OSS Interface for HP Service Activator

Paulo Jorge Pontes Martins

Thesis to obtain the Master of Science Degree in

Telecommunications and Informatics Engineering

Supervisors: Prof. Rui António dos Santos Cruz
Eng. João Manuel Malveiro Valentim

Examination Committee

Chairperson: Prof. Paulo Jorge Pires Ferreira
Supervisor: Prof. Rui António dos Santos Cruz
Member of the Committee: Prof. Luís Manuel Antunes Veiga

May 2015
Acknowledgments

I would like to thank my parents Anabela Fernandes Pontes and Ilidio de Jesus Martins for their friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not be possible.

I would also like to acknowledge my dissertation supervisors Prof. Rui Cruz and João Valentim for their insight, support, encouragement and sharing of knowledge that has made this Thesis possible.

Last but not least, to all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life. Thank you.

To each and every one of you – Thank you.
Abstract

Communications Service Providers (CSPs) are taking too much time to deploy new services, mostly due to its hard network infrastructure, which leads to an increasing demand for more dynamic and scalable networks. Hewlett-Packard (HP) and its product HP Service Activator (HPSA) are a reference on the automation of Operations Support Systems (OSS) fulfillment for CSPs, working as an abstraction layer between the network and the rest of the OSS. Network Configuration Protocol (NETCONF) is the most recent network management protocol standardized by Internet Engineering Task Force (IETF) and one of the main growing standards in the network management market. This project presents an analysis to the most common used network management protocols used by CSPs for services provisioning and activation and implements a solution which enables CSPs to use HPSA with a NETCONF south-bound interface, supporting all the operation defined in the standard and the possibility of concurrently manage more than one device. The final tests confirm not only the usability of this solution to interact with NETCONF enabled network devices but also the advantages of using this plug-in instead of other available solutions.

Keywords

NETCONF; HP Service Activator; Communications Service Providers; Network Management; Activation; Provisioning
Resumo

Os Operadores de Telecomunicações estão demorando demasiado tempo para lançar novos serviços e parte do problema deve-se ao facto de a infraestrutura de rede ser demasiado rígida o que leva portanto a um aumento da procura por redes mais dinâmicas e escaláveis. A HP e o seu HPSA são atualmente uma referência ao nível de automatização de processos de suporte a operações, funcionando como uma camada entre a infraestrutura de rede e o resto dos processos de OSS. O NETCONF é o mais recente protocolo de gestão de redes aceite como standard pelo IETF e um dos com maior crescimento no mercado de gestão de redes. Este trabalho apresenta uma análise aos protocolos de gestão de redes mais usados pelos Operadores de Telecomunicações para provisionamento e ativação de serviços e implementa uma solução que permite aos Operadores de Telecomunicações utilizarem o HPSA com NETCONF como interface de gestão de rede, suportando todas as operações definidas pelo standard e apresentando também a possibilidade de gerir concorrentemente mais do que um dispositivo na rede. Os testes finais comprovam não só a usabilidade desta solução para interagir com dispositivos de rede que suportem NETCONF, mas também a vantagem de utilizar este plug-in em detrimento de outras soluções disponíveis.

Palavras Chave

NETCONF; HP Service Activator; Operadores de Telecomunicações; Gestão de Redes; Ativação; Provisionamento
# Contents

## 1 Introduction
- 1.1 Objectives and Contributions ............................................. 3
- 1.2 Document Layout ................................................................ 4

## 2 Background
- 2.1 eTOM – Enhanced Telecom Operations Map .............................. 7
- 2.2 Network Management ............................................................. 9
  - 2.2.1 CLI – Command Line Interface ......................................... 10
  - 2.2.2 SNMP – Simple Network Management Protocol .................. 10
  - 2.2.3 COPS – Common Open Policy Service ............................... 10
  - 2.2.4 CIM – Common Information Model .................................... 10
  - 2.2.5 NETCONF ................................................................. 11
    - 2.2.5.A YANG ................................................................. 14
    - 2.2.5.B Current Implementations ............................................ 15
    - 2.2.5.C NETCONF Evolution ................................................. 16
  - 2.2.6 Critical Analysis ............................................................ 16
- 2.3 HP OSS Fulfillment ............................................................... 16
  - 2.3.1 HP Service Activator ...................................................... 18
    - 2.3.1.A Workflows ............................................................ 19
    - 2.3.1.B HPXA Plug-ins ....................................................... 21
    - 2.3.1.C GenericCLI Plug-in ................................................ 22
    - 2.3.1.D GenericHTTP Plug-in ............................................. 23
    - 2.3.1.E Inventory ................................................................ 23
    - 2.3.1.F Other Activation Tools .............................................. 24

## 3 GenericNETCONF design, development and implementation ......... 25
- 3.1 Design Requirements ............................................................ 27
- 3.2 GenericNETCONF Architecture ............................................. 28
- 3.3 Development Environment .................................................... 29
# List of Figures

2.1 eTOM Level-0 view of Level-1 .................................................. 8  
2.2 NETCONF Conceptual Layers ..................................................... 12  
2.3 HP OSS Fulfillment [1] ............................................................... 17  
2.4 HPSA Order Fulfillment life-cycle ........................................... 18  
2.5 HPSA Structure ................................................................. 19  
2.6 Typical HPSA Workflow ....................................................... 20  

3.1 GenericNETCONF as a southbound interface of HPSA ................. 27  
3.2 GenericNETCONF layered architecture .................................... 29  
3.3 Execution of a get operation using NETCONF Plug-in .................. 35  

4.1 Test Environment ............................................................... 41  
4.2 Test Environment HPSA Inventory Beans ................................. 43  
4.3 Test Environment HPSA Inventory Tree .................................. 44  
4.4 Test Environment HPSA Enable Turing Workflow ....................... 44
List of Tables

2.1 NETCONF base capabilities ...................................................... 13
2.2 NETCONF base operations ...................................................... 14

3.1 Serial execution mode. ............................................................ 35
3.2 Parallel execution mode. ......................................................... 36
3.3 Atomic Serial execution mode. .................................................. 36
3.4 Atomic Parallel execution mode ............................................... 37

B.1 JUnit battery tests used to evaluate GenericNETCONF functionality .......... 62
# Listings

2.1 RPC request and reply messages ................................. 13  
2.2 Generic CLI telnet connection Template .......................... 23  
3.1 GenericNETCONF par.xml configuration file .................... 30  
3.2 Device connection information ................................. 31  
3.3 Get operation reply ............................................ 32  
3.4 Get-Config operation reply .................................... 32  
3.5 Get operation with output variable as "output1" ............ 34  
3.6 Device operations .............................................. 34  
4.1 Get netopeer modules configuration .......................... 42  
4.2 Enable the Turing-Module in Netopeer’s candidate configuration using GenericCLI .. 45  
4.3 Enable the Turing-Module in Netopeer’s candidate configuration using GenericNETCONF 45  
A.1 Schema is used to validate the XML specification received by GenericNETCONF .... 57
Acronyms

API Application Program Interface
BSS Business Support Systems
CIM Common Information Model
CLI Command Line Interface
COPS Common Open Policy Service
COPS-PR Common Open Policy Service Usage for Policy Provisioning
CRM Customer Relationship Management
CSP Communications Service Provider
CRM Customer Relationship Management
DMTF Distributed Management Task Force
eTOM Enhanced Telecom Operations Map
ESP Evolved Services Platform
FAB Fulfillment, Assurance and Billing
GUI Graphical user interface
HP Hewlett-Packard
HPSA HP Service Activator
HTTP Hypertext Transfer Protocol
HTTPS HyperText Transfer Protocol Secure
IAB Internet Architecture Board

xv
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IPTV</td>
<td>Internet Protocol television</td>
</tr>
<tr>
<td>IT</td>
<td>Information technology</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunication Union</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>JNC</td>
<td>Java NETCONF Client</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>MIB</td>
<td>Management Information Base</td>
</tr>
<tr>
<td>NETCONF</td>
<td>Network Configuration Protocol</td>
</tr>
<tr>
<td>NCS</td>
<td>Network Control System</td>
</tr>
<tr>
<td>NFV</td>
<td>Network Functions Virtualization</td>
</tr>
<tr>
<td>OSS</td>
<td>Operations Support Systems</td>
</tr>
<tr>
<td>OTT</td>
<td>Over-the-top content</td>
</tr>
<tr>
<td>PIX</td>
<td>Private Internet eXchange</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality-of-service</td>
</tr>
<tr>
<td>regex</td>
<td>regular expression</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>RSVP</td>
<td>Resource reSerVation Protocol</td>
</tr>
<tr>
<td>SMI</td>
<td>Structure of Management Information</td>
</tr>
<tr>
<td>SNMP</td>
<td>Simple Network Management Protocol</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SDN</td>
<td>Software Defined Networks</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>TCO</td>
<td>Total Cost of Ownership</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>TMF</td>
<td>TeleManagement Forum</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>WBEM</td>
<td>Web-Based Enterprise Management</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XSD</td>
<td>XML Schema Definition</td>
</tr>
<tr>
<td>YIN</td>
<td>YANG Independent Notation</td>
</tr>
</tbody>
</table>
1 Introduction

Contents

1.1 Objectives and Contributions .............................................. 3
1.2 Document Layout ............................................................. 4
The Communications Industry is facing a major challenge where the only constant seems to be the “change”. Communications Service Provider (CSP) all over the world are facing great challenges to remain profitable, to meet the high customer expectations and to confront the fierce competition. 

Well defined business processes and the right automation tools for these processes have been an important factor on who succeeds in this kind of market. For this reason, the TeleManagement Forum (TMF) created a business framework [2], identifying the required and/or recommended business and operations processes to successfully run a “services-based” business.

In the Operations Support Systems (OSS) “Services Fulfillment” area of TMF’s business framework, the need to manage network and service platforms or equipments has generated significant discussion in entities like Internet Engineering Task Force (IETF) and Distributed Management Task Force (DMTF) which are responsible for creating standards for the related technologies. The evolution of the services network characteristics, such as its dimension, the number of devices, the wide range of different suppliers and even the growing diversity of services, have changed the management requirements and consequently created a demand for new technologies to manage those networks [3].

Over the years, several problems related with the lack of security and scalability to manage large infrastructures and the huge volume of information were pointed as weaknesses to the most commonly used Network Management protocols, such as Command Line Interface (CLI) and Simple Network Management Protocol (SNMP) [4]. In order to overcome the current problems the IETF is developing a new technology named Network Configuration Protocol (NETCONF) which uses Extensible Markup Language (XML) and Remote Procedure Call (RPC) to wrap its protocol messages, requiring transport protocols able to ensure authentication, integrity and confidentiality.

Hewlett-Packard (HP), a lead supplier in automated solutions for OSS Fulfillment [5], deploys HP Service Activator (HPSA), the solution responsible for services Provisioning and Activation. HPSA provides CLI and Hypertext Transfer Protocol (HTTP), two generic plug-ins to communicate with network elements in order to edit their configuration. These plug-ins are however “proprietary” in terms of the “methods/commands” used for the Provisioning and Activation processes.

1.1 Objectives and Contributions

Everyday, new Cloud, Mobile, Internet Protocol television (IPTV) or even Over-the-top content (OTT) services appear in the Information technology (IT) world, however, the CSPs processes used to configure network infrastructures are not as fast or error free as they should in order to keep up with this evolution. Hence the importance of using the right tools or protocols.

This work comes from the need and the demand from CSPs to use NETCONF as a network configuration protocol for their Activation and Provisioning services. That demand reached HPSA’s devel-
opment team, who felt the need to support that protocol in their Provisioning and Activation tools. The contribution of this work is two-fold. First, it contains an analysis of the current state of art of services Provisioning and Activation processes in CSPs, including their positioning in the standard structure of a CSP and the main tools and network management protocols used for that purpose. Second, it presents a tested solution for CSPs, enabling them to use NETCONF as a network management protocol for services Activation which, taking into account the age of the NETCONF standard – RFC 4741 [6] – and the currently available tools, is something not very common, but something with a growing hype in research and in industry.

The objective of this work is to integrate NETCONF and HPSA using the tools already provided by HPSA for its customization. This document describes the importance of HPSA in the context of the current CSPs business model, the most relevant management protocols being used and why NETCONF is a suitable solution for HPSA. Therefore, and with that purpose, a new HPSA Plug-in was designed, implemented and tested, following standard integration, conformance testing suites and methodologies from HP. The design and development of the prototype is described in detail in this document, as well as the description of the implemented solution and its evaluation.

1.2 Document Layout

In this document Chapter 2 describes the necessary background information to provide an understanding of the HPSA positioning in the CSP’s business model, the TMF’s Enhanced Telecom Operations Map (eTOM) reference framework, the most relevant network management protocols and their limitations, and how the new NETCONF standard is able to overcome those limitations. It also provides an overview of the currently implemented NETCONF solutions and a brief market analysis of the tools available for OSS Activation and Provisioning.

Chapter 3 presents the design and architecture of a solution based on NETCONF’s requirements using what HPSA offers as customization tools to integrate new network management protocols, and details its implementation. Chapter 4 describes the methods used to evaluate the solution and the results obtained. Finally, Chapter 5 concludes the document and presents some limitations of the current solution and presents perspectives of future work to improve it.
2 Background

Contents

2.1 eTOM – Enhanced Telecom Operations Map ........................................... 7
2.2 Network Management ................................................................. 9
2.3 HP OSS Fulfillment ................................................................. 16
To be able to offer differentiated and attractive services while ensuring high service quality to their
customers, CSPs need effective Operations Fulfillment process that are able to support the deployment
of more, and more demanding services. If those services do not meet the customer’s expectations, the
CSP is doomed to fail.

The OSS is responsible for eliminating process’s redundancies and automating operational pro-
cesses, and as a result, it leads to big improvements in structure costs and a major decrease in time-
to-market [1]. With an effective OSS it is possible to deploy new and more demanding services and
capabilities, so that CSPs can easily expand their business.

This section presents an analysis to CSPs structure and processes with more emphasis on Provi-
sioning and Activation processes, namely the most common used tools and standards. It also presents
a more profound analysis to NETCONF, how it appeared and its role in the CSPs market, and also a
profound analysis to HPSA, the tool created by HP to automate services Provisioning and Activation.

2.1 eTOM – Enhanced Telecom Operations Map

In 2003 the TeleManagement Forum (TMF) released the Enhanced Telecom Operations Map (eTOM)
Business Process Framework as an International Telecommunication Union (ITU) standard [2], and
even though it is not mandatory but a recommendation, it has been adopted by most CSPs [7]. The
eTOM identifies the required business processes to run a service-focused business, and divides those
processes recursively in sub-processes, until it is clearly distinguishable the “product management”
aspects from the “network operations” aspects.

The initial Level-0 process groupings view of the framework defines a structuring of the overall en-
terprise into the three major areas of Strategy, Infrastructure and Product, Operations and Enterprise
Management, as illustrated in Figure 2.1.

Strategy, Infrastructure and Product: Processes to develop the company strategy, build infrastruc-
ture, develop and manage products, and develop and manage the Supply Chain. Covers planning
and lifecycle management

Operations: Processes necessary to support customer operations and management. These processes
are more related with direct operations with the customer.

Enterprise Management: Usually considered the Corporate functions or processes, it has synergies
with almost every other process in a company, whether product-, operational- or infrastructure-
related.

Operations is decomposed into four “vertical” and four “horizontal” Level-1 processes:
**Fulfillment:** Responsible for providing the customers with their product requests in a timely and correct manner.

**Assurance:** Ensures that the provided services meet the agreed Service Level Agreement (SLA) and Quality-of-service (QoS) performance levels by executing proactive and reactive maintenance activities. It is continuously monitoring the performance of the network to protectively detect failures.

**Billing:** Process responsible for producing timely and accurate bills.

**Operations Support and Readiness:** Provides management, logistics and administrative support to the Fulfillment, Assurance and Billing (FAB) processes.

**Customer Relationship Management:** Has the necessary customer's information to personalize and customize the services delivery as well as identify new opportunities to increase the customer engagement with the enterprise.

**Service Management and Operations:** More focused on the knowledge about the existing services and how to manage them.
Resource Management and Operations: Is responsible for the information about the existing resources and for managing them.

Supplier/Partner Relationship Management: Closely aligned with a Supplier’s/Partner’s Customer Relationship Management (CRM) and is responsible for issuing purchase orders and track them through to delivery and handling. It is also responsible for the validation of all the necessary bills and authorize the payments.

2.2 Network Management

Despite the vast range of network management standards and protocols, most of the network management systems are still very dependent on the human interaction. A large number of Router vendors have their own configuration languages and most of them are configured through CLI or Web-based technologies. It is not an issue using manual configuration, if a CSP intends to automate the configuration processes, but it might be a problem if no automation is considered, since manual configurations can lead to errors, jeopardizing the proper functioning of the entire network.

For network management, at least three main components must be taken into consideration: a manager, a managed object and a network management protocol. The manager consists of the network administrator node/platform or facilities which generates commands and receives notifications. The managed object is a network device, including its software, and is configured by the manager. The network management protocol is used to exchange management information between the manager and the managed objects.

Over the years the ITU, IETF and DMTF “standardization” bodies, have been working on new network management standards and protocols.

For a long time, several operators had the opinion that the developments made in IETF were not taking into account the operator’s needs, especially for configuration management. Given this generalized opinion, several meetings were organized to discuss the Requirements for Configuration Management of IP-based Networks [8].

On June 4 2002, some members of the Internet Architecture Board (IAB) held a workshop where operators and protocol developers had the chance to exchange their opinions about the IETF focus on future work regarding network management.

That workshop had two main goals: Name a list of relevant network management technologies and identify their strengths and weaknesses; Understand the operator’s needs based on the already existing solutions for network management.

The outputs of that workshop can be found on RFC 3535 [9], where the main existent network management technologies identified were SNMP [4], Common Open Policy Service (COPS) [10] and
Common Information Model (CIM). Its analysis originated a set of requirements which were later used as a base for the development of NETCONF. These technologies are detailed in the following sections.

2.2.1 CLI – Command Line Interface

CLI was created to facilitate the interaction of human administrators with network equipments. CLI is based, as its name implies, on a “command line” interface typically used in operating systems such as UNIX. It is still the most used method to direct/local network element configurations, yet it has some problems that affect operators performance, when considered in large automated operations. It requires an extra effort by operators to learn different commands for different devices, lacks a formal description language to define the properties of the programmatic interface and it does not have any structured error responses [11].

2.2.2 SNMP – Simple Network Management Protocol

The SNMP was proposed in 1990 as a “simple” protocol that implements communication between a management console and the managed object. Since there are plenty of implementations using this technology, its characteristics and limitations are well known. It is widely used for the monitoring of fault and performance data and because of its stateless nature, it works well to poll basic parameters of a device, such as checking its operational liveliness, and to detect discontinuities in counters.

There are many well defined Management Information Bases (MIBs) developed by network device vendors to manage their products, yet, not all MIB modules possess writable objects which can be used for configuration. As such, SNMP although developed for system-to-system interaction is mainly used for monitoring.

2.2.3 COPS – Common Open Policy Service

COPS was developed by the IETF Resource reSerVation Protocol (RSVP) Admission Policy Working Group, to provide a Policy-Based Network Management approach. Common Open Policy Service Usage for Policy Provisioning is an extension to COPS to provide “policy provisioning”, so that a client-server system can be used to push policy-based data to the network devices.

2.2.4 CIM – Common Information Model

DMTF was created in 1990 to develop a new information model common to most management systems. CIM is an object oriented model to represent management systems information in a standardized way. A particular well known implementation of CIM is Web-Based Enterprise Management (WBEM),
created in 1996 with the support of several companies. It encodes the management information in XML and transports the management information over HTTP.

The main problems with CIM are related with its interoperability with other CIM implementations and its limited number of deployments as an interface between management systems and network devices.

### 2.2.5 NETCONF

NETCONF tries to solve the limitations found with the other above mentioned protocols, by addressing the operator’s necessities. Some of the key requirements that a new network management solution should fulfill in order to meet the operator’s needs are the following [12]:

1. There must be a clear distinction between configuration data and operational data. Operational data is the additional data on a system that is not configuration data such as read-only status information and collected statistics.

2. It must provide primitives to prevent errors due to concurrent configuration changes. Either a lock or a conflict resolution mechanism are necessary.

3. There should exist primitives to apply configuration changes to a set of network elements in a robust and transaction-oriented way.

4. There must be a way to distinguish between several configurations, and devices should be able to hold multiple configurations.

5. There should be a distinction between candidate configuration and running configuration.

6. It should be clear about the persistence of configuration changes.

7. Must report configuration change events.

8. Full configuration dump and a full configuration restore must be supported appropriately.

9. A configuration management protocol must represent configuration state and operational state in a form enabling the use of existing tools for comparison, conversion, and versioning.

NETCONF was first published as a Standard on December 2006 in RFC 4741 [6], by the NETCONF Working Group within the IETF, being later overwritten in RFC 6241 [13]. As depicted in Figure 2.2, NETCONF presents a four layer architecture, where each layer plays a very specific function:

1. The **Transport** layer provides communication between the client (manager) and the server (managed object)
2. The **Messages** layer defines the reply-request mechanism to encode RPCs and Notification messages.

3. The **Operations** Layer defines the set of operations encoded in XML and invoked as RPC methods.

4. The **Content** layer declares the data exchange in NETCONF operations.

![Figure 2.2: NETCONF Conceptual Layers](image)

NETCONF is connection-oriented, meaning that it requires a persistent connection between the pairs. The connection must grant authentication, integrity and confidentiality, and so NETCONF delegate to the transport layer the responsibility of granting the appropriate levels of security and confidentiality. Even though NETCONF is defined as being session-layer and transport independent, it is mandatory to any implementation the support of the Secure Shell (SSH) connection using the IANA-assigned Transmission Control Protocol (TCP) port 830 and the subsystem `netconf` as illustrated in the following example [14].

```
[user@client]$ ssh -s server.example.org -p 830 netconf
```

It is however possible that during the session both peers can negotiate any other transport protocol such as Transport Layer Security (TLS) [15], Simple Object Access Protocol (SOAP) [16] or BEEP [17].
The NETCONF communication model is based on RPC. As illustrated in Listing 2.1, both peers use the elements `<rpc>` to frame the requests sent by the client to the server and `<rpc-reply>` to frame the server’s answer. The `<rpc-reply>` may contain the `<ok>` element, informing that no error occurred during the execution of that request or an `<rpc-error>` with a set of information about the detailed errors detected.

Listing 2.1: RPC request and reply messages

```xml
<!-- request -->
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get/>
</rpc>

<!-- response -->
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:ex="http://example.net/content/1.0" ex:user-id="fred">
  <data>
    <!-- response contents here... -->
  </data>
</rpc-reply>
```

Every NETCONF implementation must define its Capabilities. A Capability is identified by a unique Uniform Resource Identifier (URI) and allows the clients to discover the set of operations supported by the servers allowing the client to adjust itself to exploit all the resources exposed by the devices. During the session establishment, each peer (client and server) sends a `<hello>` element with the list of capabilities. Table 2.1 lists the main capabilities defined in RFC 6241 [13] and RFC 4741 [6]. There are more capabilities, but for the purpose of HPSA Provisioning, and in the scope of this work, the focus will be on the capabilities identified in the main RFCs of NETCONF, more specifically, the ones related with configuration data.

Table 2.1: NETCONF base capabilities

<table>
<thead>
<tr>
<th>Capability</th>
<th>Capability Identifier (URI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rollback-on-error:1.0</td>
<td>urn:ietf:params:netconf:capability:rollback-on-error:1.0 [15]</td>
</tr>
<tr>
<td>validate:1.0</td>
<td>urn:ietf:params:netconf:capability:validate:1.0:1.0 [6]</td>
</tr>
<tr>
<td>startup:1.0</td>
<td>urn:ietf:params:netconf:capability:startup:1.0 [13]</td>
</tr>
<tr>
<td>base:1.0</td>
<td>urn:ietf:params:netconf:base:1.0 [6,13]</td>
</tr>
</tbody>
</table>

Capabilities are advertised in messages sent by each peer during session establishment. When the NETCONF session is opened, each peer (both client and server) must send an `<hello>` element containing a list of that peer’s capabilities. Each peer must send at least the `:base:1.0` capability. A peer may include capabilities for previous NETCONF versions, to indicate that it supports multiple protocol versions. Table 2.2 presents the operations defined in the `:base` capability.
### Table 2.2: NETCONF base operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>close-session</td>
<td>.base</td>
</tr>
<tr>
<td>commit</td>
<td>.base and .candidate</td>
</tr>
<tr>
<td>copy-config</td>
<td>.base</td>
</tr>
<tr>
<td>create-subscription</td>
<td>.notification</td>
</tr>
<tr>
<td>delete-config</td>
<td>.base</td>
</tr>
<tr>
<td>discard-changes</td>
<td>.base and .candidate</td>
</tr>
<tr>
<td>edit-config</td>
<td>.base</td>
</tr>
<tr>
<td>get</td>
<td>.base</td>
</tr>
<tr>
<td>get-config</td>
<td>.base</td>
</tr>
<tr>
<td>kill-session</td>
<td>.base</td>
</tr>
<tr>
<td>lock</td>
<td>.base</td>
</tr>
<tr>
<td>unlock</td>
<td>.base</td>
</tr>
<tr>
<td>validate</td>
<td>.base and .validate</td>
</tr>
</tbody>
</table>

#### 2.2.5.A YANG

NETCONF allows the access to native capabilities of devices in a network, defining methods to manipulate the configuration of those devices and to retrieve operational data about that device. YANG is a data modeling language used to model NETCONF’s content and operations layers and it is defined in RFC 6020 [18].

A YANG module works like a contract between the client and the server. After both peers agree which modules they will use, the client is aware of how to create valid information to be transmitted to the server and it understands the server’s replies. Using NETCONF and YANG together, standard modules may be defined, allowing the interoperability between devices from different manufacturers and allowing also those devices to expose their unique capabilities.

Compared with other widely used modeling languages, like XML schema or Structure of Management Information (SMI), YANG demonstrates clear advantages when used with NETCONF [19]. YANG is organized in a hierarchical tree and a modular structure to allow the integration between modules, making it a flexible language which eases the interaction of proprietary and standard data models.

YANG offers a wide range of default data types that are not present in other modeling languages, like “ipv4-address”, and the possibility to define new data types. Being YANG a NETCONF specific data modeling language, it takes in consideration semantics like notifications, error messages definition and lock mechanisms. This makes YANG a very straightforward manner of mapping NETCONF data models with the best approach.

YANG models can be translated to a XML equivalent format named YANG Independent Notation (YIN), allowing XML parsers to operate over those modules. The translation between YANG and YIN is reversible and lossless.
2.2.5.B Current Implementations

NETCONF is a recent protocol, yet it is being largely accepted by network administrators. It has the support of some of the world’s greatest network device vendors like **Cisco, Juniper, Ericsson** and **Oracle**, and there are already some open-source and proprietary implementations on the market.

**JNC - Java NETCONF Client:** Java NETCONF Client (JNC) [20] is a JAVA implementation of a NETCONF library which started being developed by Tail-f in the middle of 2012 and released as open-source at the end of 2012.

**EnSuite:** Yenca was one of the first NETCONF client solutions and it was the basis of the eldest NETCONF implementations, the EnSuite [21]. **EnSuite** is a framework which provides a web implementation of a NETCONF client and a server implementation in python.

**Yuma:** One of the most well known open-source solutions is **Yuma**. It is a product from **YumaWorks**, which includes both client and server applications. It was forked in 2012 and **YumaWorks** kept working only in **YumaPro**, their commercial solution.

**netconf4j:** There is also an Android NETCONF client solution named **netconf4j** and several other open-source implementations, most of them based on the previous version of NETCONF [6].

**libnetconf:** The **libnetconf** project is an open-source implementation of NETCONF already compatible with the RFC launched in 2011 [13], where the core components are written as a C library. Now, it is one of the most updated open-source implementations (its latest release was on November 2014), reason why it is used by many other implementations.

**netopeer:** An open-source implementation developed by **CESNET**, based in **libnetconf**. **Netopeer’s** manager implementation provides a set of tools, including a NETCONF CLI interface, which allows the interaction with NETCONF-enabled devices.

**NetconfX:** Is an implementation maintained by **CenteredLogic** and written in JAVA. It is a proprietary implementation, yet the source code is available for use, but not for redistribution being that it is under a GPL (GNU GPL v3) license.

**Tail-f:** The most well known proprietary solution using NETCONF nowadays is developed by **Tail-f Systems** and it presents several solutions that take advantage of NETCONF. **Tail-f** offers an application named Network Control System (NCS) which implements NETCONF and works as a client solution for network management, exposing an Application Program Interface (API) which facilitates the integration in Operations Support Systems (OSS)/Business Support Systems (BSS) solutions. **Tail-f** was bought by **Cisco Systems** on july 2014 to be part of their Evolved Services Platform (ESP), a solution to automate provisioning of physical and virtual networks.
2.2.5.C NETCONF Evolution

The demand for agility in services delivery of CSPs is driving the interest in Software Defined Networks (SDN) and Network Functions Virtualization (NFV) technologies [22], making them one the most discussed topics in the 2015 edition of the Mobile World Congress, which takes place every year in Barcelona, Spain.

NFV is a network architecture which proposes the virtualization of network functions that currently runs on commodity servers, something that was really difficult until now given the low processing power of existent CPUs and Network interfaces.

SDNs is a network architecture which separates the infrastructure layer from the controller layer which works as a mediator for applications who intend to interact with network devices and translates those application requests in network configuration. Controllers tell network devices how to behave.

NETCONF is one the main protocols proposed for the interaction either from SDN controllers to network devices, as from the application layer to the controller layer. Some well known SDN controllers, like Opendaylight and Ryu, support NETCONF as an interface. For both technologies SDN and NFV, NETCONF is one of the main configuration protocols being discussed and implemented, given its standardization for all different kind of vendors and implementations.

2.2.6 Critical Analysis

The main limitations regarding SNMP are related to scaling problems, since it does not deal very well with large amounts of data and it is not easy to differentiate configuration from operational data, making it hard to retrieve and playback configurations.

Some performance comparatives between SNMP and NETCONF show that NETCONF is definitely a better solution to deal with large amounts of data [23,24].

Although the work developed in this thesis is not focused on Common Open Policy Service Usage for Policy Provisioning (COPS-PR), since policies and QoS are not part of HPSA, yet, it is of relevance to refer that some evaluations considering network usage and protocol delay show that NETCONF may replace COPS-PR with possible advantages [25].

2.3 HP OSS Fulfillment

HP OSS Fulfillment software covers the fulfillment process stack defined by eTOM, including product catalog-based order management, activation of network and service infrastructure, and service and resource inventory management [26]. This stack (see Figure 2.3) is responsible for planning the future capacity of the network, configure the network to provide all the different services and plan and
implement the changes required in the network and services layer to support the services ordered by customers. The key components are Order management, Change and Configuration, Inventory and Activation.

Order management: It is responsible for automating the order lifecycle, including catalog management, order entry and order analytics.

Activation: Responsible for delivering, modifying or retiring a service. It automates the tasks required to configure and activate a service across the different servers, software and network infrastructure and includes adaptable Workflows, several network plug-ins and management tools.

Inventory: Includes a database of records used to manage and track all the physical and logical resources which provide services, while also documenting the network infrastructure and capabilities.

Change and Configuration: This area is responsible for supporting the change management processes in order to automate the change lifecycle making it easier to handle either planned and unplanned changes. It provides an integration between change and configuration management systems so that it is easier to predict the impact of a change in the network.

The fulfillment process can be easily represented by the five states in the diagram of Figure 2.4.

Each time there is an order, there is a verification if it is technically feasible to build the services comprising the product given the requested characteristics such as site address, etc. Passing this verification, the order is Checked.

After the verification, comes the Design. The product is decomposed according to the tree-structure defined in the Service Catalog and the resulting service tree persisted as an instance in Service Inven-
The shared infrastructure resources needed for the product are then **Reserved** in the Resource Inventory and associated with the instance in Service Inventory.

The order then needs to be **Provisioned**, that is, all network elements and other infrastructure resources which are affected according to the design of the service need to be configured to perform the service. This implies all needed logical resources — such as bandwidth, numbers/identifiers, registry entries and storage — being allocated in the infrastructure.

All the provisioned services may then be **Activated**, meaning that all necessary equipments must be configured to provide a specific service to a specific client, making it usable and subject to correct functioning of the infrastructure as monitored by Assurance subsystems. A similar process is followed in the case of a modification of an activated product.

A Product/Service can also be **Terminated**, meaning its reserved/allocated resources are released in the inventory and may be reused for new services.

### 2.3.1 HP Service Activator

HP Service Activator (HPSA) 6.1 is a customizable product from HP released in 2008 that performs tasks to activate services offered by providers of converged technologies and CSPs.

HP Service Activator (HPSA) is usually deployed by CSPs to automate and secure repetitive jobs, guaranteeing error-free processes and focusing on the processes designated in eTOM as Service Configuration and Activation and Resource Provisioning [27].

The revenue from the Activation was more than 450M US Dollars in 2009 and HPSA is one of the most used Provisioning and Activation solutions used in the market [5]. HPSA is recognized for its scalability and its user interface which eases the usability. The generic Workflow engine of HPSA can be used to model complex activation processes and its robustness makes it a high-reliable solution.
The main goal is to reduce the time-to-market of new services or products, to achieve an automated zero-touch activation and to reduce the Total Cost of Ownership (TCO). To ensure this, the Fulfillment process has to be as error free and as automated as possible, and so HPSA has a complex structure (Figure 2.5) with a set of components that communicate with each other.

![Figure 2.5: HPSA Structure](image)

HPSA's structure can be divided in two layers, plus a set of application tools that can be used to extend the framework capacity. The processing layer is the intelligence of the framework, responsible for managing complex activations, failures and errors, while the presentation layer provides network managers a way of interacting with the framework.

2.3.1.A Workflows

A workflow is a definition of an executable process. The definition is at a detailed algorithmic level, suitable for control of interactions with activation targets. Workflows are composed from a set of workflow nodes, which are executed by the workflow manager, one node at a time. In an actual workflow, each box, as illustrated in Figure 2.6, will require several workflow nodes, the number depending on the complexity of the details of the process. At the high-level, the process is typically as follows:
1. A customer order is received into a CRM system, where it is validated, approved and forwarded to the next level of processing. In a simple case it can go directly to Service Activation. In a more complex case, an order management process may be needed to decompose an order which is for a bundle of services and to separate the request for automatic activation, which can be handled by HPSA, from other tasks which must be performed manually.

2. The activation system based on HPSA receives the request for the activation of a service, maps to the appropriate workflow and starts an instance of the workflow as a job.

3. The workflow job inspects the parameter values from the request and calculates or gathers additional necessary parameter values. Those parameter values can be, for example, the selection of a device or port.

4. When all parameters are ready, the necessary activations are executed.
5. If the activation is successful, the newly activated service will be recorded in Service Inventory. If it fails, any reserved resources are again released.

6. A summary of the actions of the workflow, including service-id, main parameter values, success or failure, is recorded in HPSA’s audit trail.

7. A response message to be returned to the requester is prepared. Success or failure indication and any relevant parameters that were derived by the workflow are included in the message.

8. Finally, the response message is processed by the Order Handling system or CRM system. This is outside the scope of the activation system.

Currently, there are several business processing steps that can be automated using HPSA workflow nodes, namely, extracting data from an incoming XML message, calculating derived parameters, requesting and updating data from external repositories such as an inventory database, sending messages to external processes, waiting for input from a human operator or external process and perform hardware or software configuration according to gathered parameters. Each workflow node is implemented as a Java class, extending a base node class with specializations for process nodes, rule nodes and switch nodes.

### 2.3.1.B HPSA Plug-ins

A plug-in is a Java class that contains methods to perform configuration tasks related to a specific type of software or hardware component. A plug-in should be able to perform its operations on any component of that type as long as the target is reachable from the Service Activator subsystem and has the necessary prerequisites to enable the communication from Service Activator. Typically, this communication is established via Secure Shell, but other communication mechanisms are possible. Plug-ins provide at least an Atomic Task which is a java method that will be summoned from Activate nodes available in workflows. These atomic tasks are managed by a resource manager which is part of HPSA’s framework and which guarantees that every Atomic Task is executed atomically and reliably maintains the state of each transaction in order to recover transactions that were interrupted by a system failure.

The Atomic Tasks provided must be reversible and, for that purpose, the first parameter it receives is an operation parameter which takes either a DO_AND_CHECK or UNDO_AND_CHECK value used by the resource manager to manage the reversibility of a plug-in. Every task must return an ExecutionDescription, which is a java class provided by the resource manager library and indicates whether a plug-in executed successfully or not. If a plug–in fails its execution, either from internal causes or because it received an error from the managed device, the resource manager will call the provided Atomic Task
with UNDO_AND_CHECK as the first parameter, indicating that it must reverse the managed device to its initial state.

The way HPSA handles concurrency is by using a set of Compound Tasks which can be defined at the workflow level. Compound Tasks execute a set of defined Atomic Tasks atomically, meaning that if one fails, the resource manager will automatically execute the “UNDO_AND_CHECK” mode for all Atomic Tasks.

Three generic plug-ins are supplied as part of the HP Service Activator core product:

- generic CLI plug-in;
- generic HTTP plug-in;

These plug-ins do not include information about commands or HTTP messages to send or what to expect as responses.

2.3.1.C GenericCLI Plug-in

The CLI plug-in is controlled by a XML dialog control template. A dialog, for example, for a complex router device, may comprise a large number of commands, and so the control template may be quite large. The control document will contain information about how to establish a session with the device, a sequence of command-response exchanges, and finally how to terminate the session. The specification for each command response exchange includes the exact command line to be sent, the expected response and the patterns that allow recognition of possible error responses. Every dialog, even those with many commands, are executed as a single atomic task. In order to ensure clean failure of the task when an error response is received for a command, the CLI plug-in supports rollback of the command sequence. Rollback command sequences use rollback commands that can be specified for each command-response exchange that is part of the sequence.

The XML dialog control template typically includes a number of strings whose values must be sourced from variables of the workflow node that invokes the atomic task, representing items such as username and password for authentication of the session/request by the target, names of devices, ports, interfaces and other objects that exist and must be manipulated on the target. The way to handle this situation is to prepare a template document with replaceable placeholders for variable values, and to use a workflow node to substitute the actual values for the placeholders to obtain the final document to submit to the plug-in. In Listing 2.2 is an example of a XSD template used to generate the GenericCLI xmlSpec which receives as variables a username, a password and a session Timeout from a workflow node, in order to create a TELNET connection to a Cisco Private Internet eXchange (PIX) firewall.
Listing 2.2: Generic CLI telnet connection Template

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output doctype-system="CLIv4.dtd"/>
  <xsl:param name="pix_pswd"/>
  <xsl:param name="pix_enable_pswd"/>
  <xsl:param name="pix_timeout"/>
  <xsl:template match="/">
    <CLI>
      <Connect protocol="telnet">
        <Do timeout="{ $pix_timeout }" description="PIX device authentication failed.">
          <Confirm>
            <Pattern>PIX passwd: $</Pattern>
            <Command>
              <xsl:value-of select="$pix_pswd"/>
            </Command>
          </Confirm>
          <Error>PIX passwd: $</Error>
          <Prompt>&gt; $</Prompt>
        </Do>
        <Do description="PIX privileged (enable) mode authentication failed.">
          <Command>enable</Command>
          <Confirm>
            <Pattern>Password: $</Pattern>
            <Command>
              <xsl:value-of select="$pix_enable_pswd"/>
            </Command>
          </Confirm>
          <Error>Password: $</Error>
          <Error>usage:</Error>
          <Prompt># $</Prompt>
        </Do>
      </Connect>
    </CLI>
  </xsl:template>
</xsl:stylesheet>
```

2.3.1.D GenericHTTP Plug-in

The HTTP POST atomic task supported by the generic HTTP plug-in makes a single HTTP POST call to a specified target. It supports security features, such as HyperText Transfer Protocol Secure (HTTPS), with exchange of certificates and use of a proxy. The main parameter to achieve the intended effect is the HTTP POST message body, typically in SOAP format. This plug-in can be used when all necessary message formats are well known and simple enough for conveniently preparing templates for them and decoding the responses by parsing XML in the workflow.

2.3.1.E Inventory

The HPSA Inventory sub-system is a component of the framework for building a complete activation solution. All activation solutions require some repository in which to store details about resources in the management domain that are used for activations (switches, routers, available ports, and so on).
2.3.1.F Other Activation Tools

There is a whole range of tools targeting Provisioning and service Activation. Oracle’s ASAP, HPSA, Amdocs, Ericsson (former Telcordia), and Comptel together cover more than 50% of the market of Service Activation solutions integrated in an OSS Fulfillment solution, taking care of the Provisioning and Activation processes of the eTOM business model [5]. Most of those solutions implement CLI and none of these solutions actually has compatibility with NETCONF.

It can easily be concluded that the existing solutions implementing NETCONF for provisioning are either outdated, do not have a northbound interface making possible to integrate them in an OSS Fulfillment solution or do not have enough market presence to be reliable for large CSPs.
3

GenericNETCONF design, development and implementation

Contents

3.1 Design Requirements ......................................................... 27
3.2 GenericNETCONF Architecture ........................................... 28
3.3 Development Environment .................................................. 29
3.4 Implementation ............................................................... 30
This Project comes from the need from Service Providers to have in their activation platform the possibility of using a NETCONF interface. GenericNETCONF is a generic plug-in which enables HPSA with a NETCONF southbound interface as seen in Figure 3.1. It is designed as a plug-in for HPSA and as such its specifications related not only with the protocol, but also related to how a generic plug-in for HPSA should work.

![Figure 3.1: GenericNETCONF as a southbound interface of HPSA](image)

The architecture mentioned above was implemented in HP offices using only free or open source tools besides HPSA itself. This section describes the implemented solution, how it works and how it could be used.

### 3.1 Design Requirements

NETCONF is a connection-oriented protocol, meaning that one and only one session should be opened for each device. Given the requirements defined by IETF for NETCONF and the requirements that are usually set for an HPSA plug-in, a set of requirements are also defined for this plug-in:

1. Support all operations defined in RFC 6241
2. SSH as transport layer
3. Validate content/inputs for all supported operations
4. Support a concurrent interaction with one or more network devices

5. It must be generic, meaning that it shouldn’t be vendor specific and HPSA or allow only a set of available configurations.

6. Provide an interface/schema to be used by HPSA workflows.

7. Plug-in Tasks should be atomic

8. Atomic tasks should be reversible

9. Atomic tasks must take an initial parameter indicating whether this is the Do_AND_CHECK or the UNDO_AND_CHECK invocation of the task

Even though SSH is mandatory in all NETCONF implementations, as seen above, there are already other transport protocol definitions for NETCONF and its implementation must be easily extensible to support the implementation of new transport protocols. The same goes for Operations, RFC 6241 defines a set of operations, yet, NETCONF servers may extend its capabilities and therefore, the client implementation should also be extensible enough to support new operations and capabilities.

3.2 GenericNETCONF Architecture

In order to address the requirements and to make it as extensible as possible, the GenericNETCONF structure is based on the one defined by RFC 6241 for NETCONF implementations, following a four-layer architecture with some adaptations to work as a plug-in for HPSA as illustrated in Figure 3.2.

In order to make the plug-in as generic as possible, the content layer must be defined outside the plug-in and received as input. This gives network managers and HPSA workflows the power to adapt the content that is sent to the device accordingly to any network or non-network related variables. Every plug-in must provide at least an Atomic Task which will be the main interface from which the plug-in will be referenced. GenericNETCONF provides task_Netconf_Activate which receives the initial DO_AND_CHECK or UNDO_AND_CHECK parameter which is used by the resource manager to handle the plug-in reversibility. This task is also the interface used by HPSA to connect the content layer with the other existent layers.

The operations layer is divided in four main groups: Action, Rollback, Commit and Undo, allowing the plug-in to support concurrent activations when used together with the provided Execution Modes. For each operation there is a validation of its content which is done before execution. The operations are implemented using a command design pattern, which makes extending an operations very easily. Each operation must implement two methods, the validate(), which receives the content for that operation and
validates it, and the `execute()` method, which is the connection to the Messages and Transport layer to execute the operation on the device.

The Messages and Transport layers are managed by the open source implementation of JNC done by Tail-f. The library is used to manage the NETCONF sessions and to manage the capabilities' verification. JNC is the most updated open-source implementation of a NETCONF client done in JAVA and its architecture allows easily the implementation of new capabilities on the plug-in. Also, even though it provides by default only SSH on the transport layer, it provides an interface to extend the transport protocols available. The library also encapsulates the operations in RPC envelopes before it sends it to devices.

GenericNETCONF provides serial and parallel execution modes, making it able to support a concurrent interaction with one or more network devices. The set of devices to be managed as well as the execution mode are received as an argument in the task and the execution is managed by the plug-in’s ExecutionManager, which is responsible for executing all kind of operations for all devices, managing the success or failure of its execution and notifying HPSA of the success of its execution.

### 3.3 Development Environment

As explained above, HPSA plug-ins are Java classes that contain methods to perform configuration tasks related to a specific type of software or hardware component. Given this, most of this project was
developed in **JAVA**, being Eclipse the Integrated Development Environment (IDE) chosen to execute this project with Checkstyle plug-in installed to assure that correct Java coding style is used.

Since this project was developed for **HPSA**, the development environment was the required by **HPSA 6.1** for its execution. A Virtual Machine with 2GB of RAM, RedHat 6.4 x64 as Operative System, Eclipse Luna, Java™ SE Development Kit 6, Update 37, Oracle Database Express Edition 11g and **HPSA 6.1** was used for the development of this project.

As for external tools, **HPSA** provides a tool called ServiceBuilder as shown in Figure 2.5 to generate plug-in’s base structure. For version controlling an SVN repository provided by **HP** was used which was managed using TortoiseSVN.

### 3.4 Implementation

**HPSA** Plug-in’s base structure is well defined and can be easily generated with the provided ServiceBuilder. This base’s structure includes one Java class which extends the PARPlugin class provided by the Resource Manager and a par.xml configuration file as the one seen in Listing 3.1, which indicates the plug-in’s main class, its atomic tasks and any extra files and libs.

Listing 3.1: GenericNETCONF par.xml configuration file

```xml
<Plugin version="1.0">
    <Name>GenericNetconf</Name>
    <Version major="0" minor="8" revision="2" />
    <ClassName>com.hp.ov.activator.plugin.GenericNETCONF</ClassName>
    <NameSpace type="GLOBAL" />
    <DeploymentModel>ON_DEMAND</DeploymentModel>
    <Lock />
    <AtomicTasks>
        <Task execution="ON.LINE" exported="true">
            <Name>Netconf_Activate</Name>
            <Argument>xmlSpec</Argument>
        </Task>
    </AtomicTasks>
    <Files>
        <File file="schema.xsd" name="Schema">
            <Description>XML Schema describing the structure an contents of xmlSpec</Description>
        </File>
    </Files>
</Plugin>
```

30
<LIBs>
  <LIB name="ganymed-ssh2-build210.jar">
    <Description>SSH implementation library</Description>
  </LIB>
</LIBs>

GenericNETCONF provides one Atomic Task named task_Netconf_Activate. This task receives two parameters, the do parameter, which is mandatory on every plug-in and indicates if the plug-in should execute in DO_AND_CHECK mode or UNDO_AND_CHECK, and a xmlSpec with the plug-in's configuration and content.

After the plug-in finishes its execution, an ExecutorDescriptor is returned containing a majorCode which indicates if its execution was successful and without errors, a minorCode indicating if the network was maintained in a consistent or an inconsistent state and a stderr with the error description in case of error.

### 3.4.1 Interface

The xmlSpec must follow the schema defined in Appendix A and the input can be either the actual XML document or the path of a file containing the document in the form file:filepath. To validate and interpret the input, the plug-in contains a set of classes generated from the schema using JAXB, a tool provided with the Java Development Kit (JDK) to create a JAVA representation of either a XMLs or a XML Schema Definition (XSD) file. The xmlSpec contains the following information:

1. Connection information for the set of devices which will be configured.
2. Execution Mode to indicate how the plug-in should behave when activating multiple devices.
3. Specific Operations to configure each device and its content.
4. Specific Operations to execute if the configuration fails.
5. Specific Operations to execute if the configuration finishes successfully.

Given that NETCONF is a connection-oriented protocol, meaning that one and only one session should be opened for each device. The destination info is configured in the input XMLs specification as seen in Listing 3.2:

**Listing 3.2:** Device connection information
```
<device name="{$name}" host="{$host}" port="{$port}" user="{$user}" pass="{$password}">
...
</device>
```
After the session is created, it is kept open until there is a request to close it or the execution of all operations are finished.

### 3.4.2 Operations

Operations are implemented using a command design pattern. Every operation must implement an `Operation.java` abstract class and implement the `execute()` and `verify()` methods.

Currently, the base operations from table 2.2 are implemented in the plug-in with the exception for the create-subscription operation, since it is out of the scope of HPSA functions. They are:

**Get** Retrieves the running configuration and device state information as seen in Fig 3.3. Receives optionally as input a filter which can be either a subtree XML filter, defined in RFC 6241 [13] or an XPath filter. An example of a reply when executing a get operation in a NETCONF server, with a filter to retrieve statistical information relative to interface `eth0`, can be seen in the XML represented in Listing 3.3.

**Listing 3.3: Get operation reply**

```xml
<top xmlns="http://example.com/schema/1.2/stats">
  <interfaces>
    <interface>
      <ifName>eth0</ifName>
      <ifInOctets>45621</ifInOctets>
      <ifOutOctets>774344</ifOutOctets>
    </interface>
  </interfaces>
</top>
```

**Get-Config** Retrieves all or part of a specified configuration datastore. Receives optionally two parameters: a filter, such as get operation, and a parameter indicating the datasource to retrieve the configuration from. An example of a reply when executing a get operation in a NETCONF server, with a filter to retrieve configuration information only relative to interface Ethernet0/0, can be seen in XML represented in Listing 3.4.

**Listing 3.4: Get-Config operation reply**

```xml
<config>
  <top xmlns="http://example.com/schema/1.2/config">
    <interface>
      <name>Ethernet0/0</name>
      <mtu>1500</mtu>
    </interface>
  </top>
</config>
```

**Edit-Config** Loads all or part of a specified configuration to the specified target configuration datastore. Must receive at least two mandatory parameters. The first is the Target datastore with should be affected by this operation. The second parameter must be either a XML with the configuration to
be merged with the device’s existent configuration or an Uniform Resource Locator (URL) pointing to the source configuration.

**Copy-Config** Creates or replaces an entire configuration datastore with the contents of another complete configuration datastore. Must receive as input the Target and Source datastores which can be either a datastore existent in the target device, an URL pointing to the desired configuration or an actual XML configuration for the Source.

**Delete-config** Deletes a given configuration datastore received as Target input. The `<running>`, configuration datastore cannot be deleted.

**Lock** Allows the client to lock the entire configuration data store of a device lasting until either the lock is released or the NETCONF session closes. Must receive as input the target datastore from the device which will be locked.

**Unlock** Allows the client to release a previously locked datastore.

**Close-session** Closes the existent session.

**Kill-session** Forces the termination of a known NETCONF session. The session-id must be passed as input.

**discard-changes** Discards the existent configuration in candidate datastore and replaces it with the configuration existent in the running datastore.

**validate** Validates a given configuration. The configuration to be validated must be given as input and it should be either a datastore existent in the target device, an URL pointing to the desired configuration or an actual XML configuration.

A set of variables are available to customize how the plug-in treats each operation:

1. **name**: operation name.
2. **output**: for each operation, a variable can be set as the upload variable. The plug-in executes the operation and, if there is any output, it is uploaded to the defined variable. An example using the output variable can be seen in Listing 3.5.
3. **retry**: If set to true, the plug-in repeats the operation in case of failure. Default: false.
4. **numOfRetries**: Number of times the operation execution should be retried before it fails. Default: 3.
5. **timeBetweenRetries**: Time in seconds between retries. Default: 60.
6. **rollbackOnError**: If set to false, the following operation will be executed as if there was no failure in any operation. Default: true.

Listing 3.5: Get operation with output variable as "output1"

```xml
<operation name="get" var="output1">
  <filter type="subtree">
    <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
      <authentication/>
    </system>
  </filter>
</operation>
```

For each of these operations, the plug-in is able to verify if the required content was received as input and interact with any device which supports the :base, :candidate and :validate capabilities.

As defined in the architecture, to support concurrent configurations, GenericNETCONF supports the definition of four types of operations: Action, Commit, Rollback and Undo as seen in Listing 3.6.

Listing 3.6: Device operations

```xml
<device name="{$name}" host="{$host}" port="{$port}" user="{$user}" pass="{$password}"
  <action>...
  </action>
  <commit>...
  </commit>
  <rollback>...
  </rollback>
  <undo>...
  </undo>
</device>
```

Assuming that the plug-in receives "DO_AND_CHECK" as initial parameter, the Action group of operations are always executed until either all the operations are executed successfully or one of them fails. If all Action operations finish successfully, the plug-in will execute the commit operations for all managed devices independently, meaning that if the execution of one operation fails, the plug-in will execute all the following operations until it is finished. Rollback operations are executed when one Action operation fails, depending on the plug-in's Execution Mode. Undo operations are executed in reverse mode when the plug-in receives "UNDO_AND_CHECK" as the initial parameter from the resource manager.

### 3.4.3 Execution Modes

Any activation for a managed device is executed by the plug-in as seen in Figure 3.3. The replies received by the plug-in are interpreted in order to verify if it is an `<ok>`, a `<data>`, or an `<rpc-error>`, message. This way, the plug-in is able to handle error or return device state data to the workflow.

For different types of activation, it may be necessary for a workflow to interact in different ways with the network. GenericNETCONF takes advantage of the multiple concurrency for which the protocol was
designed while giving HPSA the control needed to always leave the network in a consistent or in a known state. To assure this, it provides four different execution modes:

**Serial (default)** is the simplest execution mode and follows a serial order as seen in Table 3.1. The operations for each device are executed independently and in order. Taking into consideration, for example, the configuration of two different devices, the operations for Device2 are all executed after the operations for Device1 are finished or one of them have failed, as represented in Table 3.1.

<table>
<thead>
<tr>
<th>Device 1</th>
<th>Device 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>Connect</td>
</tr>
<tr>
<td>Read</td>
<td>Read</td>
</tr>
<tr>
<td>Write</td>
<td>Write</td>
</tr>
<tr>
<td>Commit</td>
<td>Commit</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Disconnect</td>
</tr>
</tbody>
</table>

The operations run until one fails or all of them finish with success. If one operation fails, the plug-in will start the Rollback operations for that device and will not execute the following operations for the following devices, returning failure as the majorCode of the ExecutionDescriptor. If all operations complete with success, the Commit section is executed. In both cases, after one device is finished, the plug-in follows up with the configurations for the remaining devices. The ExecutionDescriptor will also contain a minorCode indicating that the network was left in a consistent or inconsistent state, dependent on the success of rollback operations.
**Parallel** The Parallel execution mode does not assure order in the configuration of each device as seen in Table 3.2. If an operation fails, the plug-in will execute Rollback operations for that device. The `ExecutionDescriptor` will always return success as the majorCode, yet if there is an error the minorCode indicates that the network was left in an inconsistent state and the error is returned in the stderr. The communication with each device is running in different threads. In the example of Table 3.2, Device 1 and Device 2 are configured in parallel.

Table 3.2: Parallel execution mode.

<table>
<thead>
<tr>
<th>Device 1</th>
<th>Device 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>Connect</td>
</tr>
<tr>
<td>Read</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>Write</td>
</tr>
<tr>
<td></td>
<td>Commit</td>
</tr>
<tr>
<td></td>
<td>Disconnect</td>
</tr>
<tr>
<td>Write</td>
<td></td>
</tr>
<tr>
<td>Commit</td>
<td></td>
</tr>
<tr>
<td>Disconnect</td>
<td></td>
</tr>
</tbody>
</table>

**Atomic_Serial** As in the Serial mode, the operations for each device run in a serial order until either one fails or all of them finish. The plug-in will start Rollback operations if one operation fails, or Commit operations if every operation is finished with success. Since the communication with all devices is happening at the same time, if one aborts, it does not affect any other device and the operations for all other devices will continue until they either abort or commit.

Table 3.3: Atomic Serial execution mode.

<table>
<thead>
<tr>
<th>Device 1</th>
<th>Device 2</th>
<th>Device 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>Connect</td>
<td>Connect</td>
</tr>
<tr>
<td>Read</td>
<td>Read</td>
<td>Read</td>
</tr>
<tr>
<td>Write</td>
<td>Write</td>
<td>Write</td>
</tr>
<tr>
<td>Vote Commit</td>
<td>Vote Commit</td>
<td>Vote Abort</td>
</tr>
<tr>
<td>Rollback</td>
<td>Rollback</td>
<td>Rollback</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Disconnect</td>
<td>Disconnect</td>
</tr>
</tbody>
</table>
This mode follows a kind of two-phase commit approach. The configuration of all devices is executed as in Serial mode, but the plug-in behaves like a controller which checks if all the configurations finished successfully. After the Action operations for a given device finish execution, it votes on whether the configurations for all devices should be committed or aborted. If all votes approve that configurations should be committed, commit operations are executed for every device. If it decides to abort and rollback all configurations, rollback operations are executed for all devices, as shown in Table 3.3. Since this is the serial mode, if one device decides to Abort, the configuration will not proceed to the following devices. The majorCode depends on the success of all operations for all devices and the minorCode depends on the success of the rollback operations.

Atomic Parallel This mode also follows a two-phase commit approach similar to the mode described before. The difference resides on the fact that the configuration for all devices is executed in Parallel as seen in Table 3.4, with the plug-in checking if all the configurations went successfully. If there is an error, rollback operations are executed for all devices. The majorCode depends on the success of all operations for all devices and the minorCode depends on the success of the rollback operations.

<table>
<thead>
<tr>
<th>Device 1</th>
<th>Device 2</th>
<th>Device 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>Connect</td>
<td>Connect</td>
</tr>
<tr>
<td>Read</td>
<td>Read</td>
<td>Read</td>
</tr>
<tr>
<td>Write</td>
<td>Write</td>
<td>Write</td>
</tr>
<tr>
<td>Vote Commit</td>
<td>Vote Commit</td>
<td>Vote Abort</td>
</tr>
<tr>
<td>Rollback</td>
<td>Rollback</td>
<td>Rollback</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Disconnect</td>
<td>Disconnect</td>
</tr>
</tbody>
</table>

These execution modes must be used wisely by network managers and together with the different types of operations in order to maintain the network in a known state.
4 Evaluation

Contents

4.1 Test Environment ......................................................... 41
4.2 JUnit Tests ................................................................. 42
4.3 HPSA Plug-in Test ......................................................... 43
4.4 Generic_CLI comparison .................................................. 44
GenericNETCONF is focused on the standard defined by RFC 6241 and on assuring interoperability with standard server implementations. Given this, in order to evaluate this work, it is essential to guarantee that this implementation follows the standard and is able to interact with a NETCONF server implementation which is recognized by IETF’s NETCONF working group. Also, because the project can grow beyond the standard, the evaluation should keep up with that evolution. This section contains the methods used to evaluate this project and the results of that evaluation.

4.1 Test Environment

Since there are not many available free implementations of a NETCONF server, all tests were performed using Netopeer, which is built using libnetconf. Libnetconf implements the standard operations and has been a reference for several research projects and even official interoperability meetings hosted by IETF [? ,28].

![Figure 4.1: Test Environment](image)

This work was tested using three Virtual Machines (VMs), created using VirtualBox, as seen in Figure 4.1. The first VM was configured with 2 GB of RAM and Red Hat 6.4 x64 as Operating System, the required to install HPSA [29]. Since the version used to test this plug-in was HPSA 6.1, the first machine runs Java™ SE Development Kit 6, Update 37 and uses Oracle Database Express Edition 11g for the database. Two other VMs were created with 512 MB of RAM, use also Red Hat 6.4 x64 as Operating System, and have Netopeer installed. Netopeer installation provides netopeer-cli, which works as a NETCONF client to manage devices through any terminal, and netopeer-server, which implements the server used to test GenericNETCONF. Netopeer Server has a multi-level architecture and can be extended with modules to control different areas of the managed device. By default, netopeer-server comes with three pre-installed modules, as described in Listing 4.1:
1. *ietf-interfaces* to manage network interfaces and to implement RFC 7223 [30],

2. *ietf-system* to implement part of RFC 7317 [31]

3. *turing-machine* module. Created originally for a tool called *pyang* and used as a dummy machine to test NETCONF implementations.

   Listing 4.1: Get netopeer modules configuration
   ```xml
     <modules>
       <module>
         <name>ietf-interfaces</name>
         <enabled>true</enabled>
       </module>
       <module>
         <name>ietf-system</name>
         <enabled>true</enabled>
       </module>
       <module>
         <name>turing-machine</name>
         <enabled>false</enabled>
       </module>
     </modules>
   </netopeer>
   ```

4.2 JUnit Tests

The interoperability with the standard implementations of NETCONF is one of the main goals of this work, therefore it is essential to test each operation to assure that the content which comes from workflows follows the specification and to test if the server answers successfully to each operation. Since most of this project is done in JAVA, a set of JUnit tests was created taking in consideration that GenericNETCONF can be possibly extended with custom operations and new operations which can be defined accordingly to the work done by IETF’s NETCONF working group. These tests were divided in four main groups:

1. **Input Tests**: Mostly negative tests with invalid inputs, taking in consideration the schema defined for the XML specification and which should not be considered as valid by the plug-in.

2. **Connection Tests**: Connection success and errors are tested under the four different execution modes. In non-atomic execution modes, the execution finishes successfully if the plug-in is able to connect to at least one device, yet it may or may not be consistent.

3. **Operations and Content Tests**: All operations are tested with valid and invalid content. The goal is to assure that the operation validates the input as expected and returns an error if the content is invalid.
4. **Execution Mode Tests:** Four operations are executed for both devices using each execution mode: Lock, get-config, edit-config, unlock.

All these sets of tests, assess the plug-in end to end, meaning that each test defines an input XML specification which calls the plug-in by its atomic task and asserts on its ExecutionDescriptor response code and description. The main idea of these tests is for them to be extended and to run on every release as the plug-in is updated, guaranteeing that the previously implemented operations keep consistent with the standard and the defined inputs. The results can be seen in Appendix B.

### 4.3 HPSA Plug-in Test

To test the plug-in within HPSA environment, a solution was created taking into consideration the tests’ environment. The goal of this test was to update an inventory instance existent on HPSA with devices configuration state, and update configuration of these devices, accessing HPSA from its built-in provided web interface.

![Figure 4.2: Test Environment HPSA Inventory Beans](image)

A simple inventory entity was created on HPSA with information relative to each devices modules state. The inventory instance contains the two beans represented in Figure 4.2, one relative to device information and one other with modules state information.

These beans are created in XML and deployed using a provided tool called InventoryBuilder seen in Figure 2.5.

For the tree to be visible on HPSA Graphical user interface (GUI), an inventory tree was created using a tool provided by HPSA, the InventoryTreeDesigner (also represented in Figure 2.5), which is used to create relationships between the inventory beans and to create a view that can be used by the GUI to generate the view shown in Figure 4.3.

To be able to update the inventory and the devices’ configuration, three workflows were used:

1. **Netconf_Parallel_Enable_Turing:** As shown in Figure 4.4, it is a very simple workflow with only six nodes. It maps the necessary variables from the inventory bean to the workflow local variables, and creates the input XML to be sent to the GenericNetconf plug-in generated using a XSD template
created to enable the turing-machine module on two devices (as seen in Appendix B). After the configuration, the workflow logs a message indicating if the configuration went successfully or not.

2. **Netconf.Atomic_Disable_Turing**: Follows the same structure as the previous workflow, yet it disables the turing-module in both devices using an Atomic.Parallel execution mode.

3. **Netconf.Update_Config**: Executes a get-config operation on the managed device, interprets the response and updates the information on HPSA’s inventory.

### 4.4 Generic_CLI comparison

Considering that GenericCLI is the most common used plug-in in HPSA for network configuration, it makes sense to compare both plug-ins executing a simple operation. To do it, GenericCLI java classes were imported to an Eclipse project and a benchmark class was created. Both plug-ins were executed to enable the turing-module from one of the netopeer test machines.

Netopeer stores its candidate configuration in an XML document file typically located in /usr/local/etc/netopeer/cfgnetopeer/datastore.xml, which allows us to use a tool named sed
to replace the content of the 52th line, which refers to the state of turing-module. The template used to execute this operation using GenericCLI was the one referred in Listing 4.2.

**Listing 4.2:** Enable the Turing-Module in Netopeer’s candidate configuration using GenericCLI

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output doctype-system="CLIv4.dtd" />
  <xsl:param name="host" />
  <xsl:param name="port" />
  <xsl:param name="userName" />
  <xsl:param name="password" />
  <xsl:template match="/">
    <CLI>
      <Connect protocol="ssh" pooling="false" host="\{\$host}\" port="\{\$port}\" ssh.username="\{\$userName}\" ssh.password="\{\$password}\" ssh.isEncrypted="no" ssh.allow_host="true" ssh.known_hosts="/\.ssh/known_hosts" ssh.identity="-" store_always="false" timeout="1" >
        <Do>
          <Prompt>.*#</Prompt>
        </Do>
      </Connect>
    </CLI>
    <Action description="Enable Turing Module">
      <Do>
        <Command>sed '52 s/false/true/' /usr/local/etc/netopeer/cfgnetopeer/datastore.xml</Command>
        <Error>sed:</Error>
        <Prompt>.*#</Prompt>
      </Do>
      <Undo>
        <Command>sed '52 s/true/false/' /usr/local/etc/netopeer/cfgnetopeer/datastore.xml</Command>
        <Error>sed:</Error>
        <Prompt>.*#</Prompt>
      </Undo>
    </Action>
    </CLI>
  </xsl:template>
</xsl:stylesheet>
```

To execute the same operation using GenericNETCONF, the template used was the one shown in Listing 4.3. A significant advantage of using NETCONF instead of CLI is that error handling becomes easier. Using CLI, error messages are vendor specific, for that purpose GenericLCI provides a tag `<Error>` which will run every reply against a regular expression (regex). Using GenericNETCONF, errors are handled by the plug-in and its description can be accessed through the variable `sterr` of the returned `ExecutionDescriptor`. For this specific case, the regex expression to find an error coming from `sed` is not very complex, yet for more complex activations, it is difficult and takes too much effort to design regular expressions for all different errors for all the different managed devices.
To evaluate the performance of both plug-ins, the time to execute the aforementioned operation was measured. The following method was used to reduce the uncertainty of the test:

Let’s consider $Y$ the random variable representing the running time of an HPSA plug-in. Since $Y$ follows a Poisson distribution with parameter $\lambda$ and we want to estimate the average running time, we just have to estimate $\lambda$. For that, we will use the sequence of average running times given by $\{X_n\}_{n=1}^\infty$ such that $X_n = \frac{1}{n} \sum_{i=1}^n Y_i$, where $Y_i$ are independent and have the same distribution as $Y$. Therefore:

$$E[X_n] = E\left[\frac{1}{n} \sum_{i=1}^n Y_i\right] = \frac{1}{n} \sum_{i=1}^n E[Y_i] = \frac{1}{n} \times n\lambda = \lambda$$
\[ \text{Var}[X_n] = \text{Var} \left[ \frac{1}{n} \sum_{i=1}^{n} Y_i \right] = \frac{1}{n^2} \sum_{i=1}^{n} \text{Var}[Y_i] = \frac{1}{n^2} \times n \lambda = \frac{\lambda}{n} \]

Moreover we conclude that the estimator is consistent since \( \lim_{x \to \infty} \frac{\lambda}{n} = 0 \) and so the method converges.

Using the central limit theorem, a 95\% confidence interval for \( \lambda \) is \( \lambda \pm q_{\alpha} \sqrt{\frac{1}{n}} \), where \( q_{\alpha} = 1.96 \) is the 97.5\textsuperscript{th} percentile of the \( \text{Normal}(0, 1) \). Thus, the method works as shown in Algorithm 4.1:

**Algorithm 4.1: Compute a 95\% confidence interval for an HPSA plugin's running time**

**Result:** A 95\% confidence interval for the plugin's running time

**begin**

\( n \leftarrow 0 \)

while \( \sigma \geq 1 \) do

Run an operation for one device and compute the average running time \( X_n \)

Compute the standard error \( \sigma \leftarrow \sqrt{\frac{\sigma}{n}} \)

\( n \leftarrow n + 1 \)

else \( I = [x_n - 1.96 \times \sqrt{\frac{\sigma}{n}}, x_n + 1.96 \times \sqrt{\frac{\sigma}{n}}] \)

**return** \( I \)

**end**

The interval obtained for GenericCLI was \( I = [4897.04, 4900.96] \) after 2451 runs with an average running time of 4899 ms and the interval obtained for GenericNETCONF was \( I = [628.04, 631.96] \) after 316 runs and an average running time of 630 ms. Taking a closer look, we can conclude that GenericCLI has some delay creating the ssh connection when compared with GenericNETCONF. Also, because GenericCLI took more runs to converge than GenericNETCONF, we conclude that the second is less variable than the first.

Even though GenericNETCONF was significantly faster, it was not created to replace GenericCLI. Both protocols have advantages over each other and that should be taken in consideration depending on the usage. GenericNETCONF default error handling significantly decreases not only implementation errors but also the development time, as it is not necessary to confirm all the possible error messages for all requests. Also, given that NETCONF is mostly XML, it is very easy, either for human reading as for machine interpretation, which decreases implementation errors and development time on handling replies. GenericNETCONF is also useful when executing a configuration on more than one device, as it gives the possibility to commit the managed devices’s configurations only when it is guaranteed that all devices can be configured with success. GenericCLI on the other hand, has the significant advantages of supporting most of the operations that can be executed in a given device and also the fact that CLI is present in most network devices.
5

Conclusion

Contents

5.1 Conclusions ......................................................... 51
5.2 System Limitations and Future Work ............................. 52
5.1 Conclusions

TMF’s eTOM defines the business processes that must be implemented by any CSP, but in order to make these processes work efficiently, the right tools that are able to automate those processes and make them faster and more error-free are necessary. NETCONF is the most recent network management protocol standardized by IETF. One of the main reasons for the creation of NETCONF was to automate the network management processes. The integration of NETCONF as part of the eTOM provisioning process can bring great advantages to CSP.

Given that more than 50% of the services activation market is controlled by a few top suppliers, it will for sure be a differentiator factor if one of these solutions implements a newly created network management protocol, more focused on the CSPs needs. HPSA is completely integrated in a complete Fulfillment solution and is one of the main solutions used nowadays, making it a reliable product in terms of support and maintenance.

HPSA provides the possibility for a network manager to create an activation workflow. This activation workflows can be created through several workflow nodes, where each node represents a specific action, such as sending a configuration to a specific device, interpret replies sent by network devices or even evaluate those replies. HPSA workflows can interact with the existing plug-ins using predefined templates which have a set of variables that must be defined by the workflow. Through the creation of new workflow nodes and plug-in templates, it is possible to implement a new network management protocol.

GenericNETCONF is a new plug-in, which provides HPSA with a NETCONF southbound interface, as such, its architecture is designed to address not only the requirements specified for the network management protocol but also the requirements established for a generic plug-in. Conceptually, NETCONF is partitioned into four layers: Content, Operations, Messages and Transport. This plug-in covers all the last three layers, with the content for the operations to be executed being received as an XML input. The XML may specify instructions for several devices, that are interpreted and validated by the plug-in, which generates the messages and communicates with the target devices. The plug-in is able to interact with one or more devices and supports more than one type of concurrent execution, which can also be defined in the input XML document.

GenericNETCONF was tested with success using a typical workflow, which is probably the most common environment from where it would be summoned. A small comparison was made between GenericNETCONF and GenericCLI. GenericNETCONF is not intended to replace GenericCLI. However, it can be easily concluded that both have some pros and cons, mainly in terms of error handling, development time and concurrency handling (where NETCONF shows its value and flexibility), and where CLI has the added value of being able to do practically any operation in any device. This project was also evaluated by the interoperability of the functions it supports and by the possibility of new added
operations, still maintaining the interoperability of the base operations.

5.2 System Limitations and Future Work

Since NETCONF is so young and a lot is still in discussion, the work for this plug-in must be maintained in order to meet all the updates and changes to the main RFC. Besides the main RFC, there are several other RFCs and drafts being proposed and discussed by the IETF's NETCONF working group, making it possible to extend this plug-in for other scenarios. YANG datamodels are also starting to get more and more common for NETCONF managed devices and more custom operations are currently being implemented by network devices manufacturers. One possible approach to extend this plugin, would be to create a YANG interpreter which would use custom YANG models to automatically extend the plugin capabilities.

NETCONF implementation is a growing market which will for sure be driven even further with the current evolution of NFV and SDN. Right now, NETCONF is being discussed by several entities as one of the main protocols for configuration and orchestration of SDNs and NFVs, which means this GenericNETCONF plug-in can also possibly be extended to meet the requirements for those kinds of technologies.
Bibliography


xmlSpec Schema Definition

The xmlSpec schema is used to validate GenericNETCONF’s input. It defines which variables can be used to configure the plug-in, which devices to manage and what are the operations it should execute on the managed device. At least one managed device should be sent as input and one “action” operation for that managed device. As the connection is maintained until the plugin finishes, there can not be two managed devices with the same host.

Listing A.1: Schema is used to validate the XML specification received by GenericNETCONF

```xml
  <xs:element name="netconf">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="device" maxOccurs="unbounded" minOccurs="1">
          <xs:complexType>
            
        </xs:complexType>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
<xs:sequence>
  <xs:element name="action" minOccurs="1" maxOccurs="1">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="operation" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:any processContents="skip" maxOccurs="unbounded" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="output" type="xs:string"/>
            <xs:attribute name="retry" type="xs:boolean" default="false"/>
            <xs:attribute name="numOfRetries" type="xs:integer" default="3"/>
            <xs:attribute name="timeBetweenRetries" type="xs:integer" default="60"/>
            <xs:attribute name="rollbackOnError" type="xs:boolean" default="true"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:sequence>

<xs:element name="commit" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="operation" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:any processContents="skip" maxOccurs="unbounded" minOccurs="0"/>
          </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="output" type="xs:string"/>
            <xs:attribute name="retry" type="xs:boolean" default="false"/>
            <xs:attribute name="numOfRetries" type="xs:integer" default="3"/>
            <xs:attribute name="timeBetweenRetries" type="xs:integer" default="60"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:sequence>
<xs:complexType>
  <xs:element name="rollback" minOccurs="0">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="operation" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:any processContents="skip" maxOccurs="unbounded" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="output" type="xs:string"/>
            <xs:attribute name="retry" type="xs:boolean" default="false"/>
            <xs:attribute name="numOfRetries" type="xs:integer" default="3"/>
            <xs:attribute name="timeBetweenRetries" type="xs:integer" default="60"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:complexType>

<xs:complexType>
  <xs:element name="undo" minOccurs="0">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="operation" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:any processContents="skip" maxOccurs="unbounded" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="output" type="xs:string"/>
            <xs:attribute name="retry" type="xs:boolean" default="false"/>
            <xs:attribute name="numOfRetries" type="xs:integer" default="3"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:complexType>
<xs:attribute name="timeBetweenRetries" type="xs:integer" default="60"></xs:attribute>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="host" type="xs:string" use="required"></xs:attribute>
<xs:attribute name="port" type="xs:int" default="830"></xs:attribute>
<xs:attribute name="user" type="xs:string" use="required"></xs:attribute>
<xs:attribute name="password" type="xs:string" use="required"></xs:attribute>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="executionMode" default="serial">
JUnit battery tests used to verify if all :base capabilities work correctly. The tests are used to confirm the input validation against the defined schema and to test the interoperability of GenericNETCONF with a NETCONF server implementation. All tests are executed end to end, meaning a xmlSpec is sent as input to the plug-in and its reply is evaluated against a regex expression.
Table B.1: JUnit battery tests used to evaluate GenericNETCONF functionality

<table>
<thead>
<tr>
<th>Group</th>
<th>Test</th>
<th>Description</th>
<th>Expected Result</th>
<th>Major Code</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Test 1001</td>
<td>Valid Input</td>
<td>Connection Error</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1002</td>
<td>Invalid ExecutionMode</td>
<td>^cvc.+</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1003</td>
<td>Missing host</td>
<td>^cvc.+</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1004</td>
<td>Missing port</td>
<td>Connection Error</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1005</td>
<td>Missing user</td>
<td>^cvc.+</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1006</td>
<td>Missing password</td>
<td>^cvc.+</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1007</td>
<td>Missing device</td>
<td>^cvc.+</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1008</td>
<td>Missing action</td>
<td>^cvc.+</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1009</td>
<td>Missing commit</td>
<td>Connection Error</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1010</td>
<td>Missing rollback</td>
<td>Connection Error</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1011</td>
<td>Missing Undo</td>
<td>Connection Error</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1012</td>
<td>Missing Operation on action</td>
<td>^cvc.+</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1013</td>
<td>Missing Operation on commit</td>
<td>^cvc.+</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1014</td>
<td>Missing Operation on rollback</td>
<td>^cvc.+</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1015</td>
<td>Missing Operation on undo</td>
<td>^cvc.+</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1016</td>
<td>Missing operation name</td>
<td>^cvc.+</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1017</td>
<td>Missing operation retry</td>
<td>Connection Error</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1018</td>
<td>Missing operation numOfRetries</td>
<td>Connection Error</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1019</td>
<td>Missing operation timeBetweenRetries</td>
<td>Connection Error</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1020</td>
<td>Missing operation rollbackOnError</td>
<td>Connection Error</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 1021</td>
<td>Invalid Filename</td>
<td>File not found</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td>Connection</td>
<td>Test 2001</td>
<td>Connect successfully to 1 different devices</td>
<td>OK</td>
<td>PASSED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 2002</td>
<td>Connect successfully to 2 different devices</td>
<td>OK</td>
<td>PASSED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 2003</td>
<td>Connect unsuccessfully to two different devices</td>
<td>Connection Error</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 2004</td>
<td>Connect unsuccessfully to two different devices (serial)</td>
<td>Connection Error</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 2005</td>
<td>Connect unsuccessfully to two different devices (parallel)</td>
<td>Connection Error</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 2006</td>
<td>Connect successfully to two different devices</td>
<td>OK (inc)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>get</td>
<td>Test 3001</td>
<td>No content</td>
<td>OK</td>
<td>PASSED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 3002</td>
<td>Missing filter content</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3003</td>
<td>Wrong filter Type</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3004</td>
<td>Wrong input (&lt;test-attribute-&gt;)</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td>get-config</td>
<td>Test 3011</td>
<td>No content</td>
<td>OK</td>
<td>PASSED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 3012</td>
<td>Missing source</td>
<td>Input Mandatory</td>
<td>OK</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3013</td>
<td>Missing datasource</td>
<td>Dataset Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3014</td>
<td>Wrong datastore</td>
<td>Invalid datastore</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3015</td>
<td>Missing filter</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3016</td>
<td>Missing filter content</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3017</td>
<td>Wrong filter Type</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3018</td>
<td>Wrong input (&lt;test-attribute-&gt;)</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td>edit-config</td>
<td>Test 3021</td>
<td>No content</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3022</td>
<td>Missing target</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3023</td>
<td>Missing datasource</td>
<td>Dataset Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3024</td>
<td>Wrong datastore</td>
<td>Invalid datastore</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3025</td>
<td>Missing Config or URL</td>
<td>Input Mandatory</td>
<td>OK</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3026</td>
<td>Valid config</td>
<td>OK</td>
<td>PASSED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 3027</td>
<td>Wrong input (&lt;test-attribute-&gt;)</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td>copy-config</td>
<td>Test 3031</td>
<td>No content</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3032</td>
<td>Missing target</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3033</td>
<td>Missing target datasource or URL</td>
<td>Dataset Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3034</td>
<td>Wrong target datasource</td>
<td>Invalid datastore</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3035</td>
<td>Missing source</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3036</td>
<td>Missing source datasource or config or url</td>
<td>Dataset Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3037</td>
<td>Wrong source datastore</td>
<td>Invalid datastore</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3038</td>
<td>Valid Source config</td>
<td>OK</td>
<td>PASSED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 3039</td>
<td>Wrong input (&lt;test-attribute-&gt;)</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td>delete-config</td>
<td>Test 3051</td>
<td>No content</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3052</td>
<td>Missing target</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3053</td>
<td>Missing datasource or url</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3054</td>
<td>Wrong target datasource</td>
<td>Invalid datastore</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3055</td>
<td>Valid datastore</td>
<td>OK</td>
<td>PASSED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 3056</td>
<td>Wrong input (&lt;test-attribute-&gt;)</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td>Group</td>
<td>Test</td>
<td>Description</td>
<td>Expected Result</td>
<td>Major Code</td>
<td>Result</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------------------------------------</td>
<td>-----------------</td>
<td>------------</td>
<td>--------</td>
</tr>
<tr>
<td>Operations + Content (Atomic, Parallel)</td>
<td>Test 3061</td>
<td>No content</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3062</td>
<td>Missing target</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3063</td>
<td>Missing datastore</td>
<td>Datastore Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3064</td>
<td>Wrong target datastore</td>
<td>Invalid datastore</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3065</td>
<td>Valid datastore</td>
<td>OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 3066</td>
<td>Wrong input (&lt; test-attribute &gt;)</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td>lock</td>
<td>Test 3071</td>
<td>No content</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3072</td>
<td>Missing target</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3073</td>
<td>Missing datastore</td>
<td>Datastore Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3074</td>
<td>Wrong target datastore</td>
<td>Invalid datastore</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3075</td>
<td>Valid datastore</td>
<td>OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 3076</td>
<td>Wrong input (&lt; test-attribute &gt;)</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td>Unlock</td>
<td>Test 3071</td>
<td>No content</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3072</td>
<td>Missing target</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3073</td>
<td>Missing datastore</td>
<td>Datastore Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3074</td>
<td>Wrong target datastore</td>
<td>Invalid datastore</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3075</td>
<td>Valid datastore</td>
<td>OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 3076</td>
<td>Wrong input (&lt; test-attribute &gt;)</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td>close-session</td>
<td>Test 3081</td>
<td>No content</td>
<td>OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kill-session</td>
<td>Test 3091</td>
<td>No content</td>
<td>Input Mandatory</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3092</td>
<td>Invalid SessionId</td>
<td>String Error</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3093</td>
<td>Wrong input (&lt; test-attribute &gt;)</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td>commit</td>
<td>Test 3091</td>
<td>No content</td>
<td>OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 3095</td>
<td>Wrong input (&lt; test-attribute &gt;)</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td>discard-changes</td>
<td>Test 3101</td>
<td>No content</td>
<td>OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 3102</td>
<td>Wrong input (&lt; test-attribute &gt;)</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td>validate</td>
<td>Test 3111</td>
<td>No content</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3112</td>
<td>Valid Config</td>
<td>OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 3113</td>
<td>Missing target</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3114</td>
<td>Wrong datastore</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3115</td>
<td>Missing Config or URL</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td></td>
<td>Test 3116</td>
<td>Wrong input (&lt; test-attribute &gt;)</td>
<td>Invalid parameter</td>
<td>ERROR</td>
<td>PASSED</td>
</tr>
<tr>
<td>Execution Modes</td>
<td>Test 4001</td>
<td>Serial: Lock + Get-config + Edit-config + Get-config + Unlock for 2 devices</td>
<td>OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution Modes</td>
<td>Test 4002</td>
<td>Atomic, Serial: Lock + Get-config + Edit-config + Get-config + Unlock for 2 devices</td>
<td>Ok</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution Modes</td>
<td>Test 4003</td>
<td>Parallel: Lock + Get-config + Edit-config + Get-config + Unlock for 2 devices</td>
<td>Ok</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution Modes</td>
<td>Test 4004</td>
<td>Atomic, Parallel: Lock + Get-config + Edit-config + Get-config + Unlock for 2 devices</td>
<td>Ok</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution Modes</td>
<td>Test 4005</td>
<td>Serial: Error</td>
<td>OK (inc)</td>
<td>Wrong filter</td>
<td>PASSED</td>
</tr>
<tr>
<td>Execution Modes</td>
<td>Test 4006</td>
<td>Atomic, Serial: Error</td>
<td>OK (inc)</td>
<td>Wrong filter</td>
<td>PASSED</td>
</tr>
<tr>
<td>Execution Modes</td>
<td>Test 4007</td>
<td>Parallel: Error</td>
<td>OK (inc)</td>
<td>Wrong filter</td>
<td>PASSED</td>
</tr>
<tr>
<td>Execution Modes</td>
<td>Test 4008</td>
<td>Atomic, Parallel: Error</td>
<td>OK (inc)</td>
<td>Wrong filter</td>
<td>PASSED</td>
</tr>
</tbody>
</table>