Universidad Nacional de la Plata

Facultad de Informática

Diseño e Implementación de servidor de management para la gestión de dispositivos con soporte Netconf

Ing. Martín A. BERTOLINA

Tesis presentada para obtener el grado
de Magister en Redes de Datos

Directora: Mg. Lía Molinari
Co-Director: Ing. Luis Marrone

Nov 2011

Secretaría de Postgrado

A mis viejos, Edith y Osvaldo, por dar a sus hijos lo mejor.
A Alejandra, por sus consejos y palabras de orientación.
A Andy y Mariano !

Agradecimientos

Me gustaría agradecer a la Profesora Mg. Lía Molinari por sus conocimientos transmitidos, su apoyo y dedicación en el desarrollo y evaluación de este trabajo de postgrado.

A Juan Fraire por la ayuda en el diseño de la página web. A Guido Ale por el seguimiento, corrección del informe y constante feedback.

También, no puedo dejar sin agradecer a mi familia cuyo estímulo constante y amor he confiado a lo largo de mi desarrollo profesional y personal.

Índice general

Índice de figuras

Índice de cuadros

Índice de Figuras con Código

Capítulo 1

Introducción

1.1. Motivación

Las redes de comunicaciones crecen a medida que surgen nuevas tecnologías (las inalámbricas de cuarta generación 4G, WiMAX y LTE), seguido de la demanda de usuarios por una mejor calidad de servicio y mayor velocidad, constituyéndose en un desafío para su desarrollo. En consecuencia, los operadores se ven obligados a aumentar la cantidad de dispositivos de red para satisfacer dichas necesidades e implementar nuevas soluciones para administrar la configuración de los mismos. Si bien SNMP provee operaciones para la configuración de dispositivos de red, éste es limitado para aquellas funciones relacionadas a la recolección de estadísticas e información de su estado, además de presentar dificultades en su utilización con propósitos de configuración. Las deficiencias de SNMP se evidencian en los siguientes aspectos: [35]:

- SNMP es un protocolo simple, dejando la responsabilidad de manipulación de datos de configuración a las aplicaciones de gestión, por este motivo, las herramientas de desarrollo basados en SNMP son costosas.

- Las peticiones SET se mandan independientemente, lo cual puede causar problemas serios en la red si el gestor envía varias peticiones SET para configurar un dispositivo en particular y si una de éstas falla.

- SNMP no provee ningún mecanismo para anular los cambios recientes en la configuración del dispositivo.

- SNMP no provee ningún mecanismo para sincronizar la configuración en múltiples elementos de red. Si la aplicación de gestión envía peticiones SET a un grupo de dispositivos (para que posean la misma configuración), algunas de ellas puede ser exitosas, y otras pueden fallar.

- No emplea un mecanismo de seguridad estándar. La seguridad está contenida dentro del mismo protocolo, que hace que las credenciales y el manejo de claves SNMP sea complejo y difícil de integrar con otras credenciales existentes y claves de los sistemas de gestión.

Actualmente, las Command-Line Interfaces (CLI) propietarias e interfaces web son preferidas por los operadores y algunas veces hasta son las únicas alternativas. Network configuración protocolo (NETCONF)[22] surge como una solución simple y estándar para la gestión de configuraciones. Este fue definido en RFC 4741 y proporciona a los operadores (y administradores de red) un framework y un conjunto de métodos RPC (Remote Procedure Calls) basados en codificación XML (Extensible Markup Language) para gestionar (instalar, modificar y borrar) la configuración en los elementos de red. Este nuevo protocolo fue diseñado por la IETF (Internet Engineering Task Force) para reemplazar las interfaces de management CLI (Command Line Interfaces), como SSH (Secure Shell) y SNMP (Simple Network Management Protocol).

Las principales fortalezas de Netconf se pueden resumir en:

- Capacidad para mantener y operar diferentes bases de datos (o archivos) de configuración en el dispositivo.

- Habilidad para establecer sesiones de gestión sobre comunicaciones seguras (encriptadas y autenticadas). Las mismas pueden establecerse en redes inseguras como Internet.

- La gestión puede realizarse tanto de modo centralizado como descentralizado (muchos gestores pueden manejar al mismo dispositivo).

- Soporte de transacciones (como commit o rollback) para ser ejecutadas en la etapa final de edición de la configuración.

- Clara separación de la configuración y de las variables de estado del dispositivo.

- Soporte para reemplazar, crear, borrar y unir (merge) parámetros de configuración.
- Manejo fácil de dispositivos basado en XML.
  - Es una tecnología estable y asumida por una gran comunidad de profesionales, disponible en muchas herramientas de software, haciendo el desarrollo de las aplicaciones mucho más eficiente.
  - XML se integra fácilmente en otras aplicaciones.
  - Provee características de modelado poderosas para manejo estructurado de la información.
  - Gran variedad de APIs para la manipulación y acceso de documentos XML con Document Object Model (DOM), XML schemas y XML Path (Xpath).
  - La representación en documentos XML es más legible tanto para humanos y máquinas.
- Netconf pude ser extendido fácilmente por diferentes vendors, incorporando nuevas funcionalidades (capabilities) y operaciones. Esto permite una reducción de los costos, precios y time-to-market.
- Diseñado especialmente para el manejo de la configuración de los elementos de red, tareas que no son ejecutadas regularmente.

La creación de este nuevo protocolo, ligado a nuevos métodos, lenguajes de modelado y arquitecturas de red para la gestión de elementos de red, y debido a la tendencia de las empresas operadoras a utilizar este tipo de tecnologías interoperables, proveen ventajas claras para la implementación de nuevas soluciones en este ámbito y fueron motivación para la elaboración de este trabajo de postgrado.

1.2. Objetivos propuestos

En el desarrollo de esta tesis se propuso *presentar* el nuevo protocolo Netconf RFC 4147 y su profunda relación con las tareas de management de dispositivos de red. Los objetivos principales fueron *diseñar* e *implementar* un servidor de gestión de elementos llamado NEMS (Netconf Element Management System), completamente programado en Python[8], con la función principal de administrar

la configuración de aquellos equipos con soporte del protocolo Netconf. Se ha utilizado el mismo para *validar* los métodos propuestos por el estándar y *verificar* su capacidad de controlar diversos dispositivos.

Además, se planteó *documentar* en este trabajo el diseño y la implementación del sistema de gestión propuesto, las operaciones, posibles casos de uso y las posibles extensiones futuras de los módulos asociados.

1.3. Network management

El término gestión de redes es bastante amplio dado que abarca muchas actividades y es considerado como uno de los pilares más importantes para las empresas propietarias de soluciones de red. Éste se refiere a todos los procedimientos, métodos y herramientas para la operación, administración, mantenimiento y aprovisionamiento de los sistemas de redes.

El término *Operación* en adelante se refiere a las actividades para mantener los elementos de red, y los servicios asociados que la red proporciona, funcionando continuamente (up and running). Para ello, es fundamental el *monitoreo* continuo de los equipos para poder enfocarse en los problemas (o posibles amenazas) lo más rápido posible, antes que los usuarios puedan ser afectados. La *administración* se encarga de mantener el seguimiento de los recursos de la red y cómo éstos son asignados adecuadamente. El término *mantenimiento* hace referencia a la realización de upgrades, reemplazos, reparaciones y cambios de configuraciones, también involucra actividades preventivas, correctivas y de optimización para que la red funcione lo mejor posible y se ajuste a la demanda de requerimientos. Se entiende que los procedimientos de *aprovisionamiento* se basan en la configuración y puesta a punto de los recursos para soportar algún servicio dado. La forma adoptada generalmente para caracterizar las funciones de gestión de red es ISO Management Model - *FCAPS*: Fault, Configuration, Accounting, Performance y Security[34]:

- **Fault:** se llamará al ámbito en que los problemas de red son detectados y corregidos. Se identifican potenciales dificultades o posibles amenazas y se toman las acciones adecuadas para prevenir de que ocurran o que vuelvan a suceder. De esta forma, se trata mantener la red operacional y que los períodos de inactividad se minimicen.

- **Configuration:** se designa a la estructura de red que adopta las funciones de monitoreo y control de configuraciones y equipamiento. En este área se coordinan actividades de cambios de hardware y programas, instalación de nuevo equipamiento y software, modificación de sistemas existentes y cambios de configuración de elementos de red.

- **Accounting:** es el área que posee como objetivo principal la recolección de estadísticas de la red y la realización continua del seguimiento de la información para ser utilizada en la facturación de los servicios brindados.

- **Performance:** determina la eficiencia actual de la red y su gestión se encuentra enfocada en asegurar que la performance de la red se encuentre dentro de los valores aceptables. Para ello, se recolecta la información de tiempos de respuesta, pérdida de paquetes, utilizaciones de vínculos y conexiones, etc.

- **Security:** su objetivo es controlar el acceso de los elementos de red y asegurar que la red opere en un ambiente seguro. Se establecen métodos de autenticación, autorización y auditoría, para que los usuarios locales y externos puedan acceder a los recursos adecuadamente y realizar operaciones autorizadas.

Como se desarrollará a lo largo de este trabajo, los procedimientos y operaciones de Netconf están directamente relacionados al área de configuración de elementos de red, aunque el uso de notificaciones permite también realizar actividades en el ámbito de las fallas.

Capítulo 2

Protocolo Netconf

2.1. Protocolos de gestión

La implementación de soluciones de gestión de red son cruciales para su operación, mantenimiento y administración. Por ello se han desarrollado diferentes técnicas, mecanismos y protocolos para afrontar dichas tareas. La función principal de los protocolos de management es proveer un lenguaje en común para que los gestores y los agentes (dispositivos) se comuniquen. Los dos protocolos más utilizados y reconocidos son: SNMP (Simple Network Management Protocol) y CLIs (Command Line Interfaces)[20]. En menor medida, también se implementan soluciones basadas en: custom XML, CMIP (Common management information protocol), WMI (Windows Management Instrumentation), CORBA (Common Object Request Broker Architecture), Netflow, Syslog, JMX (Java Management Extensions) y otros propietarios como JunOS CLI (Juniper), CNEOMI (Motorola), etc.

SNMP es quizás el protocolo de gestión más conocido y el más desplegado en las redes en todo el mundo, especialmente para aplicaciones de monitoreo. SNMP es un componente de Internet Protocol Suite definido por la IETF. Está basado en la noción de que la información de gestión está organizada en MIBs (Management Information Base), con variables u objetos gestionables asociadas a identificadores de objeto (OIDs = Object Identifiers). SNMP proporciona un reducido conjunto de primitivas para que el servidor de gestión pueda leer y escribir al MIB, y el agente enviar los eventos. Se pueden encontrar tres versiones de este protocolo. SNMPv1, originalmente llamado SNMP, es la versión más simple de este protocolo y la más fácil de implementar en los agentes. SNMPv2 agrega nue-

vas funcionalidades a SNMP, especialmente aquellas relacionadas a la extracción de grandes cantidades de información de gestión de los dispositivos. SNMPv3 soluciona los problemas de seguridad y otras capacidades, pero es mucho más complejo de implementar y menos simple que SNMPv1.

CLI (Command Line Interface) no es formalmente un protocolo de gestión de dispositivos, sino es una interfaz basada en texto para que los humanos podamos interactuar con los dispositivos. Todos los elementos de red poseen una interfaz en linea de comandos, generalmente propietaria. Éstas son diseñadas especialmente para ser utilizadas productivamente por los administradores de red y aplicaciones encargadas de la gestión de los dispositivos, especialmente para su aprovisionamiento, edición y visualización de configuraciones. Sin embargo, la necesidad de parseo de las respuestas de CLI representa un desafío para la construcción y mantenimiento de las aplicaciones (o scripts) utilizadas para la administración de los elementos de red.

Cabe reconocer que una tarea de gestión puede ser realizada por más de un protocolo y que algunos pueden usarse como complemento de otros. Por ejemplo, SNMP, CLI y Netconf pueden implementarse para la edición de configuraciones de dispositivos, SNMP y syslog para el reporte de eventos.

2.2. Antecedentes

En el año 2002 el Internet Architecture Board (IAB) organizó un workshop para guiar las actividades de estandarización de protocolos de gestión de red en la IETF (Internet Engineering Task Force). En él estuvieron presentes operadores de redes, desarrolladores de protocolos y fabricantes de dispositivos, quienes documentaron recomendaciones concretas en el RFC 3535[35]. Una de ellas fue aumentar los esfuerzos y recursos de la IETF para enfocarse en la gestión de dispositivos de red. Otra fue el uso del lenguaje XML para propósitos de codificación de mensajes y datos.

En el año 2003 se creó un grupo de trabajo en el área de Operaciones y Gestión (Operations and Management) de la IETF con el fin de desarrollar un protocolo de management de redes. El resultado fue la creación del protocolo Netconf con la influencia de otros protocolos propietarios como Juniper Networks'JUNOScript application programming interface (API), con la necesidad de proveer una interfaz que fuera interoperable entre fabricantes para administrar los datos de configura-

ción de los dispositivos de red. Así, se dejó de lado la propuesta de automatizar CLIs utilizando programas o scripts, especialmente debido al tiempo implementación, al costo de mantenimiento y la falta de interoperabilidad entre equipos desarrollados por diferentes vendors[26] [38].

2.3. NETCONF

La gestión de la configuración de los elementos de red se ha convertido actualmente en un requerimiento crítico para los operadores que desean disponer de redes totalmente interoperables. Numerosos operadores, desde los pequeños a los más grandes, han desarrollado sus propios mecanismos o han usado mecanismos propietarios para transferir los datos de configuración hacia y desde los dispositivos, y para el análisis de la información de estado que puede impactar también en la configuración. Las particularidades de cada uno de estos mecanismos abarcan varios aspectos, como el establecimiento de la sesión, autenticación del usuario, intercambio de datos de configuración y las respuestas de error.

Network Configuration Protocol (NETCONF) fue definido inicialmente en el RFC 4741[22] por Network Configuration (netconf) Group, dentro de Internet Engineering Task Force (IETF), en Diciembre de 2006[4], y luego fue ratificado incluyendo cambios menores e incorporando nuevas características en el RFC 6241[23] en Junio de 2011, haciendo obsoleto el anterior (a pesar de que el RFC 4741 quedó obsoleto por la publicación del RFC 6241, este trabajo se referirá al RFC 4741 cuando se mencione al protocolo Netconf). Este protocolo especifica los mecanismos para instalar, manipular y borrar los parámetros de configuración de los dispositivos de red, a su vez permite que el dispositivo exponga de manera formal una interfaz de programación de aplicación (API), para ser implementadas en aplicaciones de gestión con el fin de enviar, recibir, de forma parcial o total, los cambios de configuración requeridos.

Netconf utiliza un paradigma basado en llamadas a procedimientos remotos (RPC). El cliente (generalmente el servidor de gestión) codifica una petición RPC (u operación, request) en Extensible Markup Language (XML) y lo envía al servidor (el dispositivo) utilizando una sesión segura y orientada a la conexión. El servidor responde con un mensaje de reply también codificado en XML. Los contenidos de ambos mensajes, request y reply, son descriptos totalmente por XML DTDs o XML Schemas, o ambos, permitiendo a ambas partes reconocer la

sintaxis y restricciones impuestas en el intercambio.

Este protocolo también permite al cliente descubrir el conjunto de funcionalidades (capabilities) soportadas por el server. Las mismas le permiten ajustar su comportamiento para tomar ventaja de las características expuestas por el dispositivo. El alcance y extensión del protocolo se definen en función de la implementación de la sintaxis y semántica de nuevas funcionalidades, estándar (ligadas al RFC) y no estándar, en forma de capabilities.

XML es lengua franca en intercambio de información, provee un mecanismo flexible totalmente especificado por su estructura de datos en forma jerárquica. Netconf puede ser utilizado conjuntamente con tecnologías de transformación XML, como por ejemplo XSLT (Extensible Stylesheet Language Transformations), para la generación automática de configuraciones. Un ejemplo claro del uso de estas transformaciones es en el procesamiento de la información (configuración de vínculos, políticas, clientes y servicios) obtenida de consultas por parte del sistema de gestión a bases de datos. Dicha puede ser transformada utilizando XSLT scripts para ser enviados al dispositivo utilizando el protocolo Netconf.

SSH (Secure Shell Transport Layer Protocol) es el protocolo de transporte obligatorio que todo dispositivo Netconf (cliente o servidor) tiene que soportar, cuyos detalles de implementación están definidos en el RFC 4742[41]. Entre otras especificaciones se asigna el puerto TCP = 830 por defecto y se describe la implementación del subsistema de SHH “netconf” para escuchar peticiones desde el cliente.

El servidor puede además utilizar opcionalmente otros mecanismos de transporte alternativos. RFC 4743[24] define los mecanismos para utilizar SOAP (Simple Object Access Protocol). SOAP soporta dos protocolos diferentes de transporte. Uno es Blocks Extensible Exchange Protocol (BEEP), también llamado “SOAP over BEEP”, fue definido en RFC 4227[31] (Puerto TCP = 833). Por otro lado, Hypertext Transfer Protocol (HTTP) también definido por BEEP. En este caso, los servidores NETCONF tienen que proveer de secure HTTP (HTTPS), corriendo HTTP sobre Transport Layer Security Protocol (TLS) (puerto TCP = 832). Adicionalmente, el protocolo Netconf puede a su vez correr directamente sobre BEEP. Esta funcionalidad se encuentra definida en el RFC 4744[27] y el puerto de default TCP utilizado es TCP = 831.

En la Figura 2.1 muestra una escenario de implementación del protolo Netconf. En este caso la aplicación de management utiliza solamente Netconf como

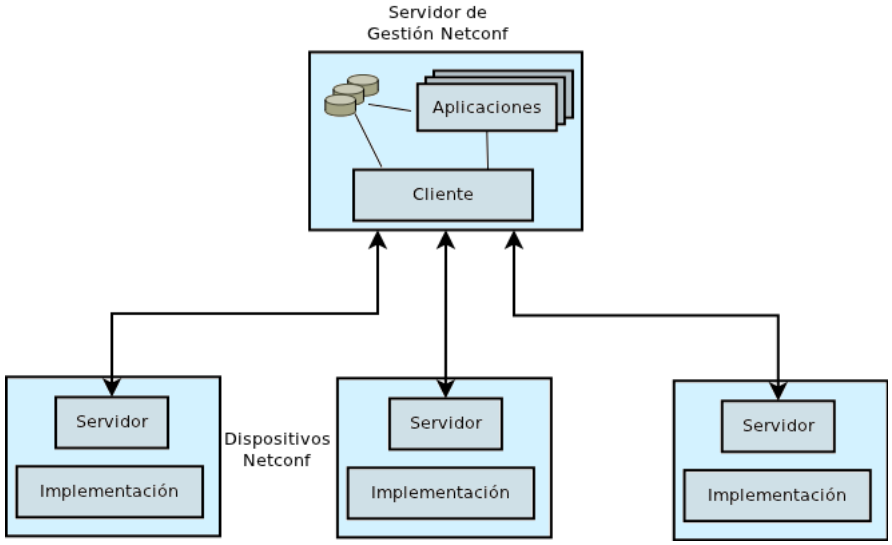

Figura 2.1: Escenario de implementación del protocolo Netconf. [26]

protocolo de gestión y actúa como cliente para enviar los cambios de configuración a los dispositivos, que incluyen un servidor Netconf. El servidor de gestión tiene la capacidad de traducir políticas, operaciones o métodos de las aplicaciones de alto nivel a mecanismos soportados por el protocolo para procesar las configuraciones de los elementos de red. El alcance de las actividades de estándares de la IETF no incluyen la manera (el cómo) en que se realiza este procesamiento. De cualquier forma Netconf provee un framework bastante completo para establecer una comunicación y realizar cambios en la configuración de los dispositivos, total o parcial, de una manera escalable y robusta. El dispositivo Netconf implementa el módulo de instrumentación. Éste se encarga de traducir las operaciones Netconf a instrucciones de dispositivo para luego ser ejecutadas sobre los datos de configuración y recursos del mismo (RFC 6244[38] proporciona pautas para la implementación de soluciones basadas en Netconf y YANG).

2.4. Terminología

A continuación se describen brevemente los términos más importantes y frecuentes relacionados al protocolo Netconf que serán utilizados en este trabajo:

- **Capability:** es una funcionalidad que se implementa como suplemento o extensión al protocolo base Netconf. Y son identificadas con un URI (Uniform Resource Identifier).

- **Client:** es la entidad que invoca las operaciones hacia el server. Generalmente, es o se encuentra en el sistema de gestión de dispositivos.

- **Server:** es la entidad que ejecuta las operaciones enviadas por el cliente. Se encuentra en el dispositivo gestionado.

- **Datastore:** es el lugar conceptual para el almacenamiento y acceso a información ubicado en el dispositivo. Un datastore puede ser implementado, por ejemplo, utilizando archivos, bases de datos, ubicaciones en memoria flash o combinación de las anteriores.

- **Configuration datastore:** es el datastore que almacena todos los datos de configuración requerida para llevar al dispositivo desde el estado default inicial al estado de operación deseado.

- **Running configuration datastore:** es un datastore de configuración que almacena la configuración que se encuentra activa en el dispositivo. Este datastore siempre existe.

- **Candidate configuration datastore:** es un datastore de configuración que puede ser manipulado sin impactar la configuración actual del dispositivo y que puede ser aplicada a al datastore running mediante una operación de commit. No todos los dispositivos soportan el datastore candidate.

- **Startup configuration datastore:** es un datastore de configuración que almacena los datos de configuración utilizados cuando el dispositivo se inicia. No todos los dispositivos soportan este datastore.

- **Notification:** es un mensaje enviado por el server al cliente para informar que un evento específico ha ocurrido.

- **User:** es la identidad autenticada del cliente, comúnmente se lo llama Netconf Username.

2.5. Netconf en capas

Netconf puede ser conceptualmente particionado en 4 capas como se muestra en la Figura 2.2.

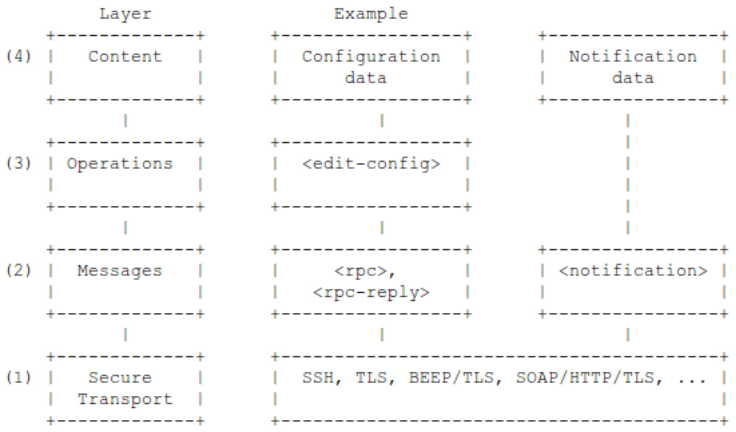

Figura 2.2: Capas de protocolo Netconf. [23]

1. **Secure Transport:** es la primera capa del protocolo Netconf. Esta proporciona un camino de comunicación entre el cliente y el servidor. Netconf puede utilizar cualquier protocolo de transporte, pero tiene que cumplir con las condiciones que impone el estándar,como por ejemplo, aquellos relacionados a la seguridad.

2. **Messages:** provee un mecanismo de enmarcado simple e independiente de la capa de transporte para codificar los RPCs y las notificaciones.

3. **Operations:** esta capa define un conjunto de operaciones básicas del protocolo invocadas como métodos RPC con parámetros codificados en XML.

4. **Content:** en esta capa se definen las estructuras de los datos de configuración, estado y notificaciones. Aunque esta capa se encuentra fuera del alcance del RFC, hay esfuerzos para estandarizarla. El lenguaje de modelado de datos YANG (RFC 6020)[14] fue desarrollado para especificar los modelos de datos para el protocolo Netconf y sus operaciones, cubriendo las capas de operaciones y contenido de la Figura 2.2.

En la etapa de establecimiento de la sesión Netconf, primero se define el canal de transporte utilizando uno de los métodos soportados tanto por el cliente como el servidor, luego se abre la sesión Netconf. Ambas entidades envían a la otra inmediatamente y en forma simultánea una lista de funcionalidades (de ahora en más utilizará el término capability). Las partes necesitan entender solamente aquellas capabilities que deben utilizar y tienen que ignorar todas aquellas que la

otra entidad no requiere o no entiende. Este intercambio de funcionalidades soportadas se realiza utilizando elementos XML <hello>. También, se proporciona información adicional, como la versión de protocolo utilizada y el identificador de la sesión (session-id). Una vez que el intercambio finaliza, el cliente está preparado para mandar al servidor las solicitudes de configuración o información de estado, el dispositivo de recibir esas solicitudes, procesarlas y responder adecuadamente. Código 1 muestra un ejemplo de intercambio de capabilities y ejecución exitosa de una operación Netconf (get-config), los elementos (o tags) XML involucrados en el mismo serán desarrollados en la sección 2.5.3.

Como se observa, el listado de capabilities se encuentra definido por el tag <capabilities> y cada una por <capability>. A su vez, el servidor incluye en su mensaje <hello> el identificador de sesión <session-id> que debe ser verificado por el cliente. Como regla general, todos los elementos de red, que soportan Netconf, están obligados a informar *urn:ietf:params:netconf:base:1.0* en este intercambio. Un ejemplo de ello es el mensaje del cliente al servidor, lo que significa que soporta solamente las funcionalidades base del protocolo y que no ha implementado ninguna extensión del mismo. El request de operación se encuentra definido en el tag <rpc> y la operación asociada con <edit-config>. Al ser una operación procesada correctamente por el servidor, este devuelve un mensaje <reply> indicando que ha sido exitosa con el elemento <ok>.

2.5.1. Protocolo de transporte: Requerimientos

Netconf utiliza un paradigma de comunicación basado en RPC. Un cliente envía uno o varios mensajes RPC request al servidor, este último responde a cada solicitud con un mensaje RPC reply asociado a través de un atributo XML ("message_id"). Esta mensajería puede utilizar por debajo cualquier protocolo de transporte que proporcione el conjunto de funcionalidades requeridas por el estándar. Netconf no se limita a ningún protocolo en si, permite que haya un mapeo a diversos mecanismos transporte para sus operaciones.

A continuación se detallan las características que el protocolo Netconf requiere para el protocolo de transporte subyacente.

- **Operación orientada a la conexión:** Netconf es orientado a la conexión, por ello requiere de una conexión permanente y confiable, y de la entrega de mensajes entre las partes en secuencia.

```
server --> cliente

<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
    <capability>urn:ietf:params:xml:ns:yang:ietf-inet-types?revision=2009-05-13</capability>
    <capability>http://acme.example.com/yang/acme-dns-resolver?revision=2009-08-12</capability>
  </capabilities>
  <session-id>232</session-id>
</hello>

cliente --> server

<nc:hello xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:capabilities>
    <nc:capability>urn:ietf:params:netconf:base:1.0</nc:capability>
  </nc:capabilities>
</nc:hello>

cliente --> server

<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1234">
  <nc:edit-config>
    <nc:target><nc:candidate/></nc:target>
      <nc:config>
        <dns xmlns="http://acme.example.com/yang/acme-dns-resolver">
          <resolver>
            <nameserver nc:operation="delete">
              <address>192.0.2.4</address>
            </nameserver>
            <nameserver nc:operation="create">
              <address>192.0.2.8</address>
              <port>5353</port>
            </nameserver>
          </resolver>
        </dns>
      </nc:config>
    </nc:edit-config>
</nc:rpc>

server --> cliente

<rpc-reply message-id="1234" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Código 1: Ejemplo de intercambio de capabilities y operación edit-config.

- **Autenticación, Integridad y Confianza:** Las entidades involucradas en una sesión Netconf asumen que se cumple con los niveles apropiados de autenticación y confianza, como por ejemplo, con el uso de TLS o SSH. El protocolo de transporte es el responsable de la autenticación del cliente al servidor, y viceversa, además del no repudio del emisor.

- **Protocolo de transporte obligatorio:** Una implementación Netconf tiene que soportar como protocolo de transporte a SSH, el mapeo se encuentra definido en el RFC 6242 [40].

2.5.2. Mensajes en modelo RPC

Como se ha mencionado anteriormente, Netconf utiliza un modelo de comunicación basado en RPC. Tanto el servidor como el cliente utilizan elementos <rpc> y <rpc-reply> para proporcionar un framework independiente del protocolo de transporte para los mensajes de solicitud y respuesta (de ahora en más, estos serán referidos en ese orden como request y reply).

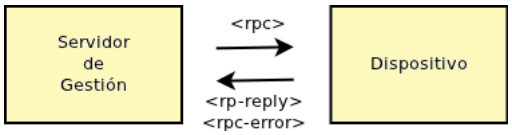

Figura 2.3: Modelo RPC.

- **<rpc>:** se utiliza para encerrar los request de Netconf que son enviados al servidor. Este tiene un atributo obligatorio, "message-id", un número que aumenta monótonamente en cada nuevo mensaje request y que tiene que estar incluido también en el mensaje de reply para poder asociarlo al request.

- **<rpc-reply>:** es la respuesta al mensaje <rpc>. Tiene al atributo "message-id" como obligatorio y es igual al enviado por request asociado.

- **<rpc-error>:** este elemento se envía en los mensajes <rpc-reply> en caso de que se haya producido algún error en el procesamiento del request <rpc>. En él se debe indicar la información necesaria para detectar el problema ocurrido, severidad, aplicación, tipo, etc. Si el servidor ha encontrado más de un error, éstos tienen que ser reportados de regreso al cliente utilizando múltiples elementos <rpc-error>.

- **<ok>:** se envía en el <rpc-reply> para informar al cliente que no ha ocurrido ningún error durante el procesamiento del request <rpc> asociado.

2.5.3. Operaciones del protocolo Netconf

El protocolo especifica un conjunto reducido de operaciones de bajo nivel para administrar la configuración de los dispositivos de red y obtener su información de

estado. Aunque el protocolo base proporciona operaciones para extraer, configurar, copiar y borrar los datastores de configuración, este puede ser extendido para soportar nuevas operaciones que son anunciadas por las partes como capabilities.

El protocolo Netconf base especifica las siguientes operaciones:

- <get-config>: esta operación se utiliza para extraer toda o una parte específica de la configuración de un datastore.
  - Parámetros:
    - source: es el datastore de configuración que se consulta
    - filter: es un filtro que se utiliza para obtener solamente la información que se requiere, si este no se encuentra presente se extrae toda la configuración del datastore. Puede ser descripto con elementos xml o con expresiones xpath. El último caso solamente aplica si ambas partes han anunciado :xpath capability durante el inicio de la sesión.
  - Respuesta positiva: Si el dispositivo ha procesado correctamente el request <rpc>, envía un mensaje <rpc-reply> incluyendo el elemento <data> con los resultados de la consulta.
  - Respuesta negativa: Uno o más elementos <rpc-error> se envían al cliente indicando que la operación solicitada no pudo ser procesada correctamente.
- <edit-config>: carga toda o parte de la configuración especificada al datastore destino indicado. Si el datastore destino no existe, el dispositivo debe crearlo.
  - Atributos:
    - Operación: indica el punto en la configuración (dentro de <config>) en que se tiene que realizar la operación de configuración deseada.
      - merge: los datos de configuración contenidos en el elemento con este atributo serán fusionados.
      - replace: reemplaza toda configuración relacionada en el datastore de configuración identificado por <target>. Si no existe el dato en el datastore es creado.

- ◇ create: los datos de configuración identificados por el elemento que contiene este atributo son agregados a la configuración si y sólo si no existen en el datastore, caso contrario se reporta un <rpc-error>.
- ◇ delete: sirve para borrar los datos de configuración asociados si y sólo si los datos existen, sino se reporta un <rpc-error>.
- ◇ remove: igual que el atributo anterior, con la diferencia de que si el dato de configuración no existe, la operación es ignorada por el servidor y no se reporta ningún error.

- Parámetros:
  - target: el nombre del datastore que será editado, tales como <running> o <candidate>.
  - default-operation: selecciona la operación por defecto para el request <edit-config>. Esta puede ser: "merge", "replace" o "none".
  - test-option: solamente se utiliza si el dispositivo informa como capability :validate. Este puede ser: *test-then-set*: realiza una validación antes de tomar la configuración. Si un error ocurre, no se procesa la operación <edit-config> y se retorna un <rpc-error>. *set*: el server toma la configuración sin realizar ninguna prueba primero. *test-only*: solamente hace la validación, sin modificar el datastore.
  - error-option: *stop-on-error*: aborta la operación de cambio de configuración cuando ocurre el primer error en su procesamiento. Este es el valor por defecto. *continue-on-error*: continua el procesamiento de la operación por más que se encuentre un error. El error se reporta al cliente. *rollback-on-error*: si algún error ocurre, el servidor para el procesamiento de la operación <edit-config> y restaura la configuración del datastore al valor inicial antes de recibir la operación. Esta funcionalidad tiene que ser reportada como capability como :rollback-on-error.
  - config: especifica de forma jerárquica el dato de configuración que se pretende modificar.

- Respuesta positiva: el server devuelve al cliente un mensaje <rpc-reply> incluyendo un elemento <ok>.

  - Respuesta negativa: se reporta el error en <rpc-error> por el cual no se pudo completar la operación.

- <copy-config>: crea o reemplaza un datastore de configuración completo por el contenido de otro datastore. Si el datastore destino no existe, éste se crea.

  - Parámetros:
    - target: el datastore destino.
    - source: el datastore origen.
  - Respuesta positiva: el server devuelve al cliente un mensaje <rpc-reply> incluyendo un elemento <ok>.
  - Respuesta negativa: se reporta el error en <rpc-error> por el cual no se pudo completar la operación.

- <delete-config>: borra la configuración del datastore de forma completa. El datastore <running> nunca puede ser borrado.

  - Parámetros:
    - target: el datastore destino al que se quiere borrar la configuración.
  - Respuesta positiva: si el dispositivo fue capaz de realizar la operación se devuelve al cliente un mensaje <rpc-reply> incluyendo el elemento <ok>.
  - Respuesta negativa: se reporta el error en <rpc-error> por el cual no se pudo completar la operación.

- <lock>: permite al cliente bloquear la configuración del datastore completamente. No permite que otros clientes, si es que se encuentran conectados también al dispositivo, puedan editar la configuración del datastore. La duración del bloqueo se mantiene desde su comienzo hasta que se invoca a la operación <unlock> o cuando la sesión Netconf se cierra.

  - Parámetros:
    - target: el nombre del datastore que se va a bloquear
  - Respuesta positiva: el server devuelve al cliente un <rpc-reply> conteniendo un elemento <ok>.

  - Respuesta negativa: se reporta el error en <rpc-error> por el cual no se pudo completar la operación.

- <unlock>: desbloquea el datastore específico que fue bloqueado anteriormente por la operación <lock>. Los parámetros y las respuestas son iguales a las de la operación <lock>.

- <get>: extrae la configuración que se encuentra corriendo en el dispositivo (running) y la información de estado del mismo.

  - Parámetros:

    - filter: especifica la porción de configuración de sistema o datos de estado que se quieren extraer. Si este parámetro no se encuentra presente, se retorna toda la información de configuración y estado del dispositivo.

  - Respuesta positiva: Si el dispositivo ha procesado correctamente el request <rpc>, envía un mensaje <rpc-reply> incluyendo el elemento <data> con los resultados de la consulta.

  - Respuesta negativa: Un o más elementos <rpc-error> se envían al cliente indicando que la operación solicitada no pudo ser procesada correctamente.

- <close-session>: solicita la terminación de la sesión Netconf de forma segura. El server cuando procesa esta operación tiene que liberar cualquier lock y recursos asociados a la sesión actual, para luego proceder con el cierre la conexión.

  - Respuesta positiva: el server devuelve al cliente un mensaje <rpc-reply> incluyendo el elemento <ok>.

  - Respuesta negativa: se reporta el error en <rpc-error> por el cual no se pudo completar la operación.

- <kill-session>: fuerza la terminación de una sesión Netconf. Si el servidor recibe esta operación, tiene que abortar cualquier operación en progreso, liberar los locks y cerrar la conexión asociada.

  - Parámetros:

- session-id: es el identificador de la sesión Netconf que se quiere terminar.

- Respuesta positiva: el server devuelve al cliente un mensaje <rpc-reply> incluyendo un elemento <ok>.

- Respuesta negativa: se reporta el error en <rpc-error> por el cual no se pudo completar la operación.

```
cliente --> server

     <rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <copy-config>
         <target>
           <running/>
         </target>
         <source>
           <url>https://user:password@example.com/cfg/new.txt</url>
         </source>
       </copy-config>
     </rpc>

server --> cliente

     <rpc-reply message-id="101"
         xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <ok/>
     </rpc-reply>

cliente --> server

     <rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <delete-config>
         <target>
           <startup/>
         </target>
       </delete-config>
     </rpc>

server --> cliente

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

Código 2: Ejemplo de operaciones copy-config y delete-config.

A su vez, se incluyen operaciones adicionales como extensiones a las operaciones base listadas anteriormente, y que son informadas por las entidades de red como capabilities soportadas luego del establecimiento de la conexión Netconf.

- <commit>: cuando la configuración en el datastore candidato esta completa, una operación <commit> provocará que la configuración en éste se copie al datastore que se encuentra actualmente corriendo (running).

- <discard-changes>: si el cliente decide no hacer una operación <commit>, la configuración del datastore candidato retorna a los valores del que se encuentra corriendo (running).

- <cancel-commit>: cancela la operación <commit> en proceso.

- <create-subscription>: habilita las notificaciones asincrónicas para reportar eventos ocurridos en el servidor.

- <validate>: verifica de forma completa la configuración por errores de sintaxis y semánticos antes de aplicar la configuración al dispositivo.

```
cliente --> server

     <rpc message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <lock>
         <target>
           <running/>
         </target>
       </lock>
     </rpc>

server --> cliente

     <rpc-reply message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <rpc-error> <!-- lock failed -->
         <error-type>protocol</error-type>
         <error-tag>lock-denied</error-tag>
         <error-severity>error</error-severity>
         <error-message>
           Lock failed, lock is already held
         </error-message>
         <error-info>
           <session-id>454</session-id>
           <!-- lock is held by NETCONF session 454 -->
         </error-info>
       </rpc-error>
     </rpc-reply>
```

Código 3: Ejemplo de operación lock no exitosa.

2.5.4. Capa de contenido

La capa de contenido de Netconf es un conjunto de definiciones de operaciones, datos de estado y configuración soportadas por el dispositivo, expresados en XML. El grupo de trabajo NETMOD (NETCONF Data Modeling Language) ha desarrollado un lenguaje de modelado de datos, llamado YANG, con el fin de definir de manera "amigable a los humanos" la sintaxis y la semántica de las operaciones sobre los datos, datos de configuración, notificaciones y operaciones del

servidor. En resumen, este lenguaje de modelado de datos describe cómo los datos son representados y accedidos. YANG se encuentra definido en el RFC 6020[14] y es acompañado por el RFC 6021[36] que define los tipos de datos (Common YANG Data Types).

Los datos de configuración, estado, notificaciones y operaciones del dispositivo se encuentran codificados en XML y tienen que cumplir con los esquemas YANG definidos para el mismo. Cabe destacar que las operaciones definidas por el RFC Netconf también se encuentran en YANG. Estos esquemas son las reglas que se debe seguir para organizar los datos en conjuntos de nodos que se estructuran en forma de árbol y jerárquicamente (ver Córdigo 4), especifican cuál es el tipo de datos de cada parámetro de configuración.

Como en XML, cada dato del árbol en YANG tiene su nombre, un valor y nodos hijos. La organización de las ramas de este árbol puede ser formada por módulos y submódulos que se incluyen para extender las funcionalidades y/o los parámetros de configuración soportados por las entidades de red. Con YANG, fabricantes de dispositivos de red y sistemas de gestión pueden ahora "hablar" el mismo idioma, permitiendo así la interoperabilidad entre equipos desarrollados por diferentes fabricantes.

En una solución típica basada en YANG y Netconf, el comportamiento del cliente como del servidor esta dirigido por los contenidos de los modulos YANG. El servidor incluye definiciones de los modulos como meta-data que se encuentra disponible en el motor Netconf.

La función del motor Netconf (ver Figura 2.4) es procesar los request que arriban al servidor, utilizar meta-data para parsear y verificar el mensaje de request, realiza la operación solicitada, y retorna los resultados al cliente, un <rpc-reply> con un elemento <ok> si la operación fue exitosa. En caso contrario, retorna un <rpc-error> indicando el o los errores ocurridos en el procesamiento.

Para utilizar el lenguaje YANG, sus módulos tienen que definir el modelo de los datos específicos del dispositivo a gestionar, para luego ser cargados, compilados y codificados en el servidor. Es probable que algunos entornos sea difícil la implementación de aplicaciones para parsear YANG. En esos escenarios, se definió una gramática XML para YANG llamada YIN (YANG Independent Notation). YIN permite el uso de parsers XML que se encuentran disponibles tanto como open source como comercialmente. Esta conversión de YANG a YIN es directa y completamente reversible. Un esquema de configuración se puede trans-

formar de YANG a YIN y de YIN a YANG sin que ocurra ninguna pérdida de ninguna definición en el modelo de datos, es decir, las declaraciones en YANG son convertidas en elementos XML, preservando la estructura y contenidos de YANG.

Como los contenidos Netconf son codificados en XML, es natural la utilización de lenguajes de esquemas para su validación. Para facilitar esta tarea, YANG ofrece un mapeo estándar de los módulos YANG a DSDL (Document Schema Definition Languages), en el cual se destaca la amplia utilización de Relax NG. DSDL es considerado como la mejor opción de lenguaje estándar para esquemas debido a que se analiza, no solamente la gramática y tipos de datos de los documentos XML, sino también las restricciones de semántica y reglas para modificar la información. Además ofrece la capacidad de coordinar multiples esquemas independientes y especificar cómo estos se deben aplicar a las diferentes partes de los documentos. En el apéndice A se desarrollan los pasos necesarios para crear esquemas YANG y DSDL para validar configuraciones de datastores para servidores DHCP.

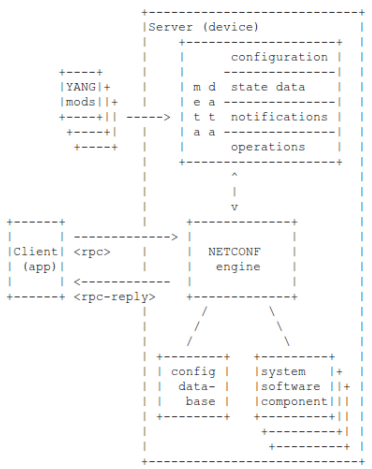

Figura 2.4: Arquitectura típica de servidor Netconf [38]

La secuencia de eventos en una interacción típica cliente/servidor Netconf puede ser la siguiente [38]:

- La aplicación del cliente ([C]) abre una sesión Netconf con el servidor (dispositivo) ([S]).
- [C] y [S] intercambian mensajes <hello> incluyendo la lista de capabilities

soportadas por cada entidad. Éstas permiten al [C] aprender de los módulos soportados por el [S].

- [C] construye y envía una operación definida en un módulo YANG, codificada en XML, dentro del elemento Netconf <rpc>.

- [S] recibe y parsea el elemento <rpc>.

- [S] verifica y valida que los contenidos del request contra el modelo de datos definidos por el módulo YANG.

- [S] ejecuta la operación solicitada, posiblemente cambiando la configuración de algún datastore de configuración. Este paso depende de la validación exitosa realizada con anterioridad.

- [S] construye la respuesta indicando si la operación ha sido exitosa o no, pudiendo incluir datos solicitados o detalles de error.

- [C] recibe y parsea el elemento <rpc-reply>.

- [C] analiza la respuesta y la procesa de manera necesaria.

```
module example-ospf {
    namespace "http://example.org/netconf/ospf";
    prefix ospf;

    import network-types {  // Access another module's def'ns
        prefix nett;
    }

    container ospf {   // Declare the top-level tag
        list area {    // Declare a list of "area" nodes
            key name;  // The key "name" identifies list members
            leaf name {
                type nett:area-id;
            }
            list interface {
                key name;
                leaf name {
                    type nett:interface-name;
                }
                leaf priority {
                    description "Designated router priority";
                    type uint8;  // The type is a constraint on
                                 // valid values for "priority".
                }
                leaf metric {
                    type uint16 {
                        range 1..65535;
                    }
                }
                leaf dead-interval {
                    units seconds;
                    type uint16 {
                        range 1..65535;
                    }
                }
            }
        }
    }
}
```

```
  <ospf xmlns="http://example.org/netconf/ospf">

    <area>
      <name>0.0.0.0</name>

      <interface>
        <name>ge-0/0/0.0</name>
        <!-- The priority for this interface -->
        <priority>30</priority>
        <metric>100</metric>
        <dead-interval>120</dead-interval>
      </interface>

      <interface>
        <name>ge-0/0/1.0</name>
        <metric>140</metric>
      </interface>
    </area>

    <area>
      <name>10.1.2.0</name>

      <interface>
        <name>ge-0/0/2.0</name>
        <metric>100</metric>
      </interface>

      <interface>
        <name>ge-0/0/3.0</name>
        <metric>140</metric>
        <dead-interval>120</dead-interval>
      </interface>
    </area>
  </ospf>
```

Código 4: Ejemplo de modelo de datos YANG y configuración XML asociada para un router.

Capítulo 3

Arquitectura y diseño del servidor de gestión de configuración

3.1. Arquitectura general y funcionalidades

La gestión de configuración permite a los operadores realizar cambios en los parámetros de configuración de aquellos elementos que conforman su red, como por ejemplo, routers, switches, servidores, etc. Sin un servidor para el manejo de configuración, se hace difícil la implementación de los cambios de configuración, se se complicada su documentación, como también su validación. Todas estas dificultades se agravan cuando la red esta conformada por gran cantidad de equipos y existen varios operadores trabajando sobre ella simultáneamente. Con esto, los propósitos principales de la gestión adecuada de las configuraciones de dispositivos son el ahorro de tiempo y la reducción de errores en la red debido a mala modificación de las mismas. Incluso en el caso de que los cambios causen errores, la implementación de una solución para el manejo de las configuraciones permite a los operadores resolverlos de manera más eficiente y rápida.

Netconf Element Management System (NEMS) es un servidor de gestión de configuración de dispositivos que soportan como protocolo de gestión a Netconf [22]. Este proporciona un entorno amigable y flexible para realizar los cambios de configuración, operaciones en datastores y multi-operador. La Figura 3.1 muestra la arquitectura de la solución propuesta e implementada, con sus componentes y la relación entre ellos. NEMS cuenta con una interfaz de usuario proporcionada

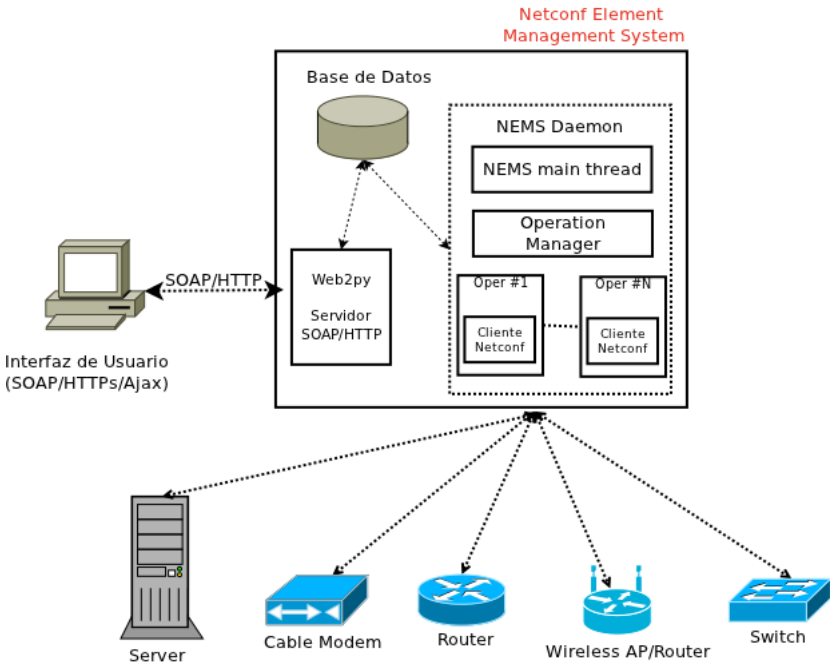

Figura 3.1: Arquitectura del NEMS.

por un servidor HTTP (aunque la comunicación utiliza HTTPS), la lógica de la aplicación, el gestor de operaciones (u operation manager), el módulo para los clientes Netconf y la base de datos:

- **Interfaz de usuario:** es una interfaz gráfica web (HTML) accedida a través de un navegador de internet que permite al usuario interactuar con el servidor y las configuraciones de los dispositivos que soportan Netconf. Está basada en Javascript (Jquery) y AJAX para la manipulación de los elementos HTML y efectos gráficos.

- **Servidor HTTP:** proporciona los servicios necesarios utilizados por la interfaz de usuario. La conexión entre el browser y el servidor se encuentra encriptada (HTTPs), soportada por certificados digitales (Apéndice C explica los pasos necesarios para la creación de certificados en entorno Linux Ubuntu). A su vez, algunos procedimientos ejecutados en el navegador utilizan SOAP (sobre AJAX) para la verificación de flags (por ejemplo, estado de los locks de edición de dispositivos) y obtención de información de la base de datos. Tanto el servidor como la interfaz de usuario están basadas en Web2py, un entorno de desarrollo web de código abierto y programado en lenguaje Python. Éste utiliza Rocket WSGI web server pero también se

puede configurar para que funcione con servidores desarrollados por terceros, como Apache, Cherokee o Lighttpd. Se responsabiliza de la creación de las tablas en la base de datos y de los triggers asociados, de la autenticación y autorización de los usuarios.

- **Thread principal**: se encarga de la administración general de todo el sistema. Se ocupa de la ejecución, monitoreo y terminación del entorno Web2py [32] (interfaz de usuario y servidor HTTP) y del gestor de operaciones (manejo de señales del sistema operativo a los procesos). Crea los loggers utilizados por el operation manager y el código de ejecución de las operaciones para logear los eventos del servidor. A su vez, verifica la correcta creación de las tablas en la base de datos.

- **Gestor de operaciones**: se encarga de recibir las notificaciones desde la base de datos para la creación, ejecución y cancelación de las operaciones sobre los datastores y configuraciones de los dispositivos.

- **Operaciones**: es el procesamiento de las operaciones del servidor. Se encargan de establecer, mantener y terminar las sesiones Netconf. Este componente esta basado en la librería de código abierto Ncclient [11]. También, se responsabilizan del procesamiento de los mensajes Netconf y procesamiento de las configuraciones representadas en formato XML.

- **Base de datos**: se ocupa de guardar la información de manera persistente, consistente y segura. La solución fue implementada con un servidor Postgresql.

El servidor fue programado por completo en el lenguaje Python [8] debido a que es adecuado para el desarrollo de aplicaciones web, manejo de XML, GUI, base de datos y scripting, pero la razón principal de su elección fue a su reducida curva de aprendizaje en comparación a otros lenguajes, es fácil de aprender para aquellas personas que no poseen mucha experiencia en programación.

El módulo de código principal (main) del programa ejecuta Web2py como un proceso nuevo hijo, el operation manager en un nuevo hilo de ejecución dentro del proceso del módulo principal, al igual que el procesamiento de las operaciones y clientes Netconf.

De ahora en adelante el término operaciones hace referencia a las operaciones propias del servidor de gestión NEMS y no aquellas asociadas al protoco-

lo Netconf, a no ser que se indique explícitamente. Una operación del servidor puede incluir una o más operaciones Netconf, un ejemplo es la operación OP_EDIT_CONFIG del servidor NEMS que involucra las operaciones Netconf <lock>, <edit-config> y <unlock>. La descripción de cada una de las operaciones soportadas por NEMS será desarrollada en 3.4.1.

Las características y funcionalidades soportadas por el servidor son las siguientes:

- Interfaz gráfica de usuario basada en web (browser Google Chrome).
- Base de datos para el almacenamiento de información de dispositivo y configuraciones (Postgresql 9.x).
- Conexión segura a través de HTTPS.
- Autenticación y autorización de usuarios. Dos niveles de usuario: Administrator y Operator.
- Creación, edición y borrado de tipos de dispositivos.
- Creación, edición y borrado de dispositivos.
- Operaciones sobre datastores de dispositivos
  - OP_COPY
  - OP_DELETE
  - OP_REQ_CURRENT_CONF
- Operaciones sobre configuraciones en datastores de dispositivos, encapsulados en operación OP_EDIT_CONFIF.
  - agregar, borrar y editar contenedores de configuración
  - agregar, borrar y editar elementos de configuración
- Validación de configuraciones contra esquemas de configuración (Relax NG, Schematron y DSDL).
- Protocolo de gestión Netconf 1.0 (Ncclient).

Requerimientos básicos para el funcionamiento del servidor son:

- Python 2.6 .
- Browser Google Chrome.
- Servidor de base de datos Postgresql 9.x con soporte de plpythonu.
- Librerías externas.
  - yang2dsdl: script escrito en Python para transformar modulos YANG en esquemas DSDL y validar instancias de datastores de configuración. Esta herramienta está incluida dentro de la solución Pyang [7].
  - ncclient: librería en Python que facilita la programación de cliente y desarrollo de aplicaciones alrededor del protocolo Netconf [11].
  - psutil: es un módulo Python que proporciona una interface para obtener información de los procesos que se encuentran corriendo en el sistema (CPU, utilización de disco y memoria) [5].
  - psycopg2: es un adaptador para conexiones a bases de datos Postgesql basada en Python [6].
  - lxml: librería para la manipulación de documentos XML y HTML [1].
  - netifaces: librería para manejo de interfaces de red [3].

Además de las librerías y módulos mencionados anteriormente, se utilizó Google Code y SVN (Subversion) como repositorio y sistema de control de versionado de código, y Eclipse como editor. Se creó el proyecto bajo el url:

`http://code.google.com/p/netconf-element-management-system/`.

En un principio, se había desarrollado la interfaz de usuario para ser operada con el navegador Firefox, pero tres razones llevaron a la elección de Google Chrome como browser requerido. Una de ellas fue la existencia de problemas de Firefox para el manejo de imágenes declaradas con posición float. Esto causaba que las imágenes de los íconos o botones de la interfaz de usuario se mostraran desplazadas, llevando a la necesidad de modificar manualmente los márgenes en los archivos .CSS de la librería JQUERY (UI). A su vez, con el mismo navegador se enfrentaron varios problemas de procesamiento de respuestas AJAX [10], con tipos de datos de retorno XML. Por último, la obtención de los mejores resultados

en performance en el cargado de la página y procesamiento de mensajes SOAP y AJAX en el navegador Google Chrome.

3.2. Interfaz gráfica

La interfaz gráfica permite al usuario del servidor NEMS operar sobre las configuraciones de los dispositivos y datastores de manera fácil y segura, para ello se utilizó Web2py como entorno de desarrollo de la aplicación web. El manejo de elementos HTML y procesamiento de instrucciones fueron implementados en Javascript y Jquery. Este último es un tipo de librería para Javascript que simplifica la busqueda de elementos en DOM, la gestión de eventos, animaciones e interacciones Ajax. La estructura de la página web y su estilo fueron definidos en CSS. Las librería utilizadas para el desarrollo de la GUI son listadas a continuación:

- jquery.form: plugin para Jquery para submitir formularios HTML sobre Ajax.
- jquery.jstree: plugin para Jquery para la creación y manipulación de árboles (o trees).
- jquery.cookie: plugin para Jquery para la manipulación de cookies del lado del browser.
- soapclient: librería para transmitir mensajes SOAP sobre AJAX.
- jquery.jqgrid: plugin para Jquery para la creación y manipulación de grillas (o tablas).

3.2.1. Menú principales

Para acceder a NEMS se tiene que utilizar Google Chrome como navegador web, la estructura general de la interfaz gráfica se muestra en la Figura 3.2.

En la parte superior izquierda se encuentra el menú de usuario, éste proporciona los vínculos (links) necesarios para la gestión de información del mismo:

1. Es el nombre de usuario que se encuentra registrado actualmente en el servidor.

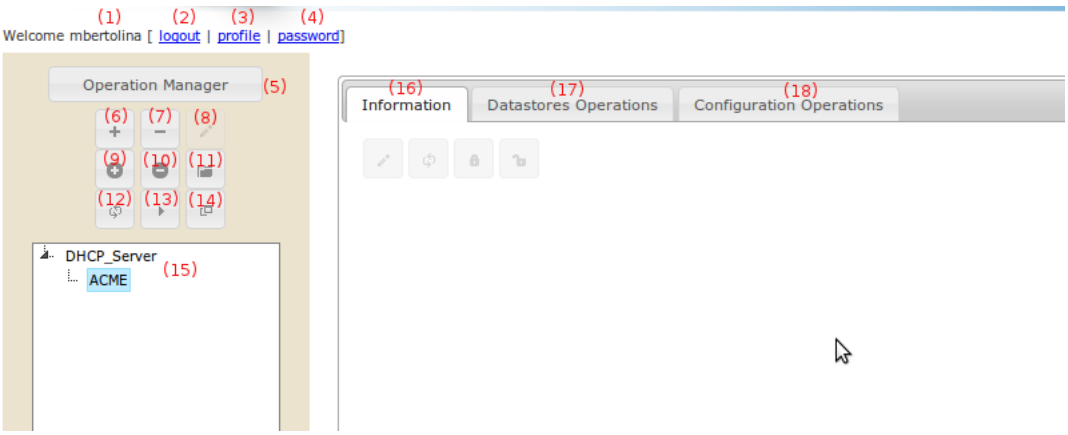

Figura 3.2: Disposición general de la UI.

2. *logout*: se utiliza para salir de la aplicación y de-registrar al usuario propietario de la sesión actual.

3. *profile*: se utiliza para modificar los datos de usuario, como su nombre, apellido y correo electrónico.

4. *password*: el usuario puede entrar a este vínculo para cambiar el password configurado para acceder al servidor.

En el margen izquierdo se ubican los botones para el acceso al operation manager, los controles de tipos de dispositivo, creación, modificación y borrado de dispositivos, selección de dispositivos y acceso directo a interfaz web para modificación manual de base de datos.

5. *Operation Manager*: se encarga de abrir el diálogo donde se encuentra la grilla para manipulación de operaciones.

6. *Add Device*: se utiliza para la creación de nuevos dispositivos.

7. *Delete Selected Device*: borra el dispositivo seleccionado en el árbol dispositivos (15).

8. *Edit Selected Device*: con éste se abre el diálogo de edición de información de dispositivo.

9. *Add Device type*: se utiliza para la creación de nuevos tipos de dispositivos. No puede existir ningún dispositivo si antes no se crea un tipo de dispositivo al cual asociarlo.

10. *Delete Selected Device Type*: elimina el tipo de dispositivo seleccionado en la topología de dispositivos (15) y con él todos los elementos de red asociados.

11. *Edit Selected Device Type*: abre el diálogo para la modificación de los esquemas para la validación de configuraciones.

12. *Refresh Topology*: actualiza la topología de dispositivos (15).

13. *Load Device Information*: carga la información del dispositivo en el browser y prepara el servidor para trabajar con sus datastores y configuraciones.

14. *Appadmin Page*: accede a una interfaz gráfica proporcionada por Web2py para la manipulación manual de la base de datos, se utiliza especialmente para la registración de nuevos usuarios.

15. *Topology*: topología donde se asocian en forma de árbol los tipos de dispositivos y los elementos de red asociados a éstos.

16. *Information Tab*: presenta la información del dispositivo seleccionado con (13) y provee funcionalidades para edición, refresco y edición de la misma, bloqueo de dispositivo para poder operar sobre él. Ningún usuario puede operar sobre un dispositivo si éste último no se encuentra bloqueado y el propietario del lock no es quien desea trabajar con el mismo.

17. *Datastores Operations Tab*: este tab provee las operaciones para manipular datastores: get config, copy config, upload config y delete datastore.

18. *Configuration Operations Tab*: proporciona el árbol de configuración de los datastores ubicados en el dispositivo y todos los métodos para su edición (eliminación, edición y creación de contenedores y parámetros de configuración).

3.2.2. Operation Manager

Operation manager es un diálogo clave para el manejo de operaciones que serán ejecutadas en los dispositivos. Éste proporciona una grilla donde se puede observar información de las operaciones que fueron creadas, procesadas o en progreso. El menú inferior proporciona las funcionalidades básicas para la gestión de éstas:

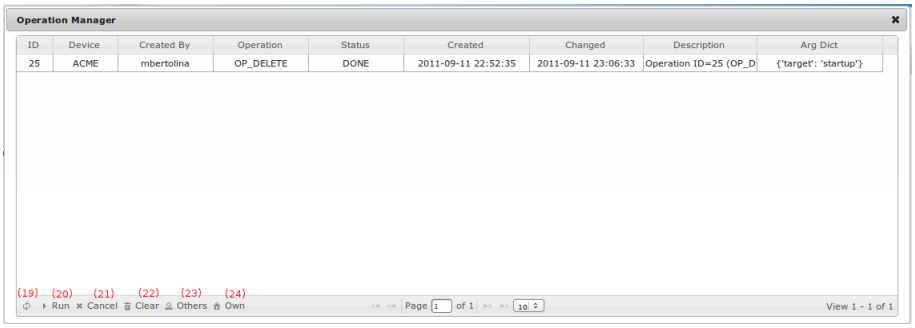

Figura 3.3: Operation Manager.

19. *Refresh*: actualiza la tabla de operaciones.

20. *Run*: informa el deseo del usuario de ejecutar la operación seleccionada. Ver diagrama de estado de las operaciones Figura 3.9. Produce el cambio de estado de las operaciones a INPROGRESS.

21. *Cancel*: cancela o anula las operaciones que se encuentran en estado PENDING.

22. *Clear*: limpia de el operation manager aquellas operaciones cuyo estado se encuentra en NEW, DONE, CANCELLED y FAILED.

23. *Others*: si el usuario registrado posee el rol de Operator podrá solamente observar aquellas operaciones que fueron generadas por otros usuarios. El cambio del estado de las mismas se encuentra restringido a aquellos usuarios definidos como Administrator.

24. *Own*: muestra solamente las operaciones que fueron creadas por el usuario actual registrado.

3.2.3. Information Tab

Este tab provee los elementos necesarios para la edición de la información general del dispositivo:

25. *Edit Selected Device*: tiene la misma funcionalidad que el (8).

26. *Refresh Device Information*: actualiza la información del dispositivo cargado.

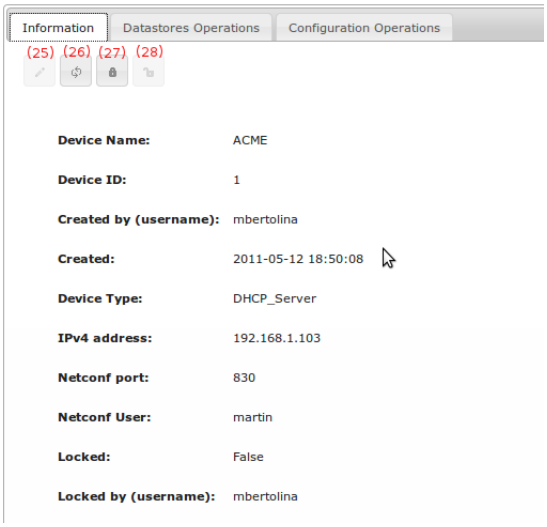

Figura 3.4: Information Tab.

27. *Lock*: bloquea el dispositivo para editar su información general, realizar operaciones sobre sus datastores o editar datos de configuración.

28. *Unlock*: desbloquea el dispositivo bloqueado por el usuario registrado.

3.2.4. Datastore Operations Tab

La Figura 3.5 presenta la interfaz para generar las operaciones relacionadas a los datastores del dispositivo de red.

29. *Get Config*: extrae la configuración actual de un datastore específico desde dispositivo y la almacena en la base de datos. Este datastore puede ser uno existente en la base de datos o uno nuevo. Esta operación se utiliza mayormente cuando el dispositivo ha sido creado recientemente y no existen configuraciones asociadas a éste en la base de datos.

30. *Copy Config*: se utiliza para copiar de manera total la configuración de un datastore a otro. El datastore fuente debe existir en la base de datos, mientras que el destino no. En caso de que el destino no exista en la base de datos y la operación es exitosa, se crea uno asociándolo a dicho elemento de red.

31. *Upload Config*: esta funcionalidad permite enviar al dispositivo una configuración específica y completa de un datastore desde definida en archivo XML proporcionado por el usuario.

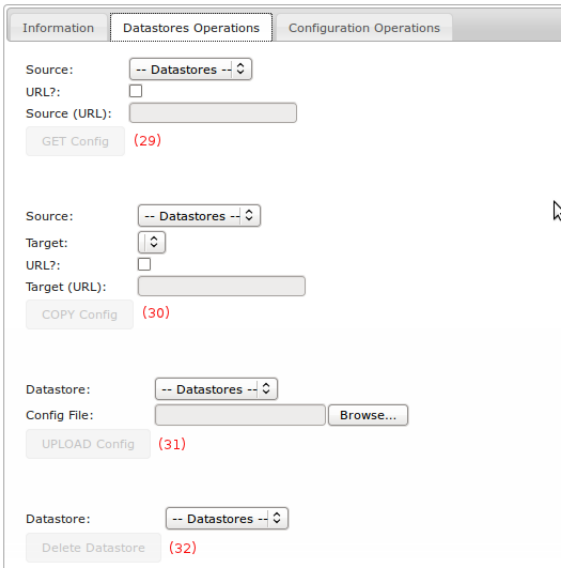

Figura 3.5: Datastore Operations Tab.

32. *Delete Datastore*: elimina la configuración del datastore de manera total. El datastore tiene que haber sido cargado con anterioridad en la base de datos.

Cada operación generada en este tab se agrega al operation manager para su administración.

3.2.5. Configuration Operations

En este tab se encuentran todas las herramientas necesarias para la manipulación de configuraciones de aquellos datastores que hayan sido cargados en la base de datos (utilizando *Get Config* del tab de operaciones en datastores). En éste se puede observar: el menú de operaciones sobre configuración, el árbol de configuración del dispositivo (Figura 3.6), la grilla de elementos de configuración (leafs) y la grilla de mapeo de prefijos a URL de namespaces XML (Figura 3.7).

33. *Datastore*: selecciona el datastore cuya configuración se desea observar o modificar.

34. *Reload*: carga el árbol de configuración con la última configuración existente en la base de datos asociada datastore seleccionado.

35. *Request Current Config*: genera una operación para extraer desde el dispositivo la última configuración del datastore seleccionado asociado. Es similar

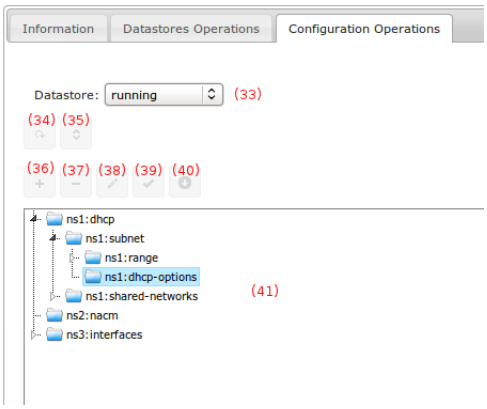

Figura 3.6: Configuration Operations Tab.

a (29). La operación creada se agrega al operation manager para su administración.

36. *Add Container*: agrega un nodo contenedor de elementos de configuración en el contenedor seleccionado en el árbol. Los contenedores están formados de elementos de configuración que son presentados en la grilla de elementos de configuración (leafs) (42).

37. *Delete Container*: borra el contenedor de configuración seleccionado en el árbol.

38. *Edit Container*: se utiliza para editar el tag XML del contenedor seleccionado.

39. *Validate and Commit*: valida la configuración modificada contra los esquemas de configuración asociados al tipo de dispositivo. En caso exitoso, se realiza un proceso de commit de la configuración modificada para luego ser enviada al elemento de red.

40. *Download Configuration*: envía la configuración modificada, previamente habiendo realizado los procesos de validación y commit correspondientes. Genera una operación OP_EDIT_CONFIG que se agregada al operation manager.

41. *Árbol de contenedores*: en este se organizan de manera jerárquica los contenedores de configuración asociados al datastore del dispositivo.

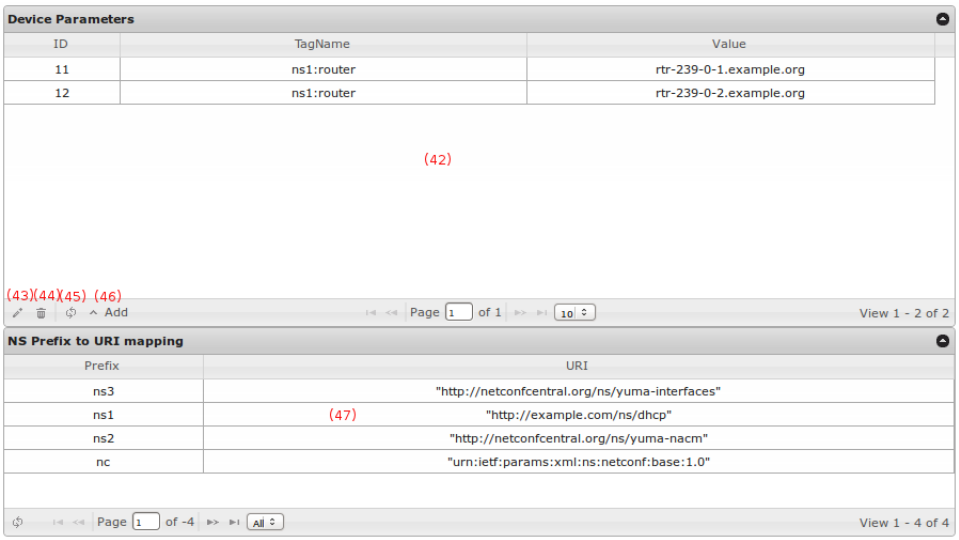

Figura 3.7: Grilla de elementos de configuración (leafs) y grilla de mapeo de prefijos a URL de namespaces.

42. *Grilla de elementos*: en ésta se presenta la información de los elementos leaf de configuración correspondientes al contenedor seleccionado en el árbol de contenedores (41).

43. *Edit element*: edita el tag XML y el valor del elemento seleccionado en la grilla.

44. *Delete element*: borra el elemento de configuración seleccionado en la grilla.

45. *Refresh*: actualiza la grilla con los valores valores actuales y modificados.

46. *Add element*: agrega un elemento de configuración dentro del contenedor de configuración seleccionado en el árbol.

47. *Namespace URL to prefix maping*: presenta el mapeo de los namespaces XML todos los elementos y contenedores de configuración del datastore seleccionado a los prefijos XML.

3.3. Arquitectura de la base de datos

NEMS utiliza para el almacenamiento de la información y datos de configuración un servidor de base de datos Postgresql. La elección de este poderoso sistema

relacional orientado a objetos y código abierto está fundamentada principalmente en su capacidad para la generación y envío de notificaciones a las conexiones activas registradas para recibirlas, la implementación de Python como lenguaje de programación nativo en la base de datos y el soporte de tipos de datos XML. Aunque la implementación actual del NEMS no soporta el tipo de datos XML para guardar y manipular la configuración de los dispositivos, ésta es una buena funcionalidad para ser incorporada próximamente, cuando nuevos métodos para procesar documentos XML sean implementados por el grupo de desarrollo de Postgresql.

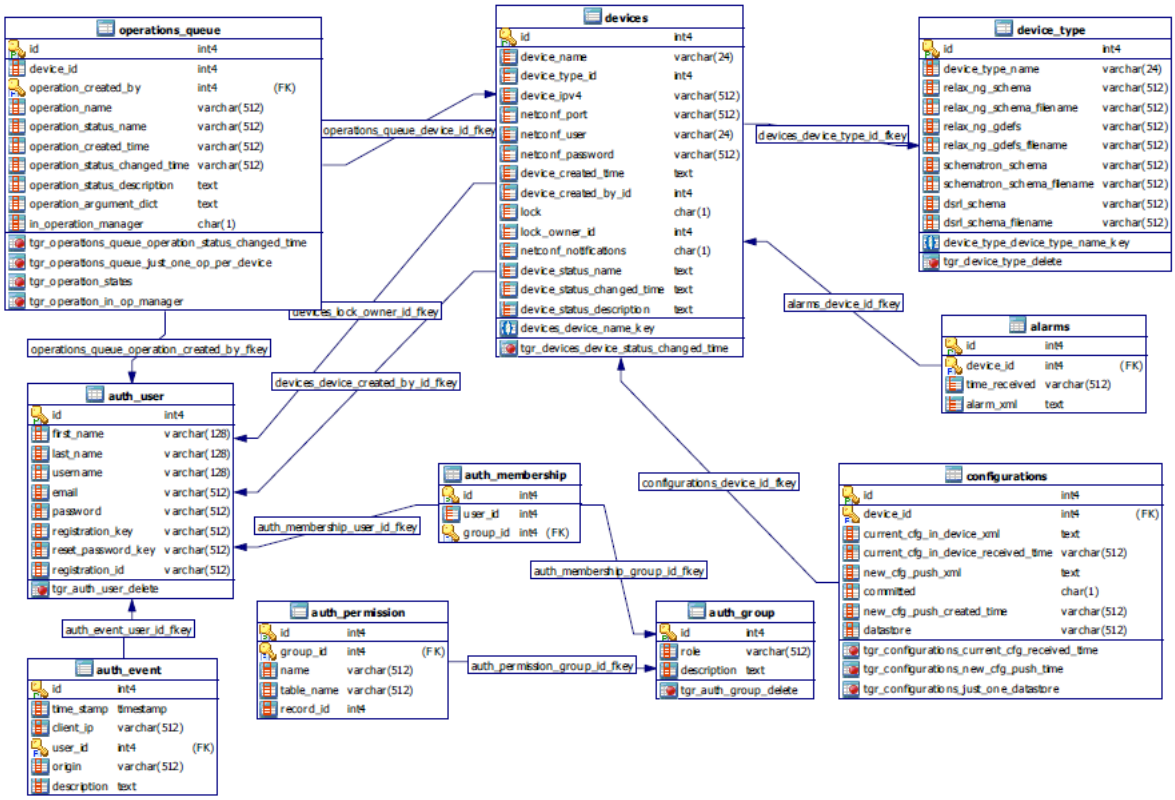

Figura 3.8: Arquitectura de la base de datos.

Para el acceso a la base de datos desde Web2py se utilizó su capa de abstracción de base de datos (Database Abstraction Layer - DAL). Es una API que mapea objectos Python en objectos de la base de datos, como son las queries, las tablas y los records.

NEMS daemon genera las conexiones y cursores a la base de datos con la librería Psycopg2, ésta es un adaptador para Python específico para acessos a

servidores Postgresql. Las ventajas principales tenidas en cuenta para su incorporación a la solución NEMS fueron el soporte a las especificaciones Python DB API 2.0 y la capacidad para funcionar con hilos de manera segura (thread safe donde los hilos pueden compartir conexiones). Esto significa que Psycopg2 fue diseñado para aquellas aplicaciones que trabajan con una gran variedad de hilos donde numerosos cursores que ejecutan numerosos INSERTS y UPDATES concurrentes se crean y se destruyen.

La Figura 3.8 muestra el diseño de la base de datos con sus tablas, tipos de datos, índices, llaves y relaciones entre ellas. También se encuentran definidas las funciones ejecutadas automáticamente por los disparadores (triggers).

Las tablas definidas son:

- **auth_user**: almacena la información de los usuarios que tienen acceso a la aplicación, su nombre, apellido, nombre de usuario, password, dirección de email, status (registration pending, accepted, blocked).
- **auth_group**: almacena la información de grupos o roles para los usuarios en una estructura many-to-many. Un grupo esta identificado por un rol y una descripción. NEMS define dos roles: Administrator y Operator. Administrator es grupo que posee todos los privilegios en el acceso a la información y ejecución de operaciones. Operator es totalmente restringido a todas las operaciones, posee solamente acceso de lectura a la información de los dispositivos y configuraciones.
- **auth_membership**: vincula los usuarios con los grupos en una estructura many-to-many.
- **auth_permission**: asocia los grupos con los permisos de acceso a las tablas. Un permiso está identificado por un nombre, y opcionalmente, una tabla o un record. Por ejemplo, miembros de cierto grupo pueden tener permisos para realizar un UPDATE en un record específico de una tabla.
- **auth_event**: registra los accesos a la interfaz de usuario y los cambios en las tablas de usuarios y grupos.
- **device_type**: almacena la información necesaria para realizar la validación de la configuración de los dispositivos asociados a cada tipo de dispositivo. En ella se encuentran las referencias a los archivos de esquemas guardados en disco.

- **devices**: guarda información de los dispositivos, su nombre, tipo de dispositivo (definido en la tabla device_type) e información para establecer la sesión Netconf (dirección ip, puerto, usuario, password). También se especifican dos flags, por un lado, *lock* para informar si el dispositivo se encuentra bloqueado para su edición (y quien es el usuario propietario de ese bloqueo), tanto para la modificación de información en esta tabla, como también cambios en configuraciones en los datastores y ejecución de operaciones. El flag *netconf_notificacions* indica si las notificaciones de eventos asincrónicos Netconf se encuentran habilitadas en el dispositivo (actualmente este no es utilizado, se tuvo en cuenta futuras implementaciones).

- **alarms**: almacena información de los eventos asincrónicos Netconf recibidos por la aplicación cuando las notificaciones se encuentran habilitadas en el dispositivo.

- **configurations**: guarda información de las configuraciones de los datastores (current_cfg_in_device_xml) y de la nueva configuración (new_cfg_push_xml) para ser enviada en el caso de que se hayan realizado cambios en la actual. El flag *commited* indica que una nueva configuración existe en el la columna new_cfg_push_xml, que fue procesada y validada contra los esquemas del tipo de dispositivo asociado y lista para ser enviada al elemento de red.

- **operations_queue**: esta tabla almacena todas las operaciones sobre los datastores y configuraciones de los elementos de red que serán ejecutadas por la aplicación. Especifica el nombre, el dispositivo asociado, su estado (NEW, PENDING, INPROGRESS, CANCELLED, FAILED), cuando fue creada y cuando cambió de estado, y los parámetros necesarios para su ejecución. El flag *in_operation_manager* indica si esta operación es visible o no por la interfaz gráfica de usuario (en el operation manager).

Una buena parte de las condiciones de operación del servidor NEMS y de las restricciones de la base de datos están definidas por los triggers declarados en cada tabla. Éstos tienen la capacidad de ejecutar funciones asociadas cuando existen cambios en la tabla (antes o después, de modificaciones en las filas y de borrar y agregar nuevas) y limitar dichas modificaciones de acuerdo a ciertos requerimientos. Los disparadores asociados a las tablas son los siguientes:

- Tabla devices:

- *tgr_devices_device_status_changed_time*: actualiza el valor de device_status_changed_time cada vez que el estado del dispositivo cambia de estado, el mismo puede ser CONNECTED (cuando una sesión Netconf se encuentra activa) y DISCONNECTED.

- Tabla device_type:

  - *tgr_devices_type_delete*: se ejecuta cuando un tipo de dispositivo se elimina. Verifica que ninguno de los dispositivos asociados al tipo de dispositivo esté bloqueado (flag lock de la tabla devices), lo que significa que el dispositivo esta siendo editado por algún usuario. En el caso que encuentren dispositivos bloqueados, la operación de eliminación de tipo de dispositivo se aborta y la fila asociada no se modifica, lo mismo sucede si existen operaciones en el operation manager. Esta última restricción se agrega para que no se pueda borrar ningún tipo de dispositivo si tiene asociado elementos de red con operaciones ejecutándose o por ser procesadas por el servidor, existentes en el operation manager.

- Tabla auth_user:

  - *tgr_auth_user_delete*: se dispara cada vez que un usuario es eliminado. Verifica que éste no posea ningún dispositivo bloqueado. Dicha restricción se agrega para que no queden dispositivos bloqueados por usuarios que ya no existen. En caso de que esto sucediera, el dispositivo quedaría bloqueado para siempre, provocando que usuario existente no pueda bloquear de nuevo el dispositivo para operar sobre él. En dicha situación, debido a que el lock se encuentra asignado a un usuario que ya no existe, cualquier usuario Administrador podría acceder a appadmin de NEMS para modificar manualmente la base de datos y desbloquear el dispositivo.

- Tabla auth_group:

  - *tgr_auth_group_delete*: se dispara cuando un grupo es eliminado. Verifica que todos los usuarios asociados a éste no posea ningún dispositivo bloqueado, al igual que trg_auth_user_delete.

- Tabla configurations:

- *tgr_configurations_current_cfg_received_time*: actualiza el tiempo en que se recibió la configuración actual del dispositivo.
- *tgr_configurations_new_cfg_push_time*: actualiza el tiempo en que se se validó y se realizó el proceso de commit de la nueva configuración a enviarse al dispositivo.
- *tgr_configurations_just_one_datastore*: cada vez que un nuevo datastore es agregado a la base de datos y asociado a un dispositivo, el trigger verifica que no exista otro datastore con el mismo nombre en esta tabla.

- Tabla operations_queue:

  - *tgr_operations_queue_operations_status_changed_time*: actualiza el tiempo cada vez que se modifica el estado de la operación.
  - *tgr_operations_queue_in_op_manager*: controla los cambios de estado del flag *in_operation_manager* y envía notificaciones a las conexiones activas de la base de datos cada vez que una operación se elimina (limpia) o agrega al operation manager.
  - *tgr_operations_queue_just_one_op_per_device*: se dispara cada vez que una nueva operación se agrega a esta tabla. Se veridica que exista solamente una operación asociada a un dispositivo. Esta restricción se agrega para que no haya dificultades en la ejecución de las operaciones, por ejemplo, con esta restricción el usuario no puede ejecutar un OP_DELETE sobre un datastore completo y luego otro un OP_EDIT_CONFIG sobre el mismo datastore. Este trigger permite que las operaciones a los dispositivos de red se ejecuten de manera serial (o en secuencia) y en el orden en que fueron invocadas.
  - *tgr_operations_queue_states*: verifica que los cambios en los estados sean de acuerdo a la Figura 3.9 y que en cada uno se envíe una notificación a las conexiones registradas para escuchar esas modificaciones. Cada una de dichas notificaciones indica la operación en la que se modificó el estado.

Los cambios de estado de las operaciones permitidos están ilustradas en le Figura 3.9. Cuando las operaciones se crean se les asocia el estado NEW. PENDING indica que el propietario de la operación (el owner) desea ejecutarla (1). Cuando

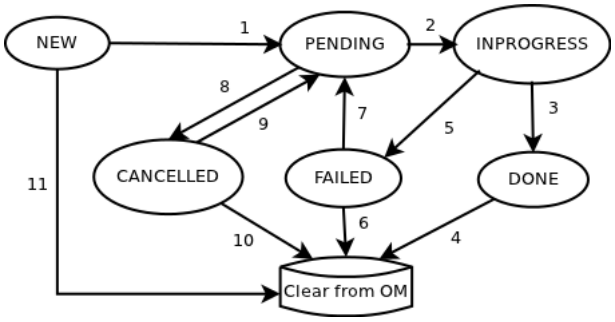

Figura 3.9: Posibles cambios de estado de las operaciones.

el servidor empieza el procesamiento de una operación PENDING, cambia su estado a INPROGRESS (2) para informar que la misma está siendo ejecutada. DONE indica que la operación fue procesada con éxito (3). En caso de ocurrir algún error o inconveniente en el procesamiento de la operación por parte del servidor, el estado de la operación cambia a FAILED (5), en esta situación se le da la oportunidad al usuario de volver a ejecutar dicha operación, cambiando su estado a PENDING (7). Si el operador se arrepiente de la ejecución de una operación y ésta se encuentra en PENDING, puede cancelarla y el estado de la operación se modifica a CANCELLED (8). Siempre y cuando esta última no haya sido eliminada del operation manager, se la puede volver a ejecutar (9). En caso en que la operación haya sido procesada con éxito (DONE), fallida (FAILED), cancelada (CANCELLED) o recién creada(NEW), puede eliminarse (limpiar) del operation manager, corresponde a (4), (5), (10) y (11).

Eliminar o limpiar una operación del operation manager significa que la misma dejará de mostrarse en en su tabla, pero esto no involucra una eliminación del record asociado a esta operación en la base de datos. Solamente se setea a false el flag *in_operation_manager_queue*. Todas las operaciones se eliminan de la base de datos sólo cuando los usuarios que las generaron son eliminados.

3.4. Aplicación, operation manager y operaciones

El servidor NEMS comienza su funcionamiento con la ejecución del código principal nems_main.py. Éste se ocupa de validar el archivo de configuración ./cfg/Cfg.xml contra el esquema ./cfg/Cfg.xsd, para luego cargarlo en memoria. Esta

validación es necesaria para verificar que los parámetros de configuración que el usuario debe especificar se encuentren definidos correctamente para permitir un adecuado funcionamiento. En el caso de ocurrir un error en el procesamiento del archivo Cfg.xml será informado en consola y se terminará la ejecución del servidor NEMS. Dichos parámetros se encuentran listados en la Tabla 3.1.

Parámetro	**Descripción**
LOG_PATH	Directorio donde se crearán los archivos de logs. Especificar el path completo desde el root (/).
DEBUG_LEVEL	Nivel de logging usado por los loggers, es un número entero. CRITICAL = 50, ERROR = 40, WARNING = 30, INFO = 20, DEBUG = 10, NOTSET = 0.
WEB2PY_PATH	Directorio donde se encuentra el ejecutable web2py.py. Especificar el path completo desde el root (/).
WEB2PY_PORT	Puerto utilizado para acceder a la página web a través del browser. Si éste se encuentra vacío, se utiliza el 8000 como por defecto.
WEB2PY_IP_ADDRESS	Especifica la dirección IP a la que accederán los usuarios. Si no se define ninguno se toma 127.0.0.1 por default, pero sólo puede ser accedido por la máquina donde reside el servidor NEMS.
WEB2PY_PASSWORD	Password necesario para acceder la interfaz appadmin.
WEB2PY_SSL_CERTIFICATE_PATH	La ubicación del certificado digital para utilizar en HTTPS. Por ejemplo: /tmp/NEMS/cfg/server.crt
WEB2PY_SSL_PRIVATE_KEY_PATH	Ubicación de la clave privada. Por ejemplo: /tmp/-NEMS/cfg/server.key.insecure
POSTGRESQL_IP_ADDRESS	Dirección IP del servidor base de datos Postgresql. Si no se especifica, se utiliza 127.0.0.1 como default.
POSTGRESQL_PORT	Puerto para conectarse a la base de datos. Si no se especifica, se utiliza 5432 como default.
POSTGRESQL_USERNAME	Nombre de usuario utilizado para conectarse a la base de datos.
POSTGRESQL_PASSWORD	Contraseña para conectarse a la base de datos.
POSTGRESQL_DATABASE	Nombre de la base de datos utilizada por el servidor NEMS.

Cuadro 3.1: Parámetros de configuración del servidor NEMS en archivo cfg.xml

NEMS utiliza la librería estándar Python *logging* para generar los loggers (en nems_main.py) encargados de registrar en archivos la información de ejecución y eventos ocurridos en el servidor. Estos mensajes se guardan en el directorio especificado por el parámetro de configuración LOG_PATH en los siguientes archivos: NEMS_Logger.log (para información general de funcionamiento del servi-

dor), Operation_Manager_Logger.log (para mensajes del Operation Manager) y Operations_Logger.log (datos de ejecución de operaciones específicas).

A su vez, nems_main.py es responsable de la ejecución y control de estado del hilo Operation Manager y del entorno Web2py. Se definen las señales de sistema que la aplicación podrá recibir para la terminación segura de esos dos componentes y del servidor mismo (SIGTERM y SIGHUP).

El hilo operation manager se encarga de la ejecución y control de las operaciones del servidor NEMS, su funcionamiento se encuetra definido en el archivo nems_operation_manager.py. Éste registra su conexión a la base de datos para recibir todas las notificaciones relacionadas a los cambios de estado de las mismas y toma las acciones correspondientes para cada una de ellas. Es responsabilidad del operation manager el establecimiento de la conexión a la base de datos y provisión de cursores a los hilos de operación para poder procesarlas. Con el arribo de cada notificación de cambio de estado de operaciones a PENDING, éste instancia la clase correspondiente a la operación solicitada para que el procesamiento de la misma se realice en un hilo nuevo (como muestra la Figura 3.1). El ciclo de vida de cada hilo de operación comienza con el arribo de una notificación y culmina con la terminación de dicha operación.

Todas las operaciones soportadas por el servidor se encuentran definidas en clases en el archivo nems_operations.py y heredan de la clase *operation* todas sus funcionalidades. Esta última se encarga de definir los atributos y métodos comunes que todas las operaciones deben compartir para su procesamiento y ejecución. El operation manager antes de instanciar la clase correspondiente a la operación requerida, verifica que la clase para la operación solicitada se encuentre definida en el archivo nems_operations.py. El nombre de la operación en la base de datos tiene que ser ser igual al nombre de la clase correspondiente. Las operaciones soportadas por el servidor de gestión de configuraciones NEMS son:

- *OP_REQ_CURRENT_CONF*
- *OP_EDIT_CONFIG*
- *OP_COPY*
- *OP_DELETE*

3.4.1. Operaciones

OP_REQ_CURRENT_CONF

Esta operación es la primera a invocada luego de la creación de los dispositivos. Permite obtener la configuración actual que se encuentra ejecutando en el datastore específico en el dispositivo de red y cargarla en la base de datos para su procesamiento. Si no se realiza este proceso luego de la creación del elemento de red en el servidor,los usuarios van a disponer ninguna configuración para manipular.

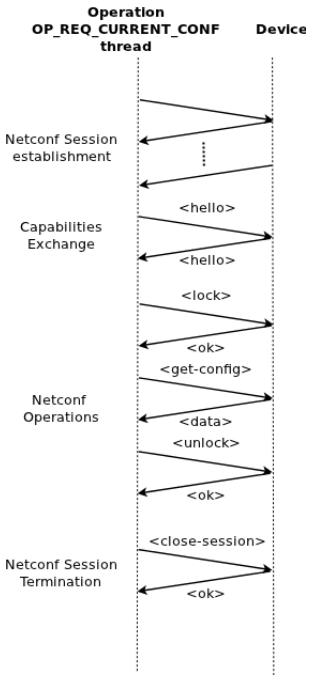

Figura 3.10: Operación OP_REQ_CURRENT_CONF.

La Figura 3.10 ilustra la mensajería y operaciones Netconf asociadas a esta operación. Primero se establece la sesión Netconf y se produce el intercambio de capabilities mediante mensajes <hello>. En caso que el datastore desde el que se desea obtener la configuración es "candidate", "startup" o "url", se verifica que las capabilities asociadas a estos hayan sido informadas por el dispositivo, *:candidate*, *:startup* y *:url*. Los datastores identificados por *url* son aquellos que no son running, startup, ni candidate, como por ejemplo archivos en disco, donde url indica la ubicación del mismo.

Si la verificación de las capabilities es exitosa se procese al bloqueo del datastore utilizando la operación Netconf <lock>. Esta última se invoca para que

ningún otro agente o cliente Netconf, si existe, pueda modificar la configuración en el momento en que el servidor NEMS solicita la misma. Si el bloqueo es exitoso, se procede con la ejecución de la operación <get-config> para obtener toda la información almacenada en el datastore del dispositivo.

En el caso que el datastore asociado al dispositivo no exista en la base de datos del servidor NEMS, se crea uno nuevo con los datos obtenidos del dispositivo en formato XML. Si existe un datastore en la base de datos con el mismo nombre, se actualiza la configuración con la última obtenida desde el elemento de red. Para terminar la operación del servidor, se desbloquea el datastore con un request <unlock> y se procede a terminar la sesión con <close-session>.

OP_COPY

Esta operación se encarga de copiar la configuración en forma completa desde un datastore a otro en el mismo dispositivo. La Figura 3.11 muestra el intercambio de mensajes para la ejecución de la misma. Luego del establecimiento de la sesión Netconf y el intercambio de capabilities, se verifica, como en la operación anterior, que los datastores especificados estén soportados por el servidor de acuerdo a las funcionalidades intercambiadas en los mensajes <hello>.

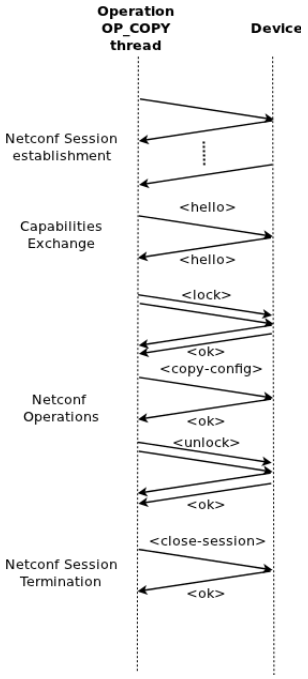

Figura 3.11: Operación OP_COPY.

Si dicha verificación es exitosa, se bloquean los dos datastores asociados (fuente y destino) y se procede a ejecutar la operación Netconf <copy-config>. Si ningún error ocurre, se desbloquea los datastores con una solicitud <unlock> y se termina la sesión con una <close-session>. La operación del servidor termina copiando el contenido del datastore fuente en la base de datos al destino correspondiente. En caso de que este último no exista en la base de datos, se crea uno y se lo asocia al dispositivo de red.

Es requerimiento que el datastore fuente exista en la base de datos antes de que esta operación se ejecute. Esto permite que el usuario sea consciente que el datastore origen existe y disponga de acceso a la información de configuración que se copiará al otro datastore.

OP_DELETE

La ejecución de esta operación es similar a OP_COPY, excepto que se invoca a la operación Netconf <delete-config>. Es requerimiento que el datastore, cuya configuración se desea eliminar, exista en la base de datos. En caso de que la operación sea exitosa, se borra el datastore de la base de datos.

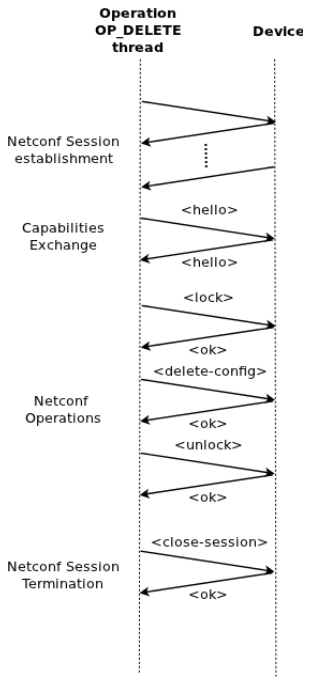

Figura 3.12: Operación OP_DELETE.

OP_EDIT_CONFIG

Ésta es la principal operación del servidor de gestión de configuraciones de dispositivos. OP_EDIT_CONFIG permite crear, modificar y borrar los elementos de configuración en un datastore específico de manera segura y elegante.

Es requerimiento previo para ejecutar esta operación, haber realizado una validación y un commit de la configuración modificada por el usuario. La *validación* contra los esquemas de datastore, asociados al tipo de dispositivo, permite verificar que los elementos de configuración que se enviarán al dispositivo sean aquellos que éste espera recibir. Se realiza el chequeo de la sintaxis y semántica de toda la configuración del datastore. *Commit* permite que la configuración de todo el datastore, previamente validado con éxito, sea cargada en la base de datos para ser utilizada por la operación OP_EDIT_CONFIG. Estos procesos de validación y commit se realizan en entorno Web2py con ayuda de yang2dsdl y librerías de procesamiento de documentos XML (lxml y Python minidom).

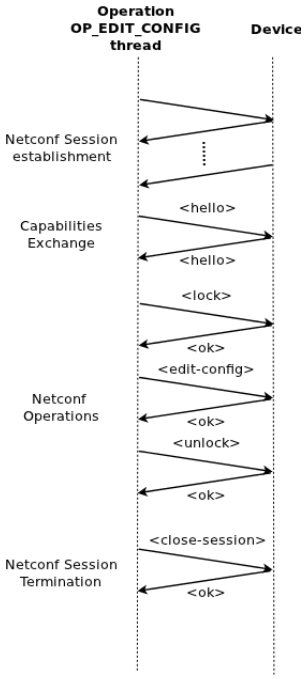

Figura 3.13: Operación OP_EDIT_CONFIG.

La Figura 3.13 muestra el flujo de mensajes Netconf asociados a esta operación. Primero se establece la sesión Netconf y se produce el intercambio de capabilities mediante mensajes <hello>. Si el datastore al que se desea modificar la configuración es “candidate”, “startup” o “url”, se verifica que las capabilities

asociadas a estos hayan sido informadas por el dispositivo, *:candidate*, *:startup* y *:url*. En caso de que el datastore destino sea "running" se verifica que la capability *:writtable-running* esté soportada por el elemento de red. No todos los dispositivos permiten la modificación de la configuración running, por ello se sugiere realizar la modificación al datastore candidate con la operación OP_EDIT_CONFIG y luego ejecutar una operación OP_COPY desde el candidate al running.

Si la verificación de las capabilities es exitosa se procese al bloqueo del datastore utilizando la operación Netconf <lock>, para luego ejecutar <edit-config>. La solicitud <edit-config> lleva en ella la configuración resultante de la modificación realizada por el usuario, de su validación contra los esquemas de configuración de tipo de dispositivo y de haber sido cargada en la base de datos por la operación commit.

Si la operación es exitosa, la nueva configuración enviada al dispositivo se carga en la base de datos como la configuración actual del datastore asociado.

3.5. Web2py environment

Web2py es un entorno web de código abierto diseñado para el desarrollo de aplicaciones web dirigidas por base de datos (database driven applications), está escrita en Python y es programable también en Python. Web2py es un entorno full-stack, lo que significa que contiene todos los componentes necesarios para construir aplicaciones web funcionales [33]. Está basada en WSGI (Web Server Gateway Interface), que define una interfaz simple y universal entre los servidores web y las aplicaciones web (o entornos web).

Web2py utiliza el patrón de desarrollo de software Model View Controller (MVC), separando la representación de los datos (el modelo) de los de presentación (view) y de la lógica de la aplicación (el controlador). También, este entorno proporciona librerías para ayudar el diseño, implementar y probar cada una de dichas partes por separado, y hacerlas funcionar todas en conjunto.

Web2py incluye una capa de abstracción para acceso a bases de datos, Database Abstraction Layer (DAL). Ésta genera comandos SQL dinámicamente para que el desarrollador no tenga que hacerlo en forma manual. La capa mencionada soporta conexiones a una gran variedad de bases de datos, especialmente PostgreSQL, que utilizado en la implementación del servidor NEMS.

La Figura 3.14 muestra el procesamiento de un request HTTP. El servidor

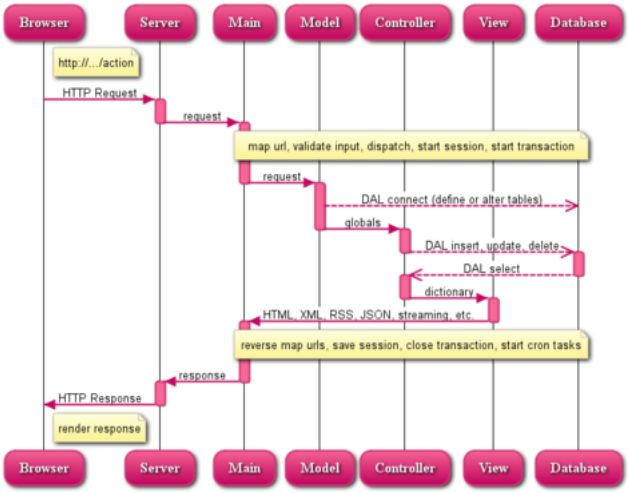

Figura 3.14: Modelo Model View Controller y procesamiento de HTTP resquests en Web2py. [33]

puede utilizar el servidor HTTP incluido en Web2py (Rocket WSGI server) o un servidor de terceros como Apache. La aplicación *main* de Web2py realiza todas las tareas de procesamiento de solicitudes, manejo de cookies, sesiones, transacciones, enrutamiento de URL y enrutamiento reverso, y ejecución de lógica de la aplicación. Web2py puede manejar más de una aplicación web, cada una de éstas está formada por modelos (models), vistas (views) y controladores (controllers). En el caso de la figura, las líneas punteadas representan la comunicación con los motores de base de datos. El dispatcher mapea los URLs solicitados a funciones definidas en los controladores, estas procesan la solicitud y retornan la respuesta para ser presentada por la vista. En el caso del servidor NEMS, Web2py no funciona siempre de esta manera ya que también se hacen solicitudes a funciones expuestas como servicios SOAP.

Las funcionalidades de la GUI y definición de la estructura de la base de datos están definidas por NEMS_Web, una aplicación Web2py para ser utilizada dentro de su entorno de desarrollo y operación. La estructura de la aplicación está dada por:

- *models*: describen la representación de los datos de las tablas en la base de datos y la relación entre ellas. También, define define funciones ejecutadas por los triggers (*db.py*).
- *controllers*: describen la lógica de la aplicación y work-flow. *default.py* define todas las funciones asociadas a solicitudes HTML y XML, mientras que el

archivo *soap_services.py* proporciona una gran variedad de funciones como servicios SOAP.

- *views*: en este directorio se encuentran las definiciones de la manera en que presenta la interfaz de usuario en el navegador web. El layout de la aplicación se encuentra especificada en archivos HTML y XML.
- *static*: en éste se encuentra la definición de la estructura y el estilo de la página web del servidor NEMS en archivos CSS (Cascading Style Sheets) y también incluye las librerías Javascript (archivos con extensión .js).
- *sessions*: almacena información relacionada a cada usuario registrado.
- *databases*: almacena información adicional de las tablas definidas en la base de datos.
- *Squemas*: en este directorio se guardan los esquemas RELAX NG utilizados para la validación de las configuraciones de los dispositivos.

3.6. Autenticación y Autorización

Web2py incluye un mecanismo potente y configurable de control de acceso basado en roles. Dentro de una organización, se crean roles asociados a diversas funciones y trabajos. Los permisos de acceso para realizar ciertas operaciones están asignadas a roles específicos. Los usuarios son asignados a roles particulares, y a través de la asignación de esos roles, adquieren los permisos para realizar las funciones específicas en el sistema. Dado a que los permisos no son asignados directamente a cada usuarios, sino que solamente los adquieren a través de los roles, la gestión de los derechos de los usuarios individuales se convierte en una cuestión de asignar simplemente roles a usuarios.

NEMS soporta dos roles de operación para ser asignadas a los usuarios: *Operator* y *Administrator*. Los usuarios asignados al rol Operator son aquellos que poseen los permisos mínimos para operar el servidor, poseen solamente acceso de lectura a la información de los dispositivos y de la configuración de los datastores cargados en la base de datos. Administrator es el rol que dispone los permisos para realizar todas las operaciones disponibles en el servidor (ejecución de operaciones a datastores, modificación de configuraciones, bloqueos de dispostiivos, etc). Además, los usuarios asignados como Administrator, podrán dar de alta y

completar el proceso de registración de nuevos usuarios, tanto para roles Operator como Administrator. Estos permisos están representados en la habilitación de los botones de operación en la interfaz gráfica como también su verificación en la ejecución de funciones definidas en los controladores.

Por defecto se crean dos usuarios, con username y password: *operator/operator* y *administrator/administrator*. El rol Operator se asigna al primero y Administrator al segundo. Se recomienda, luego de la instalación y puesta a punto del servidor, cambiar las contraseñas asociadas. Estos deben que existir siempre en la base de datos y se sugiere no borrarlos.

3.7. Implementación de nuevas funcionalidades

Hay dos componentes que pueden ser modificados para incorporar nuevas funcionalidades al sistema de gestión de configuraciones NEMS, por un lado la interfaz de usuario y por el otro la lógica de la aplicación.

Todas las operaciones que se pueden realizar en la interfaz de usuario se encuentran programadas en la aplicación NEMS_Web bajo el entorno Web2py. Para modificar la vista, o el template de la página (layout), se puede editar el archico nems_layout.html que se encuentra dentro del directorio views. Todas las interacciones, animaciones y efectos están programados en Javascript con ayuda de Jquery, y se pueden encontrar en nems.js, dentro del directorio static/js. El estilo de la página está definido en archivos .css, en static/css.

Para definir nuevas tablas o realizar cualquier otra modificación en la base de datos, se debe editar db.py dentro del directorio models. Las funciones que se ejecutan para la procesar las solicitudes de HTML/XML y SOAP están definidas en los controladores default.py y soap_services.py respectivamente.

Las funcionalidades de NEMS Server pueden ser extendidas incorporando nuevas operaciones como clases en el archivo nems_operation.py. La definición de dichas clases debe heredar los atributos y métodos de la clase *operation* y el nombre utilizado debe ser igual al nombre de la operación que será ejecutada por el operation manager. Por ejemplo, el procesamiento de la operación "OP_COPY" se encuentra definido en la clase "OP_COPY" cuya clase super o madre es *operation* (referirse al archivo nems_operations.py para más detalle).

El código se encuentra disponible en Google Code bajo la licencia GNU GPL v3.

`http://code.google.com/p/netconf-element-management-system/`.

Capítulo 4

Funcionamiento del Servidor y Operación Básica

4.1. Entorno de pruebas

Todas las funcionalidades que el servidor Netconf Element Management System soporta fueron verificadas exitosamente utilizando netconfd (versión 1.15-3) como servidor Netconf. Este se encuentra incluido en las herramientas YUMA (YANG-Based Unified Modular Automation) [13], archivos YANG rigen completamente su comportamiento, proporciona una interfaz a base la datos interna de dispositivo robusta y segura, y utiliza operaciones estándar del protocolo Netconf. En el Apéndice B se describe la configuración del servidor Netconfd y se muestran los mensajes de debug en consola interactuando con el servidor de gestión NEMS.

El servidor NEMS fue instalado en una laptop HP Pavilion dv 9825nr corriendo sobre Linux Ubuntu 11.04 32-bits, con Python 2.7.1+ y servidor de base de datos Postgresql 9.0. Las librerías Python instaladas y sus versiones fueron las siguientes:

- ncclient v0.3
- psutil v0.3.0
- psycopg2 v2.4.2
- lxml v2.3.1
- netifaces v0.3

Todas las pruebas se realizaron utilizando el modelo de datos YANG para servidores DHCP, la generación de esquemas para la validación de las configuraciones en datastores se encuentra descripta en el Apéndice A.

4.2. Login, registración de nuevos usuarios y appadmin

Para acceder a la interfaz de usuario del servidor NEMS se tiene que utilizar el navegador Google Chrome. El url a utilizar depende de la dirección y puerto asignado en el archivo de configuración del mismo, tiene la siguiente forma: `https://[NEMS_ip_address]:[NEMS_port]`.

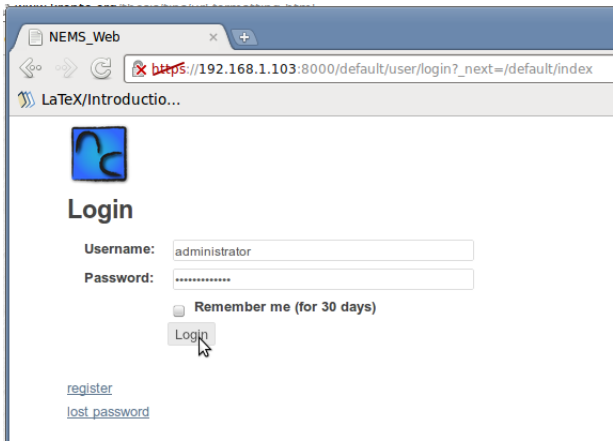

Figura 4.1: Página de Login.

Por defecto se encuentran creados los usuarios *operator/operator* y *administrator/administrator*, dentro de los grupos *Operator* y *Administrator* respectivamente. Cuando se ingresa por primera vez con estos usuarios, es conveniente cambiar los datos de perfil y contraseña. Para hacerlo, el usuario registrado debe dirigirse a los vínculos del menú de usuario, *profile* y *password* (Ver Figura 4.2).

Nuevos usuarios pueden registrarse para acceder a la aplicación (Figura 4.3), este proceso se puede realizar desde la página de login. Éstos necesitan que algún usuario perteneciente al grupo Administrador les permita el acceso y los asigne al grupo deseado.

La interfaz del servidor NEMS incluye acceso a la appadmin (application admin). Desde ella los usuarios registrados como Administradores pueden tener acceso a las tablas definidas en la base de datos de forma fácil a través de una

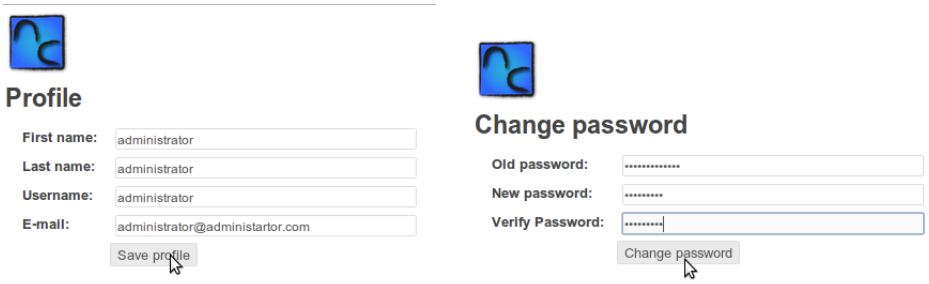

Figura 4.2: Página de perfil de usuario y cambio de contraseña.

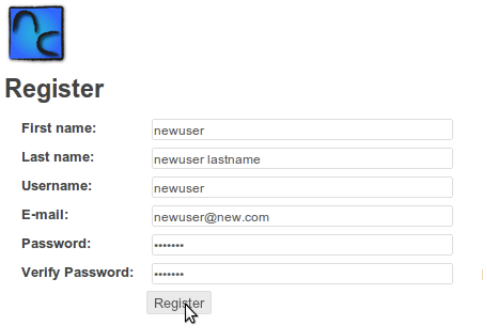

Figura 4.3: Página de registración de nuevos usuarios.

página web (Figura 4.4). Desde allí se pueden crear, modificar y borrar cualquier record en la base de datos. Se sugiere sumo cuidado en la utilización de esta herramienta.

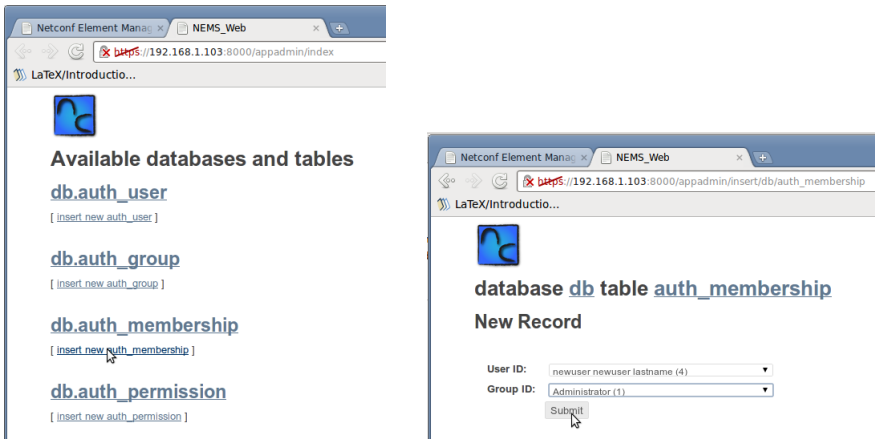

Figura 4.4: Acceso a tablas en base de datos con appadmin.

Una vez que el usuario ha ingresado los datos en el formulario de registración, el mismo queda disponible para que se le dé alta en el sistema. Para ello, un administrador debe ingresar a appadmin, asignar al usuario al grupo correspondiente

(Figura 4.4) y desbloquearlo para que pueda acceder a la aplicación (borrar el valor *pending* en la columna registration_key de la tabla auth_users, Figura 4.5).

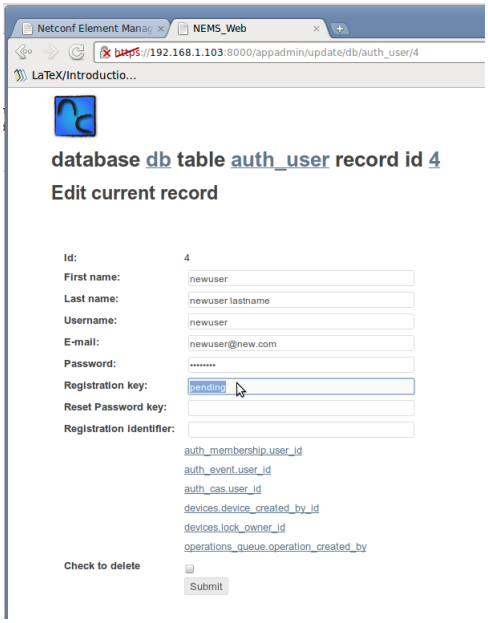

Figura 4.5: Desbloqueo de usuarios con appadmin.

En caso que un administrador desee bloquear el acceso a un usuario al servidor NEMS sin borrarlo, debe agregar la palabra *blocked* en la columna registration_key de la tabla auth_users.

4.3. Creación de nuevos dispositivos

Para la creación de nuevos dispositivos se requiere la existencia del tipo de dispositivo para asociarlos. Los tipos de dispositivo sirven para agrupar a dispositivos con características iguales y definir los esquemas de configuración contra los cuales se validan todas las configuraciones de los datastores de todos los mismos (Figura 4.6). En Apéndice A se desarrolla los pasos necesarios para la creación de estos esquemas DSDL utilizando yang2dsdl [7].

Para crear los dispositivos, se tiene que seleccionar el tipo de dispositivo en la topología, seguido por el botón *Add Device* y llenar el formulario con la información correspondiente solicitada en el diálogo de creación de elementos de red (Figura 4.7).

Una vez que el dispositivo ha sido creado, el usuario puede operar sobre éste de acuerdo a sus permisos de acceso. Con el botón *Load Device Information* se

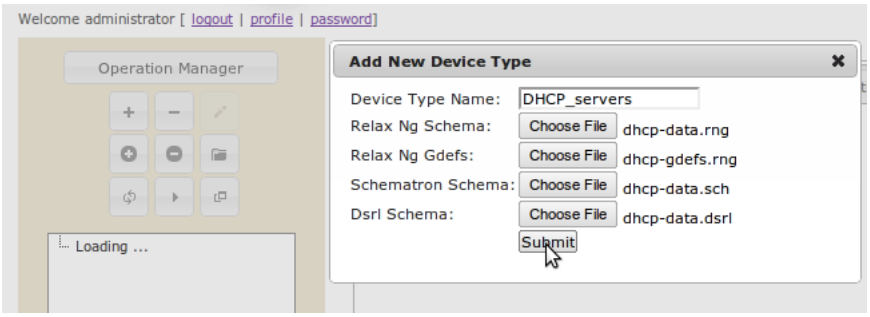

Figura 4.6: Creación de tipo de dispositivos.

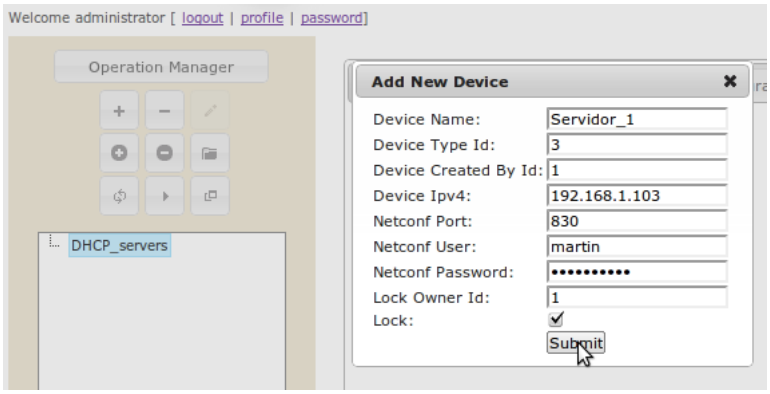

Figura 4.7: Creación de dispositivos.

carga en pantalla la informaición del elemento de red para comenzar a trabajar, esto se ilustra en la Figura 4.8.

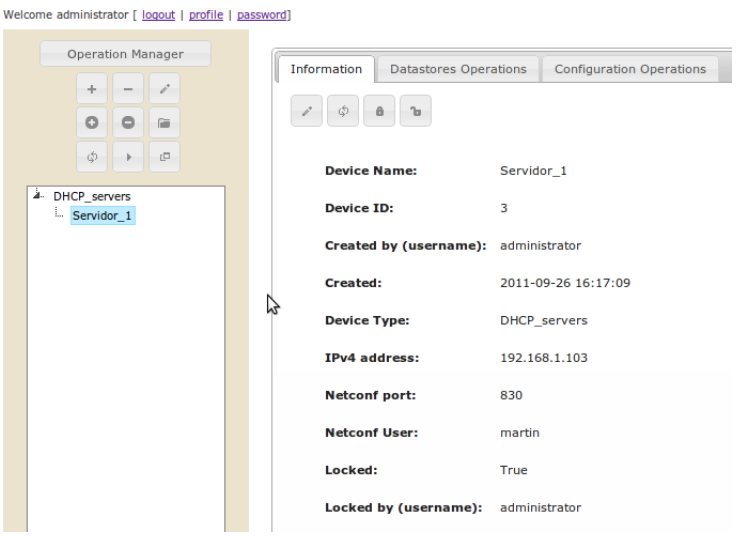

Figura 4.8: Cargado de información de dispositivos para operación.

El servidor solamente permite la operación en aquellos elementos de red que se encuentren bloqueados por el usuario registrado en el momento de operación. Esto asegura que ningún otro usuario trabaje sobre el mismo dispositivo simultáneamente. Solamente los usuarios asignados al grupo Administrator pueden realizar dichos bloqueos. Una vez que el dispositivo se encuentra bloqueado, el usuario podrá modificar la información del dispositivo (utilizando el botón *Edit Selected Device*) y ejecutar cualquier operación, tanto sobre los datastores (en el tab Datastores Operations) como las configuraciones de los datastores (en el tab Configuration Operations).

Para eliminar los dispositivos de la topología en árbol, se deben seleccionar y presionar el botón *Delete Device*. Si el mismo no se encuentra bloqueado por ningún usuario, éste y toda su información asociada se eliminará de la base de datos.

Se procede de igual manera para la eliminación de los tipos de dispositivo, aunque ésta depende de ciertas restricciones. Por un lado no el tipo de dispositivo no tiene que poseer ningún dispositivo asociado en estado bloqueado, y por el otro, que ninguno de estos contenga operaciones en el Operation Manager. Eliminar un tipo de dispositivo involucra el borrado de toda la información asociada al tipo dispositivo y todos los dispositivos asociados a éste, con sus datos, informaciones de configuración y esquemas DSDL para la validación de configuraciones.

4.4. Operaciones sobre configuraciones

El el Capítulo 3 se mostró la interfaz de usuario para las operaciones en los datastores (Sección 3.2.4) y se desarrolló la manera en que se realiza el procesamiento de cada una de ellas (Sección 3.4.1) ubicadas en el tab *Datastore Operations*. En esta sección se explican las operaciones que se pueden ejecutar sobre las configuraciones de cada datastore del dispositivo de red.

Las configuraciones de los elementos de red representados en XML cuentan con 3 componentes o elementos esenciales: los *containers*, los *leafs* y los *valores*. Cabe mencionar que esta terminología es propia del servidor NEMS y que tiene cierta relación con el lenguaje de modelado de datos YANG. Los containers son elementos XML que contienen uno o más elementos dentro. Leafs son elementos que solamente contienen el valor del elemento asociado. El valor, en terminología DOM (Document Object Model), es de tipo de nodo TEXT_NODE. Tanto los

leaf como los containers se diferencian por sus tags XML (caracteres dentro de < y >), attributos y namespaces.

```
1     <subnet>
2       <net>10.254.239.0/27</net>
3       <range>
4         <dynamic-bootp/>
5         <low>10.254.239.10</low>
6         <high>10.254.239.20</high>
7       </range>
8       <dhcp-options>
9         <router>rtr-239-0-1.example.org</router>
10        <router>rtr-239-0-2.example.org</router>
11      </dhcp-options>
12      <max-lease-time>1200</max-lease-time>
13    </subnet>
```

Código 5: Identificación de containers, leafs y valores.

Código 5 muestra un ejemplo de una porción de configuración de un datastore perteneciente a un servidor DHCP. En este caso los containers son: <subnet>, <range> y <dhcp-options>. Sin embargo, los leafs son los elementos XML restantes, como por ejemplo <low> y <max-lease-time>, cuyos valores son 10.254.239.10 y 1200 respectivamente.

Para operar sobre las configuraciones primero se debe seleccionar un datastore. Se recomienda editar la configuración de sólo un datastore a la vez, dejando los restantes para consulta debido a que se genera una única operación OP_EDIT_CONFIG por cada uno de éstos.

La estructura de datos XML de la configuración que se modifica en la interfaz gráfica de usuario se almacena en el cache del servidor para poder procesar las solicitudes del usuario entre varios HTTP requests, sin generarla de nuevo en cada arribo. Cada vez que se carga el árbol de contenedores y la grilla de elementos nuevos, utilizando el botón *Load Device Informacion*, la configuración de operaciones anteriores en cache del servidor se eliminan y se reinicia con la configuración almacenada la base de datos. En memoria cache (RAM) se almacena solamente una referencia al objeto que contiene la estructura de toda la configuración para que el garbage collector de Python no lo elimine. Dicha referencia se guardas con el username_device-id_datastore (referirse a la función *device_cfg* en el controlador default.py para más detalles).

Esta estructura de datos XML de configuración del dispositivo está descripta por una estructura auto-referenciada DOM (Document Object Model), y se genera siempre desde la configuración del datastore guardada en la base de datos.

A ésta se le agrega atributos adicionales a cada elemento XML para permitir un mejor procesamiento de las operaciones del usuario sobre la configuración. Se incorpora el atributo *nems_id* con el fin de disponer un identificador único para cada elemento (containers y leafs) de la configuración y almacenar referencias a cada uno de éstos en la estructura DOM (una tabla hash con nems_id como llaves y las referencias a los elementos como valores). Dicha tabla permite mejorar el tiempo de búsqueda de los elementos en la estructura DOM y así mejorar el tiempo de ejecución de las solicitudes (para no parsear la configuración completa cada vez que se realiza una modificación). El atributo *nems_type* se incorpora para identificar aquellos elementos que son leaf y contenedores, y el atributo *operation* para especificar la operación solicitada sobre la configuración del dispositivo (esta última corresponde a los parámetros "replace", "delete" y "create" del protocolo Netconf dentro de la operación <edit-config>).

Para agregar nuevos containers o leafs es necesario incluir la información necesaria para generar su estructura en la estructura DOM. Estos son el tag, el prefijo del namespace, el URL del namespace asociado y el valor (este último solamente para los leafs). El usuario puede generar nuevos namespaces en el caso de que éstos no se encuentren listados en la grilla de mapeo de prefijos a URL de los namespaces. Se sugiere incluir la definición de todos los namespaces que se utilizarán en los archivos de configuración con anterioridad. La Figura 4.9 ilustra un ejemplo para agregar un nuevo contenedor, teniendo como padre al contenedor ns1-shared-network, y un nuevo leaf dentro del contenedor ns1:dhcp-options.

Las operaciones soportadas para modificar la configuración de los datastores de los dispositivos son:

- En contenedores:
  - Agregar uno nuevo
  - Borrar uno existente
  - Editar el tag de uno existente
- En leafs:
  - Agregar uno nuevo
  - Borrar uno existente
  - Editar el valor de uno existente

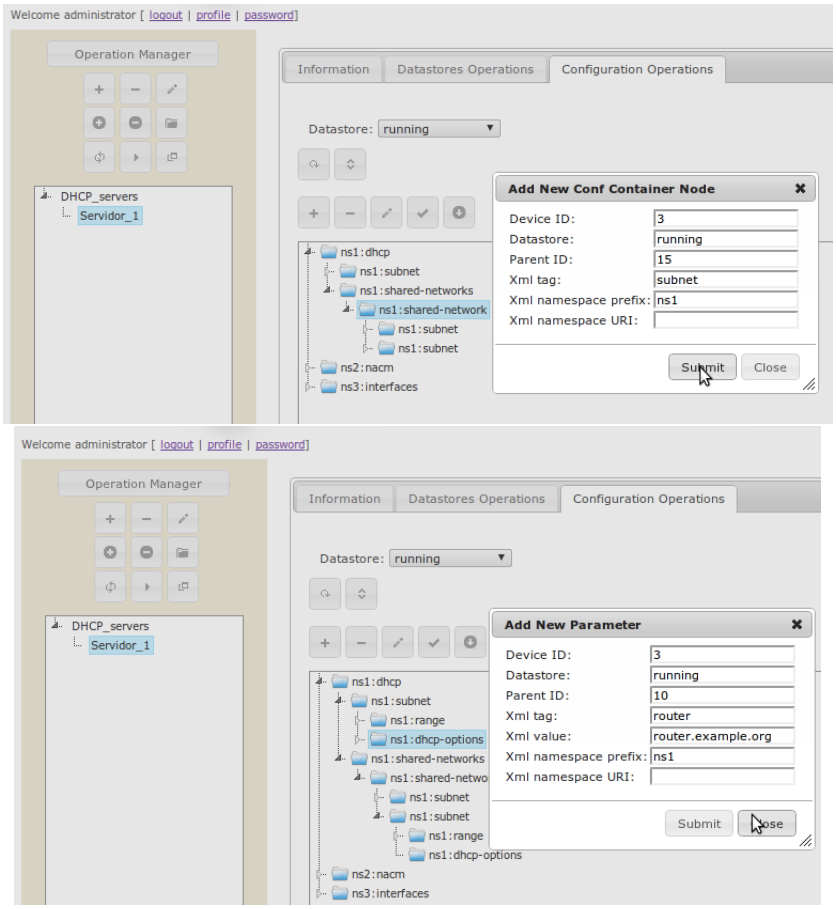

Figura 4.9: Información para agregar nuevo container y leaf.

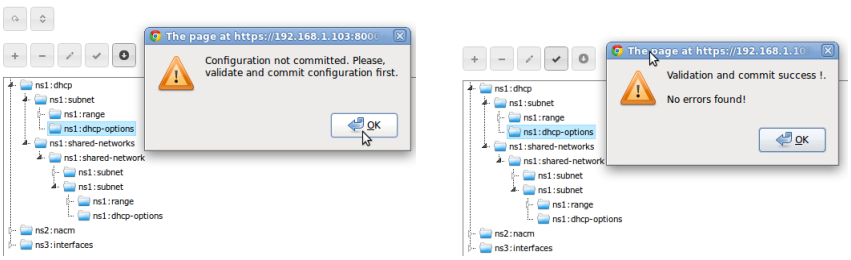

Figura 4.10: Validación de configuraciones.

4.5. Validación de datastores

Como se mencionó en capítulos anteriores, todas las configuraciones en datastores son validadas previamente antes de enviarse a los dispositivos de red. Para realizar este proceso, se remueven todos los atributos extras incorporados por el servidor (nems_id, nems_type y operation) y se procesa toda la estructura DOM para que sólo represente la configuración del datastore luego una exitosa

ejecución de la operación. La misma se valida contra los esquemas de configuración DSDL correspondiente al tipo de dispositivo asociado al elemento de red, utilizando el script yang2dsdl. Este último se ejecuta como un proceso nuevo de sistema operativo, se evalúan los valores de retorno y se parsean los mensajes de información. En caso que la validación no sea exitosa, se informa al usuario los elementos donde se encontraron errores.

Si el usuario intenta realizar una operación OP_EDIT_CONFIG sin haber validado la configuración previamente, la interfaz gráfica le informa sobre esto y no permite que la configuración sea enviada al dispositivo.

Capítulo 5

Conclusiones

5.1. Conclusiones

La implementación de soluciones para la gestión de red es esencial para su operación, mantenimiento y administración. En consecuencia se han desarrollado diferentes técnicas, mecanismos y protocolos para afrontar dichas tareas. Los protocolos de management desempeñan un rol fundamental en éste ámbito, ya que proveen un lenguaje en común para que los gestores y los agentes o dispositivos puedan comunicarse.

Netconf (RFC 4147) surge de la necesidad de disponer de un protocolo simple y estándar para la gestión de configuraciones, para reemplazar gradualmente las soluciones basadas en CLIs (Command Line Interfaces) propietarios y el SNMP (Simple Network Management Protocol). El protocolo Netconf proporciona a los administradores de red un entorno y un conjunto de métodos RPC (Remote Procedures Calls) basados en codificación XML (Extensible Markup Language) para gestionar (instalar, modificar y borrar) datos de configuración de los equipos. La creación de este protocolo, ligado a nuevos métodos, lenguajes de modelado de datos, y debido a la tendencia de las empresas operadoras a utilizar este tipo de tecnologías interoperables, proveen ventajas claras para la implementación de nuevas soluciones en este ámbito y fueron motivación para la elaboración de este trabajo de postgrado.

El desarrollo del mismo estuvo basado en el diseño e implementación del servidor de gestión de elementos de red NEMS (Netconf Element Management System), con la finalidad de administrar la configuración de aquellos equipos de red con soporte del protocolo Netconf. Dicha solución fue programada totalmente

en lenguaje Python y proporciona un entorno tanto flexible como amigable para la realización de cambios de configuración, operaciones en datastores y multi-operador.

Las características principales del servidor NEMS son: interfaz web mediante conexión HTTPs, autenticación y autorización de usuarios, registración de nuevos operadores/administradores, acceso simultáneo, almacenamiento persistente de configuraciones de dispositivos en base de datos Postgresql, soporte de protocolo base Netconf 1.0 (RFC 4147) y fácil extensión para la implementación de nuevas operaciones. El código queda disponible en Google Code bajo la licencia GNU GPL v3:
http://code.google.com/p/netconf-element-management-system/.

La propuesta inicial también incluyó como objetivo secundario la extensión de la librería Ncclient[11] para soportar el servicio de notificaciones de eventos en forma asincrónica[17]. Luego de estudio del alcance de las características del servidor se se priorizó la implementación de nuevas funcionalidades que no se tuvieron en cuenta al principio del proyecto, pero que se consideraron como fundamentales para la adecuada operación del servidor de gestión. Ejemplos de éstas son: la autenticación y autorización de usuarios para accesos diferenciados y simultáneos a la interfaz web, diseño de base de datos con triggers y notificaciones, manejo dinámico de configuraciones en forma de árbol e implementación de mecanismos de bloqueo para la edición de parámetros de configuración de diferentes bases de datos (datastores) de dispositivos.

Todas las funcionalidades del servidor NEMS se validaron exitosamente utilizando Netconfd, un servidor Netconf conforme al estándar, distribuido dentro del paquete de herramientas YUMA (YANG-Based Unified Modular Automation).

Personalmente, este proyecto fue todo un desafío ya que fue mi primer desarrollo de software. Me queda la experiencia y los conocimientos de haber trabajado con base de datos, Python, Javascript, CSS, XML y HTML, y de haber aprendido muchos conceptos nuevos, como diseño y funcionamiento de base de datos, diseño e implementación de aplicaciones web, manejo de procesos e hilos y aquellos relacionados a la programación orientada a objetos.

5.2. Trabajo a futuro

Está planeado como trabajo a futuro implementar nuevas funcionalidades al servidor NEMS para mejorar sus prestaciones, las cuales fueron surgiendo durante el desarrollo de este trabajo:

- Verificación periódica de la conexión a la base de datos
- Mejorar los handlers de excepciones cuando se producen errores en las operaciones Netconf para proporcionar mayor información a los usuarios sobre lo ocurrido.
- Implementar auto-refresh periódico para actualizar el estado de las grillas (operation manager y la de mapeo de prefijos a URI de los namespaces).
- Mejorar el proceso de alta de nuevos usuarios. Actualmente se realiza a través de appadmin.
- Agregar diálogos de confirmación en la interfaz web antes de submitir una operación a la base de datos.
- Utilización de tipo de datos XML en base de datos para manejo de configuraciones.

Apéndice A

Generación de schemas para validación de datastores

A.1. Módulo YANG y datastore de configuración

El módulo YANG utilizado para realizar las validaciones de los datastores de configuración corresponde al de un servidor DHCP. En este esquema se encuentran los parámetros necesarios, con sus posibles valores y restricciones, y fue extraído del tutorial oficial de la página Pyang (dhcp-data.yang)[7]. A su vez, se puede observar la configuración XML del datastore startup en la validación de las operaciones del servidor NEMS, y también definido como datastore startup de Netconfd (dhcp-startup-config.xml).

```
1  module dhcp {
2    namespace "http://example.com/ns/dhcp";
3    prefix dhcp;
4
5    import ietf-yang-types { prefix yang; }
6    import ietf-inet-types { prefix inet; }
7    import ietf-ieee-types { prefix ieee; }
8
9    organization
10     "yang-central.org";
11   description
12     "Partial data model for DHCP, based on the config of
13      the ISC DHCP reference implementation.";
14
15   container dhcp {
16     description
17       "configuration and operational parameters for a DHCP server.";
```

```
18      leaf max-lease-time {
19        type uint32;
20        units seconds;
21        default 7200;
22      }
23
24      leaf default-lease-time {
25        type uint32;
26        units seconds;
27        must 'current() <= ../max-lease-time' {
28          error-message
29            "The default-lease-time must be less than max-lease-time";
30        }
31        default 600;
32      }
33
34      uses subnet-list;
35      container shared-networks {
36        list shared-network {
37          key name;
38
39          leaf name {
40            type string;
41          }
42          uses subnet-list;
43        }
44      }
45
46      container status {
47        config false;
48        list leases {
49          key address;
50
51          leaf address {
52            type inet:ip-address;
53          }
54          leaf starts {
55            type yang:date-and-time;
56          }
57          leaf ends {
58            type yang:date-and-time;
59          }
60          container hardware {
61            leaf type {
62              type enumeration {
63                enum "ethernet";
64                enum "token-ring";
65                enum "fddi";
66              }
67            }
68            leaf address {
69              type yang:phys-address;
70            }
71          }
72        }
73      }
74    }
75
76    grouping subnet-list {
77      description "A reusable list of subnets";
78      list subnet {
79        key net;
80        leaf net {
81          type inet:ip-prefix;
82        }
83        container range {
84          presence "enables dynamic address assignment";
85          leaf dynamic-bootp {
86            type empty;
87            description
88              "Allows BOOTP clients to get addresses in this range";
89          }
90          leaf low {
91            type inet:ip-address;
92            mandatory true;
93          }
94          leaf high {
95            type inet:ip-address;
96            mandatory true;
97          }
98        }
```

```
99        container dhcp-options {
100         description "Options in the DHCP protocol";
101         leaf-list router {
102           type inet:host;
103           ordered-by user;
104           reference "RFC 2132, sec. 3.8";
105         }
106         leaf domain-name {
107           type inet:domain-name;
108           reference "RFC 2132, sec. 3.17";
109         }
110       }
111
112       leaf max-lease-time {
113         type uint32;
114         units seconds;
115         default 7200;
116       }
117     }
118   }
119 }
```

Código 6: Esquema YANG para servidor DHCP (dhcp-data.yang).

Una de las funciones implementadas en pyang es el mapeo de módulos de datos YANG a esquemas DSDL (RELAX NG, Schematron y DSRL). Los esquemas generados se utilizan para realizar la validación de los datastores Netconf. Esta transformación se encuentra definida en el RFC 6110 [28]. Los comandos que se muestran en el Código 8 corresponden, primero, a transformación de dhcp-data.yang a esquemas DSDL con el script yang2dsdl (que es parte de la distribución pyang), y luego, a los resultados de la validación de los contenidos de configuración de un datastore. Los archivos generados son: dhcp-gdefs.rng, dhcp-data.dsrl, dhcp-data.rng y dhcp-data.sch. El Código 9 demuestra la validación y el mensaje de error luego de cambiar el valor de *default-lease-time* a 7201.

```
<?xml version="1.0" encoding="utf-8"?>
<nc:data xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <dhcp xmlns="http://example.com/ns/dhcp">
    <max-lease-time>7200</max-lease-time>
    <default-lease-time>600</default-lease-time>

    <subnet>
      <net>10.254.239.0/27</net>
      <range>
        <dynamic-bootp/>
        <low>10.254.239.10</low>
        <high>10.254.239.20</high>
      </range>
      <dhcp-options>
        <router>rtr-239-0-1.example.org</router>
        <router>rtr-239-0-2.example.org</router>
      </dhcp-options>
      <max-lease-time>1200</max-lease-time>
    </subnet>

    <shared-networks>
      <shared-network>
        <name>224-29</name>
        <subnet>
          <net>10.17.224.0/24</net>
          <range>
            <low>10.17.224.10</low>
            <high>10.17.224.250</high>
          </range>
          <dhcp-options>
            <router>rtr-224.example.org</router>
          </dhcp-options>
        </subnet>
        <subnet>
          <net>10.0.29.0/24</net>
          <range>
            <low>10.0.29.10</low>
            <high>10.0.29.230</high>
          </range>
          <dhcp-options>
            <router>rtr-29.example.org</router>
          </dhcp-options>
        </subnet>
      </shared-network>
    </shared-networks>
  </dhcp>
</nc:data>
```

Código 7: Instancia de startup datastore de configuración de servidor DHCP (dhcp-startup-config.xml).

```
root@martin-laptop:/tmp# yang2dsdl  -b dhcp -t data dhcp-data.yang yuma-nacm.yang yuma-interfaces.yang
dhcp-data.yang:3: warning: unexpected modulename "dhcp" in dhcp-data.yang, should be dhcp-data
dhcp-data.yang:9: warning: imported module ietf-ieee-types not used
== Generating RELAX NG schema './dhcp-data.rng'
Done.

== Generating Schematron schema './dhcp-data.sch'
Done.

== Generating DSRL schema './dhcp-data.dsrl'
Done.
root@martin-laptop:/tmp# ls

dhcp-data.dsrl
dhcp-data.rng
dhcp-data.sch
dhcp-data.xml
dhcp-data.yang
dhcp-gdefs.rng
ietf-ieee-types.yang
ietf-inet-types.yang
ietf-yang-types.yang
yuma-interfaces.yang

root@martin-laptop:/tmp# yang2dsdl -s -b dhcp -t data -v dhcp-startup-config.xml
== Using pre-generated schemas

== Validating grammar and datatypes ...
dhcp-startup-config.xml validates

== Adding default values... done.

== Validating semantic constraints ...
No errors found.
```

Código 8: Generación de esquemas DSDL y validación de datastore.

```
root@martin-laptop:/tmp# yang2dsdl -s -b dhcp -t data -v dhcp-startup-config.xml
== Using pre-generated schemas

== Validating grammar and datatypes ...
dhcp-startup-config.xml validates

== Adding default values... done.

== Validating semantic constraints ...
--- Failed assert at "/nc:data/dhcp:dhcp/dhcp:default-lease-time":
    The default-lease-time must be less than max-lease-time
```

Código 9: Validación de datastore de configuración incorrecto con default-lease-time = 7201.

Apéndice B

Netconfd

B.1. Configuración de Netconfd

```
# netconfd configuration file
# version 1.12-2; 2010-06-01
#

netconfd {

## leaf access-control
#  Controls how access control is enforced by the server.
#    enforcing
#    permissive
#    disabled
#    off
#
# access-control enforcing
access-control off
#
#
## leaf log-level
#  Sets the debug logging level for the program.
#    off
#    error
#    warn
#    info
#    debug
#    debug2
#       log-level debug2
        log-level    debug3
#
## leaf-list port
#  Specify the TCP ports that the server will accept
#  connections from.   Up to 4 port numbers can be configured.
#  If any ports are configured, then only those values
#  will be accepted by the server.
#
port 830
#
```

```
#
## choice startup or no-startup
#  Selects the startup configuration file to use
#  - To skip loading any startup file, and save to the default
#    if needed, use 'no-startup'
#  - To use a specific startup file, use an absolute filespec
#  - To use a startup file in the YUMA_DATAPATH, use a relative
#    filespec.
#
#startup startup-cfg.xml
#startup dhcp-config.xml
startup dhcp-startup-config.xml
#
## leaf target
#  The database to use as the target of edit-config operations.
#     running
#     candidate
#
target candidate
target running
#
## leaf with-startup
#  If 'true', then the :startup capability will be enabled.
#  If 'false', then no distinct startup
#
with-startup true
}
```

Código 10: Archivo de configuración /etc/yuma/netconfd.conf

B.2. Ejemplo de operaciones OP_DELETE y OP_EDIT_CONFIG desde servidor NEMS.

A continuación se muestra la mensajería de debug en consola del servidor Netconfd interactuando con el servidor NEMS para dos operaciones, OP_DELETE y OP_EDIT_CONFIG. La primera operación se ejecuta para eliminar toda configuración que contiene el datastore del dispositivo. OP_EDIT_CONFIG está asociada a la carga completa de una configuración nueva desde un archivo de configuración xml proporcionado por el usuario. (Nota: solamente se observa los requests que arriban a Netconfd).

```
root@martin-laptop:/etc/yuma# netconfd --superuser=martin
--module=/home/martin/Desktop/NEMS_Eclipse/nems_eclipse/Docs/Yang_Schema_Generation/dhcp-data.yang
Starting netconfd...
Copyright (c) 2008-2011, Andy Bierman, All Rights Reserved.
.
.
.

New session 2 created OK
.
.
.
```

```
.
agt_ses msg ready for session 2
Incoming msg for session 2
<?xml version='1.0' encoding='UTF-8'?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:confirmed-commit:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:url:1.0?scheme=http,ftp,file,https,sftp</capability>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:startup:1.0</capability>
    <capability>urn:liberouter:params:netconf:capability:power-
control:1.0urn:ietf:params:netconf:capability:interleave:1.0</capability>
  </capabilities>
</hello>

xml_consume_node: skip unused node (XML_PI_NODE)
agt_top: got node
agt_hello got node
XML node (2:0): START hello
   attr: ns:5 name:xmlns (urn:ietf:params:xml:ns:netconf:base:1.0)

agt_val_parse: nc:client-hello btyp:container
Session 2 for martin@192.168.1.103 now active
.
.
.
agt_ses msg ready for session 2
Incoming msg for session 2
<?xml version='1.0' encoding='UTF-8'?>
<rpc message-id="urn:uuid:6ede714c-e96f-11e0-867e-001f3c3edee7"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <lock>
    <target><startup /></target>
  </lock>
</rpc>
.
.
.
agt_ses msg ready for session 2
Incoming msg for session 2
<?xml version='1.0' encoding='UTF-8'?>
<rpc message-id="urn:uuid:6ef06a82-e96f-11e0-867e-001f3c3edee7"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <delete-config>
    <target><startup /></target>
  </delete-config>
</rpc>
.
.
.
agt_ses msg ready for session 2
Incoming msg for session 2
<?xml version='1.0' encoding='UTF-8'?>
<rpc message-id="urn:uuid:6f01e924-e96f-11e0-867e-001f3c3edee7"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <unlock>
    <target><startup /></target>
  </unlock>
</rpc>
.
.
.
agt_ses msg ready for session 2
Incoming msg for session 2
<?xml version='1.0' encoding='UTF-8'?>
<rpc message-id="urn:uuid:6f0db696-e96f-11e0-867e-001f3c3edee7"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <close-session />
</rpc>
.
.
.
Session 2 closed
.
.
.
```

```
.
New session 3 created OK
.
agt_ses msg ready for session 3
Incoming msg for session 3
<?xml version='1.0' encoding='UTF-8'?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:confirmed-commit:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:url:1.0?scheme=http,ftp,file,https,sftp</capability>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:startup:1.0</capability>
    <capability>urn:liberouter:params:netconf:capability:power-control:1.0urn:ietf:params:netconf:
    capability:interleave:1.0</capability>
  </capabilities>
</hello>
.
agt_ses msg ready for session 3
Incoming msg for session 3
<?xml version='1.0' encoding='UTF-8'?>
<rpc message-id="urn:uuid:0ac90964-e970-11e0-867e-001f3c3edee7"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <lock>
    <target><startup /></target>
  </lock>
</rpc>
.
agt_ses msg ready for session 3
Incoming msg for session 3
<?xml version='1.0' encoding='UTF-8'?>
<rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:ns1="http://example.com/ns/dhcp"
message-id="urn:uuid:0adb2888-e970-11e0-867e-001f3c3edee7" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><startup /></target>
    <default-operation>replace</default-operation>
    <nc:config>
      <ns1:dhcp>
        <ns1:max-lease-time>7200</ns1:max-lease-time>
        <ns1:default-lease-time>600</ns1:default-lease-time>
        <ns1:subnet>
          <ns1:net>10.254.239.0/27</ns1:net>
          <ns1:range>
            <ns1:dynamic-bootp />
            <ns1:low>10.254.239.10</ns1:low>
            <ns1:high>10.254.239.20</ns1:high>
          </ns1:range>
          <ns1:dhcp-options>
            <ns1:router>rtr-239-0-1.example.org</ns1:router>
            <ns1:router>rtr-239-0-2.example.org</ns1:router>
          </ns1:dhcp-options>
          <ns1:max-lease-time>1200</ns1:max-lease-time>
        </ns1:subnet>
        <ns1:shared-networks>
          <ns1:shared-network>
            <ns1:name>224-29</ns1:name>
            <ns1:subnet>
              <ns1:net>10.17.224.0/24</ns1:net>
              <ns1:range>
                <ns1:low>10.17.224.10</ns1:low>
                <ns1:high>10.17.224.250</ns1:high>
              </ns1:range>
              <ns1:dhcp-options>
                <ns1:router>rtr-224.example.org</ns1:router>
              </ns1:dhcp-options>
            </ns1:subnet>
            <ns1:subnet>
              <ns1:net>10.0.29.0/24</ns1:net>
              <ns1:range>
                <ns1:low>10.0.29.10</ns1:low>
                <ns1:high>10.0.29.230</ns1:high>
              </ns1:range>
              <ns1:dhcp-options>
                <ns1:router>rtr-29.example.org</ns1:router>
              </ns1:dhcp-options>
            </ns1:subnet>
          </ns1:shared-network>
        </ns1:shared-networks>
      </ns1:dhcp>
    </nc:config>
  </edit-config>
</rpc>
.
.
```

```
183 .
184 .
185 .
186 agt_ses msg ready for session 3
187 Incoming msg for session 3
188 <?xml version='1.0' encoding='UTF-8'?>
189 <rpc message-id="urn:uuid:0aece730-e970-11e0-867e-001f3c3edee7"
190 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
191   <unlock>
192     <target><startup /></target>
193   </unlock>
194 </rpc>
195 .
196 .
197 .
198 agt_ses msg ready for session 3
199 Incoming msg for session 3
200 <?xml version='1.0' encoding='UTF-8'?>
201 <rpc message-id="urn:uuid:6f0db696-e96f-11e0-867e-001f3c3edee7"
202 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
203   <close-session />
204 </rpc>
205
206 xml_consume_node: skip unused node (XML_PI_NODE)
207 agt_top: got node
208 agt_rpc: <close-session> for 2=martin@192.168.1.103
209 (m:urn:uuid:6f0db696-e96f-11e0-867e-001f3c3edee7) [2011-09-28T01:16:03Z]
210 XML node (2:1): EMPTY close-session
211
212 agt_rpc: expecting rpc end node
213 XML node (2:0): END rpc
214
215 agt_sys: generating <sysSessionEnd> notification
216 Queing <sysSessionEnd> notification to send (id: 5)
217 Session 3 closed
```

Apéndice C

Certificados Digitales

C.1. Generación de certificados

Los siguientes pasos fueron extraídos del tutorial oficial de Ubuntu https://help.ubuntu.com/8.04/serverguide/C/certificates-and-security.html

C.1.1. Generación del archivo CSR (Certificate Signing request)

Si se requiere un certificado desde un CA (Certificate Authority) o uno generado con firma propia (self-signed), el primer paso es generar la clave. Las claves para la solicitud CSR (Certificate Signing request) se crean ejecutando el siguiente comando:

```
openssl genrsa -des3 -out server.key 1024

Generating RSA private key, 1024 bit long modulus
.....................++++++
.................++++++
unable to write 'random state'
e is 65537 (0x10001)
Enter pass phrase for server.key:
```

Por cuestiones de seguridad se recomienta que el passphrase contenga por lo menos ocho caracteres. Debe incluir números y/o caracteres de puntuación. La clave generada para el servidor es almacenada en el archivo server.key.

En el caso del servidor NEMS, se configura Web2py para correr sin solicitud de passphrase, para ello se corre el siguiente comando para obtener la clave en forma insegura:

```
openssl rsa -in server.key -out server.key.insecure
```

La clave insegura se almacena en el archivo server.key.insecure. Es una opción también la generación de una clave con extensión .pem, pero es más complicado el procedimiento de configuración.

Para crear la solicitud CSR, se tiene que ejecutar:

```
openssl req -new -key server.key -out server.csr
```

Luego de entrar correctamente el passphrase, el comando pide por los datos de la empresa u organización (Site Name, Email Id, etc.), y crea el archivo server.csr como solicitud de certificado.

Ahora se puede enviar el archivo CSR para que sea procesado por el CA, que lo utilizará generar un certificado firmado por él. Por otro lado, se puede crear un certificado self-signed desde el mismo CSR.

C.1.2. Generar el certificado self-signed

El siguiente comando se puede ejecutar para generar el certificado self-signed:

```
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

Una vez que se inserta el passphrase correcto, el certificado es generado y será almacenado en el archivo server.crt.

Ahora solamente queda especificar los path asociados al certificado y la clave en el archivo de configuración Cfg.xml del servidor NEMS.

Apéndice D

Instalación de NEMS

D.1. Instalación de NEMS

Se deben seguir los siguientes pasos para instalar el servidor NEMS, los mismos no incluyen la instalación y puesta a punto de la base de datos ni instalación de paquetes requeridos para su funcionamiento.

1. Instalar Python 2.6.
2. Instalar servidor de base de datos Postgresql 9.x con soporte de plpythonu.
3. Instalar las librerías requeridas:
   - yang2dsld [7].
   - ncclient [11].
   - psutil [5].
   - psycopg2 [6].
   - lxml [1].
   - netifaces [3].
4. Instalar Web2py de su página oficial [32].
5. Descargar el archivo .tgz con NEMS de Google Code `http://code.google.com/p/netconf-element-management-system/`.
6. Descomprimir el archivo .tgz.

7. Copiar la carpeta NEMS_Web que se encuentra dentro de web2py_files a la carpeta applications dentro de web2py local. NEMS_Web se ejecutará como una aplicación dentro del ambiente Web2py.

8. Copiar el archivo routes.py dentro de la carpeta web2py local.

9. Completar los campos necesarios del archivo de configuración Cfg.xml, ubicado dentro de la carpeta cfg. Referirse a la tabla 3.1 para más información de cada parámetro.

10. Dar permiso de ejecución al archivo nems_main.py (chmod 755 nems_main.py).

11. Ejecutar el servidor corriendo el comando: python nems_main.py

Bibliografía

[1] lxml - XML and HTML with Python. URL http://lxml.de/.

[2] Netconf central. . URL http://www.netconfcentral.org/yang_docs.

[3] Netifaces. getting network addresses from python. . URL http://alastairs-place.net/netifaces/.

[4] Network configuration (netconf) group (ietf). . URL http://datatracker.ietf.org/wg/netconf/charter/.

[5] Psutil. URL http://code.google.com/p/psutil/.

[6] Psycopg. URL http://initd.org/psycopg/.

[7] Pyang. URL http://code.google.com/p/pyang/.

[8] Python programming language. URL http://www.python.org/.

[9] Yuma tools. URL http://www.netconfcentral.org/yuma.

[10] Martin Bertolina. Issue: xml in response not well formed. 2011. URL http://groups.google.com/group/web2py/browse_thread/thread/ba90ec88ff594b02#.

[11] Shikhar Bhushan. Ncclient. python library for netconf clients. 2009. URL http://oss.schmizz.net/ncclient/.

[12] A. Bierman y B. Lengyel. With-defaults capability for netconf. RFC 6243 (Proposed Standard), 2011. URL http://www.ietf.org/rfc/rfc6243.txt.

[13] Andy Bierman. Yuma netconfd manual. version 2.0. 2011. URL http://www.netconfcentral.org/static/manuals/v2/yuma-netconfd-manual.pdf.

[14] M. Bjorklund. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020 (Proposed Standard), 2010. URL http://www.ietf.org/rfc/rfc6020.txt.

[15] Ethan Cerami. *Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI WSDL*. Pearson Education, 2002.

[16] Limiao Chen Chang, Yanan y Wu Baozhen. A netconf-based distributed network management design using web services and p2p. En *ChinaGrid Conference*. 2010.

[17] S. Chisholm y H. Trevino. Netconf event notifications. RFC 5277 (Proposed Standard), 2008. URL http://www.ietf.org/rfc/rfc5277.txt.

[18] Jr. Christopher A. Jones, Fred L. Drake. *Python XML*. O'Reilly, first editon ed$^{\underline{\text{ón}}}$., 2002.

[19] Wesley J. Chun. *Core Python Programming 2nd Edition*. Prentice Hall, 2006.

[20] Alexander Clemm. *Network Management Fundamentals*. Fundamentals. Cisco Press, 2006.

[21] Paul Deitel y Harvey Deitel. *AJAX, Rich Internet Applications, and Web Development For Programmers*. Pearson Education, 2008.

[22] R. Enns. Netconf configuration protocol. RFC 4741 (Proposed Standard), 2006. URL http://www.ietf.org/rfc/rfc4741.txt. Obsoleted by RFC 6241.

[23] R. Enns, M. Bjorklund, J. Schoenwaelder, y A. Bierman. Network configuration protocol (netconf). RFC 6241 (Proposed Standard), 2011. URL http://www.ietf.org/rfc/rfc6241.txt.

[24] T. Goddard. Using netconf over the simple object access protocol (soap). RFC 4743 (Proposed Standard), 2006. URL http://www.ietf.org/rfc/rfc4743.txt.

[25] Yoo Sun-Mi Hong, James y Hong Taek Ju. Performance improvement methods for netconf-based configuration management. management of convergence networks and services. En *Lecture Notes in Computer Science*. 2006. Vol: 4238/2006.

[26] Phil Shafer Jürgen Schönwälder, Martin Björklund. Network Configuration Management Using NETCONF and YANG. 2010.

[27] E. Lear y K. Crozier. Using the netconf protocol over the blocks extensible exchange protocol (beep). RFC 4744 (Proposed Standard), 2006. URL http://www.ietf.org/rfc/rfc4744.txt.

[28] L. Lhotka. Mapping yang to document schema definition languages and validating netconf content. RFC 6110 (Proposed Standard), 2011. URL http://www.ietf.org/rfc/rfc6110.txt.

[29] Mark Lutz. *Learning Python 4th Edition*. O'Reilly, 2009.

[30] Juniper Networks. JUNOS OS: NETCONF XML Management Protocol Guide. 2011. URL http://www.juniperpodcast.com/techpubs/en_US/junos11.2/information-products/topic-collections/netconf-guide/netconf-guide.pdf.

[31] E. O'Tuathail y M. Rose. Using the Simple Object Access Protocol (SOAP) in Blocks Extensible Exchange Protocol (BEEP). RFC 4227 (Proposed Standard), 2006. URL http://www.ietf.org/rfc/rfc4227.txt.

[32] Massimo Di Pierro. Web2py. 2007. URL www.web2py.com.

[33] Massimo Di Pierro. Web2py book (english). 2011. URL http://web2py.com/book.

[34] J. Schoenwaelder. Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 4: Management framework. ISO/IEC 7498-4, 1989. URL http://standards.iso.org/ittf/PubliclyAvailableStandards/s014258_ISO_IEC_7498-4_1989(E).zip.

[35] J. Schoenwaelder. Overview of the 2002 IAB Network Management Workshop. RFC 3535 (Informational), 2003. URL http://www.ietf.org/rfc/rfc3535.txt.

[36] J. Schoenwaelder. Common YANG Data Types. RFC 6021 (Proposed Standard), 2010. URL http://www.ietf.org/rfc/rfc6021.txt.

[37] M. Scott y M. Bjorklund. Yang module for netconf monitoring. RFC 6022 (Proposed Standard), 2010. URL http://www.ietf.org/rfc/rfc6022.txt.

[38] P. Shafer. An Architecture for Network Management Using NETCONF and YANG. RFC 6244 (Informational), 2011. URL http://www.ietf.org/rfc/rfc6244.txt.

[39] Aaron Skonnard y Martin Gudgin. *Essential XML Quick Reference*. Addison-Wesley, 2002.

[40] M. Wasserman. Using the netconf protocol over secure shell (ssh). RFC 6242 (Proposed Standard), 2011. URL http://www.ietf.org/rfc/rfc6242.txt.

[41] M. Wasserman y T. Goddard. Using the netconf configuration protocol over secure shell (ssh). RFC 4742 (Proposed Standard), 2006. URL http://www.ietf.org/rfc/rfc4742.txt. Obsoleted by RFC 6242.