contains a list of ACLs (elements). The ACL defines a tuple of source node and destination node that represents the connection that will be blocked.
Due to Verigraph Schema Design, it is necessary to provide at least 1 ACL. If you don't want to set an ACL, provide a configuration with dummy node name.

Listing 4: Firewall Configuration Example

```xml
<configuration description="A simple description" name="conf1">
  <firewall>
    <elements>
      <source>nodeC</source>
      <destination>nodeD</destination>
    </elements>
  </firewall>
</configuration>
```

Cache

A Cache Configuration contains a list of resources. A resource is a node, and cache must include all nodes behind the cache in the chain.
Remember: Cache needs the notion of internal and external networks.

Listing 5: Cache Configuration Example

```xml
<configuration description="A simple description" name="conf3">
  <cache>
    <resource>nodeA</resource>
    <resource>node1</resource>
  </cache>
</configuration>
```

NAT

A NAT Configuration contains a list of internal nodes. The source defines the list of nodes behind the NAT.

Listing 6: NAT Configuration Example

```xml
<configuration description="A simple description" name="conf4">
 <nat>
   <source>nodeA</source>
 </nat>
</configuration>
```

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 7: Cache Configuration Example

```xml
<configuration description="A simple description" name="conf2">
  <dpi>
    <notAllowed>SomeString</notAllowed>
  </dpi>
</configuration>
```

Antispam

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

Listing 8: Antispam Configuration Example

```xml
<configuration description="A simple description" name="conf5">
 <antispam>
   <source>nodeA</source>
 </antispam>
</configuration>
```

MailServer

A Mail Server Configuration contains the Mail Server names.

Listing 9: MailServer Configuration Example

```
<configuration description="A simple description" name="confB">
  <mailserver>
    <name>nodeB</name>
  </mailserver>
</configuration>
```

MailClient

A Mail Client Configuration contains the Mail Server name.

Listing 10: MailClient Configuration Example

```
<configuration description="A simple description" name="confB">
    <mailclient mailserver="nodeB"/>
</configuration>
```

WebServer

A Web Server Configuration contains the Web Server names.

Listing 11: WebServer Configuration Example

```
<configuration description="A simple description" name="confB">
  <webserver>
    <name>nodeB</name>
  </webserver>
</configuration>
```

WebClient

A Web Client Configuration contains the Web Server name.

Listing 12: Web Client Configuration Example

```
<configuration description="A simple description" name="confB">
    <webclient webserver="nodeB"/>
</configuration>
```

VpnAccess

A VpnAccess Configuration contains the VpnExit name.

Listing 13: VpnAccess Configuration Example

```xml
<configuration description="A simple description" name="conf1">
  <vpnaccess vpnexit="node2" />
</configuration>
```

VpnExit

A VpnExit Configuration contains the VpnAccess name.

Listing 14: Vpn Exit Configuration Example

```xml
<configuration description="A simple description" name="conf2">
  <vpnexit vpnaccess="node2"/>
</configuration>
```

EndHost

An EndHost Configuration contains a Packet Model.

Listing 15: End Host Configuration Example

```xml
<configuration description="A simple description" name="conf2">
  <endhost body="thisisarequest"/>
</configuration>
```

Capacity Definition

The Capacity Definition element is a list of CapacityForNode elements that contain the disk requirement of each node. It will be used by Verifoo as a constraint for the deployement. The CapacityForNode element is characterised by:

- A **Node** attribute that refers to the name of a Node element in a graph
- The **Capacity** that represents the disk requirement of the node

☞ **Warning**: If a node doesn't have a capacity associated, the web service infers that it is 0.

Listing 16: Capacity Definition Example

```xml
<CapacityDefinition>
    <CapacityForNode node="node1" capacity="10"/>
 </CapacityDefinition>
```

Property Definition

The Property Definition element is a list of properties that will be checked by Verifoo for a specific graph. For now, only the isolation property is supported by Verifoo. The Property is characterised by:

- A **Graph** attribute that represents the graph on which the property will be checked.
- A **Name** that represents the property that will be checked.
- The **isSat** attribute, imposed by the web service that represents the result of the check.

Listing 17: Property Definition Example

```xml
<PropertyDefinition>
   <Property graph="0" name="IsolationProperty" isSat="true"/>
 </PropertyDefinition>
```

Host

An host is a physical machine present in the network infrastructure. An host is characterised by:

- A *Unique* **Name**
- A **Type** to distinguish client and server from middleboxes
- The **Disk Storage** available on the host
- The **Active** attribute, imposed by the web service. It's a boolean and it assumes a true value if at least one node has been deployed on the host.

After the Rest API has been called, the host will contain also a list of **NodeRef** sub-elements that represent the nodes that will be deployed on that host.

☞ **Warning**: When you try to deploy a graph on a physical network you need to indicate one special host on which the client node will be deployed, another one on which the server node will be deployed and at least another host on which the other nodes should be deployed, otherwise an exception will be thrown.

Listing 18: Hosts Example

```xml
<Hosts>
  <Host diskStorage="10" name="host1" type="CLIENT"/>
  <Host diskStorage="20" name="host2" type="MIDDLEBOX"/>
  <Host diskStorage="10" name="host3" type="SERVER"/>
 </Hosts>
```

Connection

A connection element represent the physical connection between two hosts. A connection is characterised by:

- A **Source**
- A **Destination**
- The **avgLatency** that represent the average latency on the physical link between the source and the destination.

Listing 19: Connections Example

```
<Connections>
 <Connection sourceHost="host1" destHost="host2" avgLatency ="-1"/>
 <Connection sourceHost="host1" destHost="host3" avgLatency ="-10"/>
</Connections>
```

Parsing String

It's the raw output of Verifoo execution (*model.toString()*). It is used only by the converter web service.

Listing 20: An extract of ParsingString Example

```
<ParsingString>
...
(define-fun check_isolation_n_0_nodeA_nodeB () Node
  node5)
(define-fun integer_host1 () Int
  1)
(define-fun node3@host7 () Bool
  false)
(define-fun node3@host2 () Bool
  true)
  ....
</ParsingString>
```

2.2 Error XML Schema

Listing 21: Error XML Example

```
<ApplicationError type="InvalidNodeChain" message="Nodes must be in a chain
    ↪ "/>
```

ApplicationError

ApplicationError is the root element of this XML schema, it must contain the following attributes:

- **type**
- **message**

Type

It defines the type of error that has occured, it can be:

- **XMLValidationError** The provided XML is invalid.
- **InvalidServerClientConf** The number of server or of client is invalid.
- **InvalidNodeChain** The service chain provided is not a chain.
- **PHYClientServerNotConnected** There aren't a connection between physical Client and Server.
- **InvalidPHYServerClientConf** The provided Hosts configuration for physical Client and Server is invalid.
- **NoMiddleHostDefined** In the Hosts configuration there aren't middle host.
- **InvalidNodeConfiguration** The configuration of a node mismatch with the node type.
- **InvalidVPNConfiguration** The VPN configuration is invalid.
- **InvalidParsingString** The Parsing String of Z3 Output is invalid.
- **InternalServerError** The service is unavailable.

Message

An human-readable error message.

2.3 Hyperlinks XML Schema

Listing 22: Error XML Example

```xml
<Hyperlinks>
   <Link rel="self" href="http://localhost:8080/verifoo/rest/" type="application/xml" method="GET" />
   <Link rel="deployment" href="http://localhost:8080/verifoo/rest/deployment" type="application/xml" method="POST" />
   <Link rel="converter" href="http://localhost:8080/verifoo/rest/converter" type="application/xml" method="POST" />
   <Link rel="log" href="http://localhost:8080/verifoo/rest/log" type="text/html" method="GET" />
 </Hyperlinks>
```

Hyperlinks

Hyperlinks is the root element of this XML schema and it's a list of **link** elements that represent link to the other resources. The **link** element has the following attributes:

- **rel** expresses the type of the relationship of the resource
- **href** is an HTTP link to the resource
- **type** indicates the content type that a request to that resource should have
- **method** indicates the required HTTP method

3 Rest API Description

3.1 Service Design

Resources Design

Resources	URLs	XML Repr	Meaning
ROOT	/	Hyperlinks	XML file with the hyperlinks to the other resources
deployment	/deployment	NFV	XML file with integrated deployment information
converter	/converter	NFV	XML file with integrated deployment information
log	/log		A limited portion of the log

Operation Design

Resources	Method	Query Params	Req. body	Status	Resp.body	Meaning
ROOT	GET			200	Hyperlinks	OK
deployment	POST	complete:boolean	NFV	200 400	NFV ApplicationError	OK Bad Reques
converter	complete:boolean POST		NFV	200 400	NFV ApplicationError	OK Bad Reques
log	GET			200	HTML	OK

The query parameter for the two POST operations, specify if the reply will be the complete version or the shorter one. In the latter, the hosts that aren't active after the deployment are omitted from the XML (also their connections are omitted).

3.2 API Description

Deployment API

This is the main API for the Verifoo Web Service. It provides, for each graph, the verification of the validity of the network model and the optimised deployment on the hosts.
Example request

- **POST** http://localhost:8080/verifoo/rest/deployment
- Accept: **APPLICATION_XML**;
- Content: XML file with the physical topology and the desired service chain (in an NFV element).

Example response

- 200: **OK**
- Content-Type: **APPLICATION_XML**;
- Content: XML file with integrated deployment information (in an NFV element).

Error Response

- 400: **BAD_REQUEST**
- Content-Type: **APPLICATION_XML**;
- Content: XML file with an ApplicationError element that specifies the type of error.

Converter API

This API provides a converter for Verifoo output. The **<parsingstring>** element in the XML file is filled with the Verifoo output.
Example request

- **POST** http://localhost:8080/verifoo/rest/converter
- Accept: **APPLICATION_XML**;
- Content: XML file with the physical topology, the desired service chain and the output model provided by Verifoo in the Parsing String element

Example response

- 200: **OK**
- Content-Type: **APPLICATION_XML**;
- Content: The same XML file recived as input with integrated deployment information.

Error Response

- 400: **BAD_REQUEST**
- Content-Type: **APPLICATION_XML**;
- Content: XML file with an ApplicationError element that specifies the type of error.

Log API

This API provide a convenient way for accessing the log of log4j2 for debugging purposes.
Example request

- **GET** http://localhost:8080/verifoo/rest/log

Example response

- 200: **OK**
- Content-Type: **TEXT_HTML**;