Paulo César Costa Tavares

Editor de YANG

Paulo César Costa Tavares

Editor de YANG

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Professor Doutor José Luís Guimarães Oliveira, Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Professor Doutor Pedro Alexandre de Sousa Gonçalves, Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro.

o júri

presidente

Doutor Armando José Formoso Pinho
Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais

Doutor José Luís Guimarães Oliveira
Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Doutor Pedro Alexandre de Sousa Gonçalves
Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda

Doutor António Manuel de Jesus Pereira
Professor Coordenador do Departamento de Engenharia Informática do Instituto Politécnico de Leiria

palavras-chave Gestão de redes, NETCONF, YANG, IDE.

resumo O desenvolvimento de aplicações de gestão tipicamente requer a definição do modelo de dados, a criação das aplicações que respeitem esse modelo de dados e a implementação das interfaces de comunicação. Apesar de essas tarefas serem normalmente desenvolvidas por profissionais bem treinados, estes têm que as implementar usando diferentes aplicações, numa sequência coordenada que frequentemente são obrigados a repetir devido a um qualquer erro na definição inicial do modelo de dados. A adopção de tecnologias Web no desenvolvimento de aplicações de gestão NETCONF permite a automatização de vários procedimentos. Este documento apresenta uma plataforma de desenvolvimento integrado para soluções baseadas em NETCONF sob a forma de um *plug-in* para o IDE Eclipse. O trabalho inclui a criação de um *parser* para a linguagem YANG que foi integrado com o IDE que permite o desenvolvimento total de aplicações de gestão baseados neste protocolo.

keywords Network Management, NETCONF, YANG, IDE.

abstract The development of network and systems management software typically requires the data model definition, the construction of applications respecting that data model and also the implementation of the communication interfaces. Although such tasks are usually performed by welltrained professionals, they have to perform those tasks using different applications, in a coordinated sequence that they may unfortunately repeat due to errors in data model definition. The adoption of web technologies in NETCONF design allows the automation of several development tasks. Current work presents an integrated development platform for NETCONF based-solutions in the form of a *plug-in* for the Eclipse IDE. The work includes the creation of a YANG parser that was integrated within the IDE and that enables the complete creation of the management applications.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1.
INTRODUÇÃO

A utilização de sistemas informáticos têm-se vindo a massificar tendo-se tornado cada vez mais complexos com o desenvolvimento de novas tecnologias (software e hardware), maiores taxas de transmissão e armazenamento de dados. Este crescimento levou igualmente a que a administração desses mesmos sistemas tivesse de evoluir ao longo do tempo.

Na década de 1960, o predomínio dos sistemas computacionais na maioria das empresas eram grandes servidores e *mainframes*. Cada servidor suportava os seus vários utilizadores através dos vários terminais. Estes servidores eram administrados por um operador dedicado através de um terminal especial que normalmente se encontrava numa área reservada. Cada fabricante fornecia no seu terminal de gestão a sua própria versão de *software* para administração do sistema. Este software consistia tipicamente numa interface baseada em texto para a visualização do estado dos parâmetros do sistema e permitia a sua alteração nos ficheiros de configuração contidos na *mainframe*. Existia pouca interoperabilidade entre os vários sistemas de gestão, uma vez que cada empresa especificava o seu próprio sistema, dificultando assim a interligação de equipamentos de fabricantes distintos [1].

Nos anos 80 com o desenvolvimento na tecnologia de sistemas de rede e de normas abertas como o *Internet Protocol* permitiu que equipamentos de diferentes fabricantes pudessem comunicar mais facilmente entre si. Isto originou o desenvolvimento de redes computacionais cada vez maiores, que por sua vez gerou a necessidade de criar de novas normas para a gestão dessas mesmas redes. Simultaneamente o desenvolvimento de computadores pessoais e o aparecimento do modelo cliente-servidor implicou que se repensasse como deveria ser realizada a gestão de sistemas. A necessidade de *standards* conduziu à especificação das normas SNMP (*Simple Network Management Protocol*) [2] e MIB (*Management Information Base*) [3] pelo IETF, assim como o *Common Management Information Protocol* [4] pelo ITU. Subsequentemente o SNMP foi maioritariamente adoptado como o esquema de gestão para redes de computadores em detrimento do CMIP.

Ambos os esquemas, SNMP e CMIP, replicavam no paradigma do modelo cliente-servidor com o modelo agente-gestor. Neste modelo o gestor fornecia as diferentes funções de gestão aos diversos agentes. O agente, que se encontrava em cada equipamento administrado, facultava o acesso à informação requerida para a sua gestão, e o *software* de gestão centralizado apresentava essa informação ao administrador. A arquitectura agente-gestor é a mais predominante arquitectura de gestão implementada nos sistemas de hoje em dia.

Outro aspecto inovador do modelo de gestão SNMP foi a normalização da informação da gestão através de formatos standards (*Management Information Base*, MIBs), que facilitou significativamente a complexidade de administração de sistemas e das redes. Antes do seu

desenvolvimento, a informação de gestão era codificada em diferentes formatos que dificultava a implementação de sistemas que operassem com formatos distintos.

Nos inícios da década de 1990, o *Common Object Request Broker Arquitecture* (CORBA) [5] ganhou popularidade como o método para a implementação de sistemas distribuídos. Vários produtos para administração de sistemas foram desenvolvidos baseados no modelo agente-gestor onde o CORBA era usado como protocolo de comunicação ao invés do SNMP. Enquanto o SNMP era usado principalmente para dispositivos de rede e as especificações MIB dificultavam a representação mais complexas de estruturas de informação, os objectos CORBA permitiam a representação da informação de gestão em estruturas arbitrárias e complexas. A norma SNMP também tinha algumas debilidades relativamente aos mecanismos de segurança. A arquitectura *Telecommunications Information Networking Architecture* (TINA), que era dominante nas redes de telecomunicação nos anos 90, baseou as suas especificações de gestão no CORBA tendo sido predominante para a gestão de servidores e sistemas dentro das empresas.

A implementação do protocolo HTTP (*Hyper Text Transfer Protocol*) [6] permitiu a expansão da Internet na segunda metade da década de 90 e expôs algumas das deficiências da arquitectura de gestão CORBA. As empresas começaram a necessitar de implementar várias camadas de *firewalls* para atenuar as ameaças à segurança de dentro e de fora das suas redes, e os protocolos CORBA necessitavam de certas excepções nessas *firewalls* para que pudessem comunicar. Embora o CORBA fornecesse mecanismos para que o gestor e o agente comunicassem um com o outro, não dispunha de uma norma para a informação da gestão, isto é, o protocolo não especificava um modelo de dados, relegando aos fornecedores a criação da sua própria especificação para o mesmo.

O *Desktop Management Task Force* (DMTF) [7] foi formado em 1990 para desenvolver um modelo de informação comum para a maioria dos sistemas de gestão – o análogo das MIBs para a gestão de redes. Foi criado o *Common Information Model* (CIM) [8] que usava a abordagem orientada a objectos para representar a informação dos sistemas de gestão numa forma normalizada. Outra norma desenvolvida pelo DMTF foi o *Web-Based Enterprise Management* (WBEM) [9] que fornecia um protocolo de gestão baseado em tecnologias Web. A tecnologia WBEM utiliza o modelo de dados CIM para representação da informação, codifica a informação de gestão em CIM-XML [10] e efectua o transporte de informação em HTTP [11].

Para além do desenvolvimento de protocolos e representação da informação de gestão, a área de gestão de sistemas tem vindo rapidamente a adoptar tecnologias emergentes no âmbito computacional. Com a massificação da implementação de bases de dados relacionais levou que a informação de gestão pudesse ser implementada em base de dados e que a sua manipulação fosse realizada essencialmente através de operações à própria base de dados. Subsequentemente, a popularidade do *Internet browser* impeliu que muitas consolas de gestão fossem redefinidas para operarem usando como interface o *browser*, e que todas as operações de administração fossem realizadas através de *scripts* em execução num servidor Web. Com o conceito de usabilidade e factor humano a terem cada vez mais importância na

indústria, novas iniciativas como o *policy-based management* [12] e computação automática têm sido implementadas com o objectivo de melhorar a usabilidade dos sistemas de gestão.

No início do século 21, e ao contrário das expectativas iniciais, o SNMP continuava a não ser usado para a configuração de equipamentos de rede, mas primordialmente para a monitorização da rede. Em 2002, o *Internet Architecture Board* (IAB) juntamente com alguns membros principais do IETF e vários administradores de redes reuniram-se para debater esta questão [13]. Concluíram que a maioria dos administradores preferiam usar a interface de consola para a configuração dos equipamentos, por esta ser baseada em texto em oposição às *Basic Encoding Rules* (BER) [14] do SNMP. Outro factor importante à não utilização do SNMP foi o facto de muitos dos fabricantes não fornecerem as funções necessárias para configuração completa do equipamento via SNMP.

Por volta da mesma altura, a *Juniper Networks* [15], que vinha usando uma codificação baseada em XML para a administração de redes, partilhou esse conhecimento com a IEFT para servir de base à criação de um novo protocolo de gestão de redes – o NETCONF – criado com o intuito de melhor servir quer as necessidades dos fabricantes como as dos administradores de redes. O *Network Configuration Protocol* (NETCONF) [16] é um protocolo de gestão de elementos de rede que utiliza tecnologias Web, tais como SOAP e XML para o transporte de mensagens e para a codificação dos elementos dos dados de gestão. O protocolo define as mensagens que são trocadas entre o agente e gestor, bem como a sua temporização, mas não define o modelo de dados a ser utilizado pelos elementos de gestão.

A definição desse modelo de dados foi atribuída ao NETMOD, grupo de trabalho inserido no IETF, que tinha como objectivo a criação de uma linguagem de fácil leitura e que especificasse a semântica de operação e configuração dos dados, notificações e operações, a que chamou YANG.

A linguagem do modelo de dados YANG foi publicada na RFC 6020 [17] em Outubro de 2010 e tem como base a representação da estrutura de dados num formato de árvore XML. O YANG pode ser convertido para o formato XML equivalente, designado YIN (*YANG Independent Notation*), sendo esta tradução directa e totalmente reversível, o que confere a este modelo de dados uma maior interoperabilidade entre as aplicações de gestão já existentes.

1.1. MOTIVAÇÃO

A gestão de redes baseada no protocolo NETCONF tem sido alvo de muita atenção e pesquisa por diversos indivíduos e empresas da área de redes. O NETCONF ultrapassa muitas das limitações impostas pelo SNMP assim como de outros protocolos de gestão, providenciando uma melhor configuração de equipamentos de rede através o uso eficaz da tecnologia XML conjuntamente com outras tecnologias relacionadas. Estas características têm impelido o NETCONF a ser visto como um protocolo promissor, sendo cada vez mais adoptado em novas configurações para a gestão de equipamentos de rede [18]. Para tornar o

NETCONF interoperável e capaz de manipular os dados de configurações de forma normalizada foi criada a linguagem de modelação de dados YANG. Esta permite a modelação da semântica e organização dos dados de estado e configuração, operações e notificações a serem manipuladas pelo protocolo NETCONF. *Huiyang e al.* [19] revelam uma análise realizada as diferentes linguagens de modelação de dados do NETCONF (XML *Schema*, RELAX NG e YANG) onde concluíram que o YANG é, no geral, a melhor opção para a linguagem de modelação de dados para o protocolo. A sintaxe YANG é similar a linguagens de programação C ou C++ promovendo a sua legibilidade e desenvolvimento de módulos por parte dos administradores de redes.

Já existe um elevado número de dispositivos que suportam a sua configuração através do protocolo NETCONF. O protocolo também permite que seja implementado numa variedade de ambientes, onde se inclui a sua implementação sob a forma de um serviço Web. Um serviço Web é uma tecnologia baseada em XML que permite que um desenvolvimento independente da linguagem de programação ou da plataforma em que é implementado. Uma implementação NETCONF neste ambiente beneficia de inúmeras vantagens, desde a reutilização de normas existentes, facilidade no desenvolvimento da aplicação e integração com sistemas já implementados. A RFC 4743 [20] especifica a descrição de serviços NETCONF através de ficheiros WSDLs (*Web Services Description Language*) que, combinados com o XML *Schema* base do NETCONF (definido na RFC 4741) providenciam descrições suficientes para a geração automática de aplicações NETCONF sobre o protocolo de transporte SOAP. O SOAP é amplamente usado em serviços Web estando disponível um elevado número de ferramentas que possibilitam a geração automática através de informações especificadas em documentos WSDL, de aplicações para os clientes e os respectivos servidores.

Este trabalho visa implementar um editor para facilitar a criação de módulos de dados YANG tendo estes a possibilidade de serem derivados de módulos já existentes em formato SMI. O editor também deve permitir a conversão do modelo criado para um formato XML *Schema* para que possa ser integrado numa aplicação de gestão NETCONF gerada a partir da descrição do serviço contidas nos ficheiros WSDLs descritos na RFC 4743. Este trabalho irá descrever em detalhe a metodologia e tecnologias necessárias para implementação de uma aplicação NETCONF sobre SOAP, sendo facilmente replicável, mesmo por indivíduos sem grandes conhecimentos de desenvolvimento de serviços Web, possibilitando a realização de todo o ciclo completo de desenvolvimento de aplicações de gestão NETCONF de uma forma simples e rápida.

1.2. OBJECTIVOS

Os objectivos deste trabalho podem ser divididos em dois pontos principais.

O primeiro ponto prende-se com o desenvolvimento um editor de YANG, que disponibilize as facilidades de edição normalmente disponíveis nos IDEs (*Integrated*

Development Environment) mais utilizados. Para facilitar o processo de desenvolvimento do editor foi definido que este fosse criado sob a forma de um *plug-in* para Eclipse [21].

O editor deverá contemplar as funcionalidades básicas de edição do modelo de dados, sendo as de maior importância a análise sintáctica e correspondente detecção de erros, realce de sintaxe e apresentação esquemática do código (*outline*). O editor deverá ainda fornecer aos seus utilizadores a possibilidade de criação de um projecto ou um novo ficheiro YANG através de um assistente.

Para além das funções de edição o *plug-in* deverá englobar ferramentas que permitam a tradução do código YANG para os formatos YIN e XSD correspondestes bem como a importação de MIBs para o formato YANG.

O segundo ponto deste trabalho refere-se à criação de um método para a implementação de um sistema distribuído para administração de redes baseado em NETCONF que permita a integração de um modelo de dados YANG previamente definido no editor. Em mais pormenor, esse método deverá cingir a sua execução para operar no próprio Eclipse conjuntamente com o editor YANG desenvolvido. A aplicação distribuída deverá ser desenvolvida em linguagem java sendo gerada através do WSDL referente ao protocolo NETCONF. A aplicação deverá ainda permitir a edição do código gerado, continuando o desenvolvimento das aplicações de gestão geradas pelo editor, bem como o seu teste da aplicação de gestão e depuração de erros.

1.3. Estrutura e Organização do Documento

O presente e primeiro capítulo contextualiza o âmbito do projecto, introduzindo os conceitos, estabelecendo o enquadramento e definindo os objectos a que o trabalho se propõe.

O segundo capítulo apresenta uma visão geral das várias tecnologias de gestão existentes focando em particular o protocolo NETCONF e a sua correspondente linguagem de modelação de dados YANG.

O terceiro capítulo detalha todo o processo de desenvolvimento do *plug-in* Editor YANG, pormenorizando as diferentes tecnologias necessárias à sua implementação e o mecanismo para a criação da aplicação de gestão distribuída.

No quarto capítulo são apresentadas as funcionalidades e métodos desenvolvidos através da descrição de um exemplo prático passível de ser realizado na aplicação implementada.

No capítulo cinco é realizada uma análise a todo o sistema desenvolvido sendo indicadas as suas vantagens e desvantagens, considerações para um trabalho futuro e conclusões gerais do trabalho.

CAPÍTULO 2.
TECNOLOGIAS DE GESTÃO DE REDES

A gestão de redes refere-se às actividades, métodos, procedimentos e ferramentas que envolvem a operação, a administração, a manutenção e o aprovisionamento de sistemas de rede.

A operação consiste em manter a rede e os serviços que esta proporciona no estado mais correcto possível. Isto inclui a monitorização da rede para a descoberta de problemas o mais cedo possível, idealmente antes mesmo de os utilizadores se terem apercebido ou poderem ter sido afectados.

A administração refere-se ao acompanhamento dos recursos utilizados e como estes são distribuídos. Inclui também todos os procedimentos para manter a rede sob controle.

A manutenção está relacionada com a realização de reparações, substituições ou actualizações, de *software* ou *hardware* dos equipamentos pertencentes à rede. A manutenção envolve também medidas correctivas e preventivas de como melhorar o funcionamento da rede, como, por exemplo, o ajustamento dos parâmetros de configuração de um dispositivo.

O aprovisionamento tem como propósito a configuração dos recursos da rede para o suporte a um determinado serviço, como por exemplo o ajuste das configurações da rede para a activação do serviço de voz para um novo cliente.

Para a caracterização das funcionalidades de gestão e administração de uma rede a recomendação mais adoptada é o FCAPS. O FCAPS é um modelo funcional para a gestão de redes definido pelo ITU-T e ISO na especificação M.3400 [22]. Faz a partição da gestão de rede em cinco áreas funcionais: gestão de falhas dos dispositivos e serviços da rede, gestão da configuração dos dispositivos e dos serviços da rede, gestão da utilização e avaliação da rede, gestão do desempenho da rede e gestão da segurança.

- Gestão de Falhas: Para detectar, registar e notificar os utilizadores para, e o mais eficazmente possível, corrigir automaticamente problemas que surjam na rede para que esta mantenha o seu desempenho o mais eficiente possível.
- Gestão de Configuração: Para definir a informação da configuração do sistema para que o funcionamento das várias versões de *hardware* e os elementos de *software* possam ser controlados e geridos.
- Gestão de Contabilidade: Para medir a utilização e actividades individuais ou de um grupo na rede com o propósito de regulação e compilar o uso da mesma.
- Gestão do Desempenho: Para medir e disponibilizar vários aspectos de performance dos sistemas e monitorizar o desempenho e optimização dos mesmos. As variáveis de

desempenho da rede incluem o volume de transferências e utilização da linha como os tempos de resposta.

- Gestão da Segurança: Para controlar o acesso aos recursos da rede para que esta não possa ser sabotada e para que o acesso a informações sensíveis só possam ser realizado por aqueles que tenham autorização.

As normas mais predominantes e usadas hoje em dia usam como base o modelo agente-gestor. A entidade gerida pode ser um dispositivo ou uma aplicação a ser gerida através de um agente de gestão previamente instalado. Cada agente colecta os diferentes dados locais de gestão e transmiti-os ao gestor, que é uma instância de *software* a operar na entidade gestora, usando para isso um protocolo comum que ambos compreendem. O gestor recolhe os dados de todos os agentes e armazena-os num repositório para subsequentemente realizar, de acordo com a sua configuração, as diferentes funções de gestão e operações no sistema.

Com a expansão das redes, a evolução das arquitecturas de redes e o crescimento de requisitos para a gestão de redes, os protocolos de redes tiveram consequentemente de acompanhar essa evolução. Os protocolos de gestão de redes são usados para a especificação de como a informação de gestão é trocada entre os gestores e os seus agentes. Nas próximas secções são analisados os mais populares protocolos de gestão de redes, sendo focado em mais pormenor o protocolo de gestão NETCONF que está na base deste trabalho.

2.1. SUPORTE PROPRIETÁRIO DE GESTÃO

A interface de consola apesar de não ser um protocolo de gestão é o método mais adoptado pelos administradores de redes para a configuração dos dispositivos por si administrados. Os diversos fabricantes de equipamentos foram desenvolvendo interfaces proprietárias, que utilizando um conjunto de comandos por vezes específicos do modelo dos equipamentos, permitem a configuração dos mesmos.

A grande maioria dos componentes que podem ser encontrados num sistema de rede oferece a possibilidade de poderem ser geridos através de uma consola. Os comandos de gestão proporcionam a capacidade de modificação da configuração do dispositivo, assim como a possibilidade de visualização de contadores, e de estatísticas e o estado dos diferentes parâmetros que compõem a configuração do próprio componente. A consola pode ser acedida através de uma ligação directa ao próprio dispositivo ou através de uma ligação remota. Usualmente cada equipamento tem o seu próprio conjunto de comandos de configuração definidos pelo seu fabricante, o que diminui significativamente a interoperabilidade de dispositivos de diferentes empresas.

2.2. COMMON MANAGEMENT INFORMATION PROTOCOL

O *Common Management Information Protocol* (CMIP) [23] é um protocolo para a gestão de redes que possibilita a implementação de serviços definidos pelo Content Management Interoperability Services (CMIS), permitindo a comunicação entre aplicações de gestão de rede e agentes de gestão. O CMIP e CMIS foram baseados no modelo de gestão de rede especificados pelos ISO e OSI e definido na recomendação ITU-T X.700 [24].

O modelo CMIP realiza a gestão da informação através da gestão de objectos e permitindo a sua modificação ou a realização de acções a esses mesmos objectos. O CMIP também providencia boas medidas de segurança suportando a autorização, o controlo de acesso e *logs* de segurança, assim como o envio de notificações caso sejam detectadas situações anómalas na rede.

Devido à complexidade e requerimento de recursos necessários para os agentes e gestores, a maioria dos dispositivos TCP/IP não suportavam o CMIP sendo este principalmente usado para o suporte de equipamentos de telecomunicações.

2.3. SIMPLE NETWORK MANAGEMENT PROTOCOL

O protocolo *Simple Network Management Protocol* (SNMP) [2] é um protocolo de gestão, definido pelo IETF, frequentemente usado para a monitorização de equipamentos de rede. É um protocolo especificado para a camada de aplicação que possibilita a sua implementação tanto em UDP como em TCP. O protocolo implementa um mecanismo de comunicação entre um agente e um gestor onde os dados de informação são especificados e guardados em módulos MIB (*Management Information Base*) [3]. A *Structural Management Information* (SMI) representa a estrutura da base de dados MIB sob a forma de árvore através de tabelas conceptuais, onde cada recurso gerido é representado por um objecto. A SMI define uma *framework* através da qual um módulo MIB pode ser definido e construído, ou seja, define os componentes constituintes do módulo e a linguagem formal para a descrição dos objectos geridos. A SMI especifica que todos estes objectos devem ter um nome, caracterizado por um *Object Identifier* (OID), uma sintaxe definindo o tipo de dados do objecto e uma codificação descrevendo como a informação associada com o objecto através do formato BER (*Basic Encoding Rules*) para ser transmitido na rede. A norma SMI foi especificada em duas versões SMIv1, na RFC 1155 [25] e SMIv2, na RFC 2578 [26], sendo que a versão mais recente oferece uma sintaxe mais rica e precisa para a definição de módulos MIB.

O protocolo SNMP também foi evoluindo, tendo sido especificadas várias versões. Dessas, as que se tornaram mais populares foram SNMPv1 [2], SNMPv2 [27], SNMPv3 [28]. O SNMPv1 foi a primeira versão do SNMP, normalizado em 1990, e suporta quatro operações básicas: *Get, GetNext, Set* e *Trap*. O SNMPv1 usa um modelo de pergunta-resposta onde a operação *Get* e *GetNext* apenas permite a leitura eficaz de uma só entrada. Este mecanismo tornava o processo para a leitura total da informação de gestão demasiado demoroso e, não tendo implementado um mecanismo pouco robusto em questões de segurança, levou a que a

maioria dos administradores apenas o usassem para a monitorização das variáveis existentes nas MIB e não para a configuração dos parâmetros nos equipamentos.

O SNMPv2, a segunda versão do SNMP, foi normalizado em 1996 e adicionou duas novas operações à versão anterior. O *GetBulk* que permite a leitura mais eficiente de várias entradas da tabela MIB e o *Inform* para a administração da comunicação de gestão. O *Inform* pode ser entendido como um serviço análogo à operação *Trap* mas para a gestão da comunicação. Contudo o novo sistema de segurança *party-based* definido na SNMPv2 foi visto como sendo demasiado complexo o que levou a que o SNMP continuasse a ser maioritariamente usado para a monitorização da informação contida nas MIBs [29]. A RFC 1901 define o *Community-Based Simple Network Management Protocol* (SNMPv2c) [30] que compreende a SMMPv2 sem o seu modelo de segurança onde este é substituído pelo esquema de segurança definido na versão SNMPv1. A RFC 1910 também define outra alternativa à versão SNMPv2 denominada *User-Based Simple Network Management Protocol* (SNMPv2u) [31], que especifica um mecanismo de segurança melhor que da versão SNMPv1 mas sem a excessiva complexidade especificada na norma SNMPv2. Uma variante da SNMPv2u foi comercializada como SNMPv2* e o seu mecanismo de segurança foi posteriormente adoptado como sendo uma das duas *frameworks* de segurança passíveis de serem de implementadas pela versão SNMPv3.

A terceira versão do SNMP (SNMPv3), normalizada em 2004, adicionou várias medidas de segurança às anteriores versões do SNMP. Forneceu um método de autenticação aos gestores e agentes SNMP, encriptação da comunicação, assim como o suporte para a definição de regras de controlo para a definição de que entidades tinham acesso a modificar certos parâmetros nas MIBs.

Apesar de todas as melhorias promovidas no SNMP, este continua a ser nos dias de hoje maioritariamente usado como protocolo para a monitorização de informação, sendo a sua versão SNMPv1 a mais implementada em sistemas de redes [32].

2.4. WEB-BASED ENTERPRISE MANAGEMENT

O *Web-Based Enterprise Management* (WBEM) [9] é uma tecnologia criada pelo DMTF que utiliza o protocolo HTTP para o transporte dos dados de gestão entre o gestor e o agente. Em mais detalhe o WBEM representa os dados de gestão em *Common Information Model* (CIM) [8] e codifica a informação num formato XML denominado CIM-XML [10] que são transportados sobre a camada protocolar HTTP. Cada agente implementa um servidor HTTP [11] para receber os pedidos, um mecanismo para decifrar e responder às instâncias CMI-XML, e um software interno para a recolha dos dados de gestão locais e como representa-los no modelo CIM. Cada gestor necessita de implementar um cliente HTTP para aceder ao agente e bibliotecas para codificar os pedidos em CIM-XML e descodificar as respostas.

O WBEM permite, para além das operações básicas de leitura e escrita dos dados de gestão, a criação de novas instâncias de recursos identificados pelo modelo CIM. O esquema

de representação da informação de gestão também pode ser directamente manipulada usando as operações proporcionadas pelo WBEM.

2.5. NETWORK CONFIGURATION PROTOCOL

O protocolo *Network Configuration Protocol* (NETCONF) [16] define um mecanismo simples através do qual um dispositivo de rede pode ser gerido, os seus dados de configuração recolhidos, e novas configurações especificadas e manipuladas. O protocolo permite o acesso ao dispositivo através de um API (*Application Programming Interface*) para enviar e receber a totalidade, ou apenas uma parte, dos dados de configuração.

O protocolo NETCONF tem como base o paradigma RPC (*Remote Procedure Call*). O cliente, usando uma sessão segura e orientada à ligação, codifica o seu RPC num formato XML e envia-o ao servidor. O servidor responde com um RPC-Reply novamente codificado em XML. Os conteúdos das perguntas e respostas trocados podem ser descritos em XML DTDs (*Document Type Definitions*), XSDs (*XML Schemas*) ou ambos, permitindo que ambas as partes reconheçam as restrições de sintaxe impostas na transferência da informação.

Um aspecto essencial do NETCONF é que replica, de forma semelhante, as funcionalidades nativas presentes nos dispositivos no seu protocolo de gestão. Esta característica permite a diminuição dos custos de implementação e um acesso mais eficiente a novos recursos.

O protocolo NETCONF foi desenvolvido segundo uma arquitectura baseada em quatro camadas protocolares como é ilustrado na Figura 1.

1. A camada de protocolo de transporte providencia a via de comunicação entre o cliente e o servidor.

2. A camada RPC especifica um mecanismo simples e uma estrutura de transporte independente para a codificação dos RPCs.

3. A camada de operações define o conjunto básico de operações codificadas em XML que podem ser invocadas nos métodos RPC.

4. A camada de dados declara os dados de configuração que podem ser acedidos manipulados pelas operações na configuração do dispositivo.

Figura 1: Camadas do protocolo NETCONF

O protocolo NETCONF foi idealizado para poder ser sobreposto sobre qualquer protocolo de transporte que obedeça a um conjunto pré-determinado de requisitos. Sendo o NETCONF orientado à conexão, este requer uma conexão persistente entre as partes. Esta conexão deve prover a entrega fiável e sequencial dos dados transmitidos. O NETCONF delega ao protocolo de transporte a responsabilidade para autenticação, integridade dos dados e confidencialidade da ligação. O protocolo define quatro esquemas para a sua implementação usando determinados protocolos de transporte. São eles: *Secure Shell* (SSH) [33]; *Simple Object Access Protocol* (SOAP) [20]; *Blocks Extensible Exchange Protocol* (BEEP) [34]; *Transport Layer Security* (TLS) [35]. A integração do protocolo de transporte SSH é obrigatória em qualquer implementação que use o NETCONF, embora seja permitido aos elementos negociar a alteração da sessão, durante o processo de troca de capacidades, para outro tipo dos protocolos de transporte acima referidos.

O protocolo NETCONF usa um modelo de comunicação baseado no paradigma *Remote Procedure Call*. Os participantes na sessão NETCONF usam os elementos *<rpc>* e *<rpc-reply>* para permitir um método de transporte, independente do protocolo, para a realização de pedidos e obtenção de respostas NETCONF. O elemento *<rpc>* é usado para encapsular um pedido NETCONF enviado pelo cliente ao servidor. Para além de conter o método que se pretende executar no dispositivo de destino, inclui um atributo obrigatório designado "*message-id*", definido por uma string arbitrária especificada pelo cliente. Essa string normalmente codifica um inteiro que é incrementado sequencialmente a cada mensagem RPC. O receptor da mensagem não necessita de interpretar este parâmetro, apenas salvaguarda o seu valor para o atribuir ao elemento *<message-id>* na sua mensagem *<rpc-reply>* correspondente.

O elemento *<rpc-reply>* pode conter um elemento *<ok>* que tem como função informar o cliente que não ocorreu nenhum erro durante a sessão NETCONF iniciada pelo elemento *<rpc>*. Caso haja a ocorrência de erros o elemento *<ok>* é substituído por um elemento *<rpc-error>*. O elemento *<rpc-error>* é constituído por um conjunto de informações que detalham os erros detectados. Essas informações são especificadas pelos elementos:

- *<error-type>:* que determina em que camada conceptual o erro ocorreu.
- *<error-tag>:* que identifica a condição do erro.

- *<error-severity>:* que identifica a severidade do erro (error ou warning) determinada pelo dispositivo.

- *<error-app-tag>:* que especifica a condição de erro, caso exista, caracterizada pelo modelo de dados ou especifica da implementação.

- *<error-path>:* que contêm a expressão do caminho absoluto XPath identificado pelo nó associado ao erro ocorrido. Este elemento pode não ser apresentado se o elemento transmitido não poder ser associado a uma condição de erro especifica ou se o parâmetro presente na tag <error-info> for suficiente para identificar o nó associado ao erro.

- *<error-message>:* inclui uma string adequada à leitura humana que descreve a condição de erro. Este elemento só será apresentado se tiver sido especificada a mensagem apropriada associada à condição de erro.

- *<error-info>:* que apresenta o atributo ou elementos que originaram o erro. Este elemento pode não estar presente se a condição de erro não tiver sido prevista.

O protocolo NETCONF define um conjunto de nove operações básicas, resumidas na Tabela 1, que permitem gerir as configurações e obter dados de estado do equipamento de rede. Estas operações permitem ler, editar, copiar e apagar os dados de configuração, implementar acesso exclusivo aos dados e mecanismos para terminar sessões NETCONF. O protocolo suporta mecanismos de filtragem em algumas das suas operações com o propósito de facilitar a manipulação de configurações de grande dimensão, permitindo que estas visem apenas uma parte restrita dos dados.

Tabela 1: Operações base do NETCONF

Operação	Descrição
<get-config>	Obtém toda ou parte de uma configuração presentes num equipamento.
<edit-config>	Edita parcialmente ou totalmente a configuração da base de dados especificada. Define operações para actualizar, criar, apagar ou fazer o *merge* da configuração actual com os novos dados que estão sendo enviados encapsulados no elemento *<editconfig>*.
<copy-config>	Copia inteiramente uma nova configuração ou subscreve a actual, se permitido.
<delete-config>	Apaga a configuração indicada.
<lock>	Retém o acesso à configuração indicada. Utilizado para garantir o acesso exclusivo à configuração e evitar que outras sessões tenham acesso a esta mesma configuração.
<unlock>	Liberta a configuração para que outras sessões a possam usar.
<get>	Obtém a configuração definida no equipamento e os dados de estado do dispositivo.
<close-session>	Termina a própria sessão, libertando todos os recursos associados a ela.
<kill-session>	Força o término de uma sessão NETCONF através da indicação do número de identificação da sessão que se deseja encerrar.

A operação *<get-config>* permite obter parte ou a totalidade de um conjunto de dados de configuração presentes num equipamento. O nome do conjunto a ser consultado é identificado pelo parâmetro *source* podendo este conjunto ser filtrado pela variável *filter*. O *<rpc-reply>* correspondente incluirá um elemento *<data>* contendo o conjunto de dados requerido.

O NETCONF permite que a configuração de um dispositivo possa ser modificada. Esta funcionalidade é realizada pela operação *<edit-config>* que permite a alteração de parte ou a totalidade dos dados de configuração de um equipamento. Os elementos pertencentes à subárvore *<config>* podem conter um atributo '*operation*'. Este atributo identifica o local na configuração onde deve ser realizada a operação podendo ser indicados os valores *merge, replace, create* e *delete*. Se o atributo não for especificado explicitamente é realizada a operação *merge.* Esta operação insere os novos dados de configuração sem que haja a alteração dos dados já existentes. A acção *replace* substitui todos os dados pelos novos dados que se pretendem inserir. O atributo *delete* remove os dados especificados pelo elemento *<config>*. Finalmente a operação *create* apenas insere os dados com a condição de esses mesmos dados ainda não existirem no dispositivo. Estas acções podem igualmente ser definidas através do parâmetro *default-operation*. A operação *<edit-config>* compreende também os parâmetros target, para a identificação do conjunto de dados a ser editado, *config* para a representação da hierarquia de dados a configurar, *test-option* para a realização de testes de validação e *error-option* para indicar se a operação será ou não abortada na ocorrência de um erro.

A operação *<copy-config>* permite criar ou substituir a configuração completa de um repositório de dados para outro. Recebe como parâmetros as variáveis *source* e *target* que indicam, respectivamente, o repositório de dados de origem e destino.

A operação *<delete-config>* possibilita apagar um repositório de dados identificado pelo parâmetro *target*. O NETCONF, pelos motivos óbvios, não permite que o repositório de dados *<running>* possa ser apagado.

O NETCONF define um mecanismo de acesso exclusivo aos dados de configuração através da operação *<lock>*. Esta permite que um cliente obtenha um acesso restrito à configuração de um dispositivo identificado pelo argumento obrigatório *target*. Este mecanismo permite que o cliente realize a alteração dos diferentes parâmetros de configuração sem receio que ocorra o acesso concorrencial por parte de outros cliente a esses mesmos parâmetros.

A operação *<unlock>* permite a libertação do acesso exclusivo atribuído a um conjunto de dados previamente definido por uma operação *<lock>*. Recebe como parâmetros o nome do conjunto de dados de configuração que deverá ser libertado.

A obtenção das duas classes de dados (configuração e de estado) é realizada pela operação *<get>*. Esta, tal como a operação *<get-config>*, também permite a filtragem desses dados pelo parâmetro *filter*. O *<rpc-reply>* correspondente incluirá um elemento *<data>* contendo o conjunto de dados requerido.

As operações *<close-session>* e *<kill-session>* permitem a terminação de sessões NETCONF. A operação *<close-session>* permite o encerramento da sessão de uma forma mais aprazível. O servidor quando recebe esta operação liberta todos *locks* e recursos associados pendentes, sendo também terminadas as conexões associadas com essa sessão. Consequentemente qualquer operação NETCONF enviada posteriormente será ignorada.

A operação *<kill-session>* força a terminação de uma sessão NETCONF. Quando, numa sessão aberta, uma entidade na rede recebe um pedido *<kill-session>* são abortadas todas as operações que estejam a ser processadas nesse momento, são ainda libertados todos os acessos restritos associados aos recursos e fechadas todas as conexões inerentes da sessão. A operação recebe como parâmetro um *session-id* indicando a sessão NETCONF a ser encerrada.

No caso de os pedidos, referentes às operações acima detalhadas, não possam ser satisfeitos por um qualquer motivo o *<rcp-replay>* a ser enviado deverá incluir um elemento *<rcp-error>* informando a causa, ou causas, da sua não realização. Exceptuando as operações *<get>* e *<get-config>*, onde é mencionado que é enviado um elemento *<data>* no seu correspondente *<rcp-replay>*, as restantes operações, na sequência de uma resposta positiva, deverão incluir, no seu *<rcp-reply>*, um elemento *<ok>* para indicarem que o respectivo pedido foi satisfeito.

A RFC 5717 [36] define adicionalmente duas novas operações ao protocolo, *<partial-lock>* e *<partial-unlock>*, que implementam um mecanismo de acesso parcial aos dados de configuração através de um filtro definido por uma expressão *XPath*. O NETCONF implementa também um mecanismo de notificações [37] baseado em outras duas operações adicionais. O cliente pode subscrever a recepção de notificações de um dado equipamento através da operação *<create-subscription>*. Quando um determinado evento ocorre o agente NETCONF notifica o cliente da sua ocorrência através da operação *<notification>*. A operação de subscrição de notificações permite especificar um filtro a ser aplicado pelo servidor para a definição dos eventos que este deverá enviar. Isto previne o servidor de enviar eventos sobre os quais o cliente não tem interesse em receber, promovendo a escalabilidade de um sistema NETCONF.

O NETCONF possibilita ao cliente a descoberta do conjunto de extensões do protocolo suportado pelo servidor. Estas capacidades (*Capabilities*) permitem ao cliente ajustar o seu modo de acção para tirar partido desses recursos expostos pelo equipamento. Cada capacidade, que é identificada por um URI (*Uniform Resource Identifier*) único, é a descrição de um conjunto de funcionalidades adicionais à especificação base do NETCONF. Essas capacidades aumentam as operações primárias do dispositivo, descrevendo operações adicionais e o conteúdo permitido nessas mesmas operações. O cliente pode descobrir as capacidades do servidor e usar qualquer operação adicional, parâmetros ou conteúdos definidos nessas mesmas capacidades. O anúncio das capacidades suportadas por cada um dos elementos (cliente e servidor) é realizado no início da sessão NETCONF através da sua inclusão nas mensagens *<hello>* trocadas entre ambos. As capacidades adicionais podem ser definidas a qualquer momento em documentos externos, permitindo assim que o conjunto de capacidades do sistema possa ser expandido ao longo do tempo.

A RFC 4741 define um conjunto de funcionalidades opcionais que podem ser anunciadas no processo de troca de capacidades, mas que não foram incluídas como funcionalidades mandatórias pelo protocolo. A lista seguinte descreve essas capacidades e definem operações que ambas as entidades devem suportar.

- *Writable-Running*: indica que o dispositivo suporta a manipulação directa do repositório running.
- *Candidate-Configuration*: indica que o dispositivo suporta o repositório candidate permitindo a manipulação da configuração sem que haja consequências directas no seu comportamento.
- *Confirmed-Commit*: permite que as alterações efectuadas na configuração de um equipamento possam ser revertidas se não for realizado a operação <*commit*> num tempo preestabelecido.
- *Rollback-on-Error*: indica que um servidor suporta a opção *rollback-on-error* na operação <*edit-config*>.
- *Validate*: indica que um servidor suporta a validação da configuração presente no repositório candidate.
- *Distinct-Startup*: designa que um servidor suporta a distinção entre as configurações de *startup* e *running*, permitindo que operações efectuadas na configuração *running* não tenham efeito nas configurações de *startup*.
- *URL*: esta capacidade divulga o suporte de expressões URL em operações que incluam os argumentos *source* e *target*.
- *XPath*: indica que a entidade suporta expressões XPath que podem ser aplicadas no elemento <*filter*> nas operações que o incluam.

Os dados que podem ser obtidos de um sistema em execução encontram-se separados em duas classes: dados de configuração e dados de estado. As duas classes de dados por sua vez são enquadradas na camada de dados e manipulados pelas camadas inferiores.

O conjunto de dados de configuração permite a sua escrita e consequentemente podem ser alterados pelo administrador. São também considerados dados de configuração, dados que podem levar um dispositivo qualquer a transitar de um estado de configuração para outro. Os dados de estado são os dados que somente possuem permissão de leitura, representando basicamente informações sobre o estado actual do dispositivo e informações estatísticas sobre o sistema. O protocolo reconhece a distinção entre estas duas classes onde a operação <*get-config*> devolve apenas os dados de configuração enquanto a operação <*get*> retorna todos os dados, agrupando os dados de configuração e de estado.

O protocolo segue uma abordagem *document-oriented* assumindo que a configuração de um dispositivo possa ser representada num documento que pode ser obtido, manipulado ou copiado. O NETCONF suporta três tipos de repositórios de configuração. O repositório *running* representa a configuração que no momento está a ser executada pelo equipamento. Este pode ser transferido para o repositório *startup*, pela operação <*copy-config*>, sendo esse repositório carregado sempre que se inicia o equipamento. O NETCONF suporta ainda o repositório *candidate*, permitindo a manipulação temporária dos diferentes elementos de

configuração presentes nesse repositório que, posteriormente, podem ser cometidos e aplicados no repositório *running* pela operação *<commit>*.

O modelo de dados e os aspectos relacionados com ele estão fora do âmbito do protocolo NETCONF. É pressuposto que o modelo de dados do dispositivo seja bem conhecido pela aplicação e que ambas as partes estejam cientes de questões como o seu *layout*, conteúdo, activação, pesquisa, substituição e a gestão dos dados, bem como quaisquer outras restrições impostas por ele. O NETCONF transporta os dados de configuração, dentro do elemento *<config>*, específico do modelo de dados suportado pelo dispositivo. O dispositivo usa as capacidades para anunciar o conjunto de modelos de dados que este pode implementar e a definição dessas capacidades detalham as operações e restrições impostas pelo modelo de dados. Existe ainda a possibilidade de suporte de múltiplos modelos de dados por parte de um equipamento, podendo estes modelos ser *standards* ou proprietários.

2.5.1. YANG

O YANG [17] é uma linguagem de modelação de dados para o protocolo NETCONF, que permite a descrição da hierarquia dos nós de dados e restrições existentes entre eles. O YANG define o modelo de dados, bem como a forma de manipular esses dados através das operações presentes no protocolo NETCONF.

O NETCONF permite o acesso as capacidades nativas dos dispositivos existentes numa rede, definindo métodos para a manipulação dos repositórios de dados de configuração, resgatar dados operacionais e invocar operações especificas. O YANG fornece métodos para definir o conteúdo transportado pelo NETCONF, isto é dados e operações. Utilizando estas tecnologias conjuntamente, módulos normalizados podem ser definidos permitindo interoperabilidade e homogeneidade entre dispositivos, possibilitando ao mesmo tempo a exposição das capacidades únicas inerentes de cada dispositivo.

O YANG permite a um administrador criar um modelo de dados, definir a organização dos dados no modelo e definir restrições nesses mesmos dados. Uma vez definido, o módulo YANG age como um contracto entre o cliente e o servidor, onde ambas as partes compreendem o seu par e como este irá operar. O cliente tem o conhecimento de como criar informação valida a ser transmitida ao servidor e compreende os dados enviados pelo servidor. O servidor conhece as regras para a manipulação dos dados e como estes devem ser processados.

Os módulos YANG organizam hierarquicamente os dados em árvore onde cada nó é definido pelo seu nome e o seu respectivo valor ou um conjunto de nós descendentes. O YANG permite uma descrição clara e concisa dos seus nós assim como a sua interacção com outros nós. Os modelos de dados YANG podem ser estruturados em módulos ou submódulos. Um módulo pode importar dados de módulos externos ou incluir dados de outros submódulos. Os módulos YANG podem descrever restrições a serem aplicadas aos dados, condicionando a sua representação ou valor baseando-se na presença ou valor de outros nós na sua hierarquia. O

YANG implementa um conjunto predefinido de tipos de dados, fornecendo também mecanismos para a criação de novos tipos. A linguagem incorpora mecanismos de extensibilidade e flexibilidade não presentes em outras linguagens de modelos de dados. Novos módulos podem ser criados para aumentar a hierarquia de dados definida em outros módulos, através da introdução dos dados adicionais pretendidos, nos locais apropriados, na organização de dados já definida. O YANG também permite a extensão da própria linguagem através da definição de novos procedimentos.

A sintaxe YANG é similar à SMIng [38] e às linguagens de programação como C e C++, promovendo uma mais fácil definição e legibilidade do código. Os *tokens* YANG compreendem palavras reservadas, strings, ponto e virgula (';') ou chavetas ('{' ou '}'). As strings podem ser representadas entre aspas ('"<string>"), plicas ('""<string>"""') ou simplesmente por uma palavra singular. As palavras reservadas compreendem o conjunto de palavras definidas para a especificação das instruções YANG ou por um prefixo seguido de dois pontos (':') e de uma palavra definindo uma extensão da linguagem.

Um módulo YANG é definido por uma sequência de instruções. Cada instrução é especificada pela sua palavra reservada, seguida de zero ou um argumentos, seguido por ';' ou um bloco de sub-instruções contidas entre chavetas. A sua representação em formato *Extended Backus-Naur Form* (EBNF) é ilustrada na Figura 2.

```
statement = keyword [argument] (";" / "{" *statement "}")
```

Figura 2: Instruções YANG em formato EBNF

A instrução 'module' define o nome do módulo e agrupa todas as instruções que pertencem a esse módulo, detalhando toda a informação relativa a este. Um módulo contém três tipos de instruções: *module-header statements, revision statements* e *definition statements*. As instruções do tipo *module-header* descrevem o módulo e fornecem informações sobre este, os *revision statements* detalham informações sobre o seu histórico e evolução e as *definition statements* definem o corpo do módulo onde toda a modelação de dados é definida.

Seguidamente são detalhadas as instruções da linguagem YANG mais importantes onde são ilustrados por um excerto de código exemplificativo da sua sintaxe e a sua correspondente representação em XML que é incluída no envio das mensagens NETCONF.

A instrução '*leaf*', ilustrada na Tabela 2, é usada para a definição de nós folha na árvore de dados que representa um módulo YANG. Tem como argumento um identificador seguido por um bloco de sub-instruções que detalham a informação da folha. Um nó *leaf* representa um valor e não adiciona nós descendentes na árvore de dados.

Tabela 2: Instrução Leaf

YANG	NETCONF XML

leaf host-name {
type string;
description "Hostname for this
system";
}
	<host-name>
	my.example.com
	</host-name>

A instrução '*leaf-list*', exibida na Tabela 3, é usada para definir uma sequência de nós *leaf* contendo apenas um valor de um tipo particular por cada folha. Este recebe como argumento um identificador, que especifica o nome da lista, seguido por um bloco de sub-instruções que detalham a sua informação.

Tabela 3: Instrução Leaf-List

YANG	NETCONF XML

leaf-list domain-search {
type string;
description "List of domain names to search";
}
	<domain-search> high.example.com
	</domain-search>
	<domain-search> low.example.com
	</domain-search>
	<domain-search> everywhere.example.com
	</domain-search>

A instrução '*container*' é usada para definir um grupo de nós numa sub-arvore. Tem como argumento um identificador que representa o nome do conjunto, seguido por um bloco de sub-instruções. Um nó *container* não expressa um valor mas sim uma lista de nós descendentes na árvore de dados. Estes nós descendentes são definidos nas suas sub-instruções. Um *container* pode conter qualquer número de nós descendentes podendo estes ser de qualquer tipo, incluindo *leafs, lists, containers* e *leaf-lists*. A Tabela 4 apresenta um exemplo da sua utilização.

Tabela 4: Instrução Container

YANG	NETCONF XML

container system {
container login {
leaf message {
type string;
description
"Message given at start of login session";
}
}
}
	<system>
	<login>
	<message>Good morning</message>
	</login>
	</system>

A instrução '*list*' é usada para definir o conteúdo de um nó de dados. Recebe como argumento um identificador que define o nome da lista seguido por um bloco de sub-instruções que detalham a sua informação. A lista é identificada singularmente pelo valor das suas *key leafs*. A lista pode ser definida por múltiplas *keys leafs* podendo conter um número não determinado de nós descendentes de qualquer tipo, incluindo *leafs, lists, containers*, etc. A lista pode também existir em múltiplas instancia na árvore de dados. Um exemplo da sua utilização encontra-se ilustrado na Tabela 5.

Tabela 5: Instrução List

YANG	NETCONF XML

list user {
  key "name";
  leaf name {
  type string;
  }
  leaf full-name {
  type string;
  }
  leaf class {
  type string;
  }
}
	<user>
	  <name>glocks</name>
	  <full-name>Goldie Locks</full-name>
	  <class>intruder</class>
	</user>
	<user>
	  <name>snowey</name>
	  <full-name>Snow White</full-name>
	  <class>free-loader</class>
	</user>
	<user>
	  <name>rzell</name>
	  <full-name>Rapun Zell</full-name>
	  <class>tower</class>
	</user>

A linguagem YANG permite, para além dos dados de configuração, a modelação de dados estado. Estes dados podem ser identificados pela instrução '*config*'. Esta recebe como argumentos as strings '*true*' ou '*false*' identificando se os dados pertencentes à sua hierarquia representam, respectivamente, dados de configuração ou de estado. Isto influencia directamente a operação NETCONF *<get-config>* uma vez que esta apenas retorna os dados de configuração. Para que todos os dados, configuração e de estado, possam ser colectados será necessário o uso da operação *<get>*.

Similarmente à maioria das linguagens de programação, o YANG providencia um conjunto de tipos de dados predefinidos, embora alguns destes reflictam características particulares inerentes ao domínio da problemática de gestão. A Tabela 6 reúne todos os tipos predefinidos da linguagem YANG.

Tabela 6: YANG Built-In Types

Nome	Definição
binary	Sequencia de octetos
bits	Conjunto de bits ou flags
boolean	Valor booleano ('true' ou 'false')
decimal64	Número decimal com sinal em 64-bit
empty	Define que uma dada leaf que não contém qualquer valor
enumeration	Valores de um conjunto de nomes designados
identityref	Referência para uma identidade abstracta
instance-identifier	Referência para um nó na árvore de dados
int8	Inteiro com sinal em 8-bit
int16	Inteiro com sinal em 16-bit
int32	Inteiro com sinal em 32-bit
int64	Inteiro com sinal em 64-bit
leafref	Referência para uma instância leaf
string	Conjunto de caracteres para a leitura humana
uint8	Inteiro sem sinal em 8-bit
unit16	Inteiro sem sinal em 16-bit
uint32	Inteiro sem sinal em 32-bit
uint64	Inteiro sem sinal em 64-bit
union	Valor correspondente a um tipo definido pelos seus membros

Como já anteriormente referido a linguagem YANG permite a definição de novos tipos. Estes tipos podem ser usados localmente no módulo, em módulos e submódulos que o incluam ou em outros módulos que o importem. A instrução '*typedef*' permite a especificação desses novos tipos, que são derivados de tipos base. Estes tipos bases podem ser os tipos predefinidos do YANG ou um tipo derivado definido por outra instrução '*typedef*'. A instrução recebe como argumento um identificador que define o seu nome seguido por um bloco de sub-instruções que detalham a informação do novo tipo. Um excerto exemplificativo é mostrado na Tabela 7.

Tabela 7: Instrução Typedef

YANG	NETCONF XML

typedef percent {
  type uint8 {
  range "0 .. 100";
  }
  description "Percentage";
}
leaf completed {
  type percent;
}
	<completed>20</completed>

O YANG facilita a reutilização de código através da instrução '*grouping*'. Esta instrução é usada para definir um bloco de nós reutilizável, podendo ser vista como a definição de uma estrutura ou *record* numa convencional linguagem de programação. A reutilização pode ser realizada no próprio módulo ou em módulos que o incluam ou importem. A instrução recebe como argumento um identificador seguido por um bloco de sub-instruções detalham a informação referente ao grupo. Uma vez definido este pode ser referenciado através da instrução '*uses*'. Esta instrução permite ainda certas informações, contidas no grupo, possam ser rescritas. Um exemplo da utilização da instrução *grouping* é ilustrado na Tabela 8.

Tabela 8: Instrução Grouping

YANG	NETCONF XML

grouping target {
  leaf address {
  type inet:ip-address;
  description "Target IP address";
  }
  leaf port {
  type inet:port-number;
  description "Target port number";
  }
}
container peer {
  container destination {
  uses target;
  }
}
	<peer>
	  <destination>
	  <address>192.0.2.1</address>
	  <port>830</port>
	  </destination>
	</peer>

O modelo de dados YANG permite a definição de agregados de alternativas, onde apenas uma pode existir num dado momento. Esta operação é realizada pela instrução '*choice*', que recebe como argumento um identificador que é usado para identificar o nó *choice* na estrutura da árvore. A instrução *choice* consiste num número de ramos definidos pela instrução '*case*'. Cada instrução *case* pode conter múltiplos nós, mas cada nó apenas pode

aparecer em um *case* numa hierarquia *choice*. Quando um elemento referente a um *case* é criado todos os outros elementos definidos pelas restantes instruções *case* são implicitamente removidos. A Tabela 9 seguinte demonstra um exemplo da sua utilização.

Tabela 9: Instrução Choice

YANG	NETCONF XML

container food {
choice snack {
case sports-arena {
leaf pretzel {
type empty;
}
leaf beer {
type empty;
}
}
case late-night {
leaf chocolate {
type enumeration {
enum dark;
enum milk;
enum first-available;
}
}
}
}
}
	<food>
	<pretzel/>
	<beer/>
	</food>

Como foi referido anteriormente o YANG permite a extensão do modelo de dados com a inserção de nós adicionais, quer no próprio módulo, quer em módulos externos. Isto permite, por exemplo, a adição de novos parâmetros em módulos standards promovendo a sua interoperabilidade. A instrução '*augment*' define a localização na hierarquia do modelo de dados onde os novos nós são inseridos e onde a instrução '*when*' permite a expressão de condições de quando estes novos nós são válidos. O argumento da instrução '*augment*' é expressa por uma *string* identificando o caminho para o nó no esquema da árvore. Este nó é denominado por *target node*, podendo este referenciar um elemento *container, list, choice, case, input, output* ou *notification*. Este elemento pode seguidamente ser caracterizado através dos nós definidos nas sub-instruções expressas na instrução *augment*. Um exemplo da sua utilização é ilustrado na Tabela 10.

Tabela 10: Instrução Augment

YANG	NETCONF XML

augment /system/login/user {
when "class != 'wheel'";
leaf uid {
type uint16 {
range "1000 .. 30000";
}
}
}
	<user>
	<name>alicew</name>
	<full-name>Alice N. Wonderland</full-name>
	<class>drop-out</class>
	<other:uid>1024</other:uid>
	</user>

As funcionalidades básicas do NETCONF também podem ser expandidas pela linguagem YANG. O YANG permite a definição de novas operações e notificações através das instruções '*rpc*' e '*notification*', respectivamente. Estas permitem a sua caracterização pormenorizada

através das suas sub-instruções respectivas. As Tabelas 11 e 12 ilustram excertos de código exemplificativos para a criação de novas operações rpc e notificações.

Tabela 11: Instrução RPC

YANG	NETCONF XML

rpc activate-software-image {
  input {
  leaf image-name {
  type string;
  }
  }
  output {
  leaf status {
  type string;
  }
  }
}
	<rpc message-id="101"
	xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
	  <activate-software-image
	xmlns="http://acme.example.com/system">
	  <image-name>acmefw-2.3</image-name>
	  </activate-software-image>
	</rpc>
	
	<rpc-reply message-id="101"
	xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
	  <status
	xmlns="http://acme.example.com/system">
	  The image acmefw-2.3 is being installed.
	  </status>
	</rpc-reply>

Tabela 12: Intrução Notification

YANG	NETCONF XML

notification link-failure {
  description
  "A link failure has been detected";
  leaf if-name {
  type leafref {
  path "/interface/name";
  }
  }
  leaf if-admin-status {
  type admin-status;
  }
  leaf if-oper-status {
  type oper-status;
  }
}
	<notification
	xmlns="urn:ietf:params:netconf:capability:notification:1.0">
	  <eventTime>2007-09-01T10:00:00Z</eventTime>
	  <link-failure
	xmlns="http://acme.example.com/system">
	  <if-name>so-1/2/3.0</if-name>
	  <if-admin-status>up</if-admin-status>
	  <if-oper-status>down</if-oper-status>
	  </link-failure>
	</notification>

Os módulos YANG podem ser traduzidos para um formato XML equivalente denominado *YANG Independent Notation* (YIN), permitindo que aplicações que usem *parsers* XML ou *scripts Extensible Stylesheet Language Transformations* (XSLT) possam operar sobre esses módulos. A tradução de YANG para YIN é bidireccional e reversível. O mapeamento entre formatos YANG e YIN não modifica a informação contida no modelo. A notação YIN permite a representação de módulos YANG em XML possibilitando o uso de ferramentas de processamento de ficheiros em formato XML para a filtragem dos dados e validação, geração automática de código, documentação, entre outras tarefas. A Tabela 13 ilustra a conversão entre um módulo YANG e a sua correspondente representação no formato YIN.

Tabela 13: Comparação da sintaxe YANG e YIN

YANG	YIN

module acme-foo {
 	
namespace "http://acme.example.com/foo";
 	
prefix "acfoo";
import my-extensions {
prefix "myext";
}
list interface {
key "name";
leaf name {
type string;
}
leaf mtu {
type uint32;
 	
description "The MTU of the interface.";
 	
myext:c-define "MY_MTU";
}
}
}
	<module name="acme-foo"
	xmlns="urn:ietf:params:xml:ns:yang:yin:1"
	xmlns:acfoo="http://acme.example.com/foo"
	xmlns:myext="http://example.com/my-extensions">
	<namespace
	uri="http://acme.example.com/foo"/>
	<prefix value="acfoo"/>
	<import module="my-extensions">
	<prefix value="myext"/>
	</import>
	<list name="interface">
	<key value="name"/>
	<leaf name="name">
	<type name="string"/>
	</leaf>
	<leaf name="mtu">
	<type name="uint32"/>
	<description>
	<text>The MTU of the interface.</text>
	</description>
	<myext:c-define name="MY_MTU"/>
	</leaf>
	</list>
	</module>

2.6. IMPLEMENTAÇÕES NETCONF

Apesar de a tecnologia de gestão NETCONF ser ainda uma tecnologia emergente, esta tem sido progressivamente aceite pela comunidade associada ao domínio de redes, recebendo uma cada vez maior atenção por parte de empresas e institutos ou universidades. As próximas secções detalham algumas ferramentas já existentes relativas ao protocolo que permitem a implementação de aplicações baseadas em NETCONF.

2.6.1. YUMA

A *Yuma Tools* [39] é um completo SDK desenvolvido pelo NETCONF CENTRAL, Inc. que inclui um *daemon* NETCONF (netconfd), um cliente que opera através de comandos introduzidos numa consola, várias ferramentas de manipulação de módulos YANG e uma documentação bastante completa. Implementa uma completa solução de desenvolvimento, desde a inclusão de ferramentas que permitem a tradução de ficheiros MIB para módulos YANG, que permitem a tradução desses módulos para YIN e XSD, a geração de *skeletons* em linguagem C e ferramentas para a criação de documentação do projecto. As aplicações geradas pelo Yuma usam o *daemon* NETCONF que carrega o código desenvolvido como um módulo para a manipulação de objectos definidos pelo utilizador. Uma vez que a aplicação segue uma arquitectura modular, o cliente criado anteriormente pode proceder à alteração dos objectos definidos pelo utilizador sem que seja necessário qualquer processo de

recompilação. A aplicação implementa uma comunicação sobre SSH e permite o uso de todas as operações NETCONF definidas nas NETCONF RFCs 4741 e 5277.

2.6.2. NETOPEER

O Netopeer [40] é uma implementação *open-source* para o protocolo NETCONF criado pelo CESNET *librouter* Project para fornecer um meio de configuração de um elemento de monitorização denominado FlowMon. Foi desenvolvido em linguagem C e realiza a completa implementação do protocolo NETCONF descrito na RFC 4741, permitindo a gestão de equipamento de rede baseado no protocolo NETCONF sobre a camada de transporte SSH. O gestor inclui uma interface baseada em comandos para a realização da gestão através da consola podendo também, esta gestão, ser realizada com o auxílio de um *browser*. O Netopeer permite a configuração da *probe* nos repositórios *startup* e *running*, podendo realizar a completa transferência de informação entre ambos, antes de qualquer edição dos dados de configuração. Segue uma abordagem de modelação de dados em RELAX-NG [41] uma vez que foi desenvolvido anteriormente à especificação da linguagem YANG.

2.6.3. YENCA E YENCA P

O YENCA [42] foi a primeira implementação NETCONF a ser desenvolvida. A primeira versão foi lançada em 2004 pelos laboratórios LORIA-INRIA. Consiste na implementação de um agente descrito em linguagem C e um gestor em Java. Fornece um suporte para a gestão de interfaces de rede e de tabelas de encaminhamento. Implementa a comunicação sobre uma camada de transporte não normalizada de SOAP sobre TCP. Mais tarde os laboratórios LORIA-INRIA aperfeiçoaram a arquitectura da sua plataforma e o suporte para a realização da gestão que denominaram *EnSuite* [43]. Foi desenvolvido um novo agente NETCONF, chamado YENCAP, desenvolvido em python e um novo gestor *Web-based* que suporta a comunicação entre ambos sobre SSH. A plataforma EnSuite suporta ainda a integração de modelos de dados YANG, mas não permite o envio ou recepção de notificações NETCONF.

2.6.4. NETCONF4ANDROID

O Netconf4Android [44] implementa um API para um cliente NETCONF. Foi desenvolvido em Java e usa o SSH como camada de transporte. A aplicação permite a integração com o YUMA SDK, suporte módulos descritos em formato YANG e criação automática de *skeletons* para a implementação de uma aplicação distribuída para o seu uso numa plataforma *Android*.

2.6.5. NCCLIENT

A aplicação ncclient [45] é uma biblioteca escrita em *python* que auxilia o desenvolvimento e *scripting* de clientes baseados no protocolo NETCONF. Implementa uma aplicação NETCONF sobre transporte SSH e suporta todas as capacidades e operações descritas na RFC 4741. A API é integrada com gestor NETCONF também desenvolvido em *python open source* e distribuída sobre uma licença *Apache*.

2.7. IMPLEMENTAÇÕES PARA O PROCESSAMENTO DE MÓDULOS YANG

Apesar do seu curto período de vida, a linguagem YANG tem disponível várias implementações que permitem o seu processamento e funcionalidades para a sua tradução para outros formatos, facilitando o seu desenvolvimento e a sua interoperabilidade com outros sistemas.

O Jyang [46] é um YANG *parser* desenvolvido em regime de software aberto em linguagem Java por Emmanuel Nataf da Madynes Inria. Consiste numa aplicação baseada em linha de comandos, que permite o parser de módulos ou submódulos YANG, podendo ser realizada a sua respectiva conversão para o formato YIN. A distribuição inclui um API bem documentado permitindo que a aplicação possa ser usada programaticamente.

A aplicação Pyang [47] consiste num validador e conversor de módulos YANG escrito em *python* em consonância com a YANG RFC. Permite a tradução de documentos YANG para YIN e vice-versa, conversões de YANG para DSDL e XSD e a criação de documentação UML baseada nos módulos YANG. Adicionalmente inclui um *plug-in Framework* para geração automática de código.

O Smidump faz parte do projecto LIBSMI [48], desenvolvido na Universidade Técnica de Braunschweig por Frank Strauss. O programa Smidump permite a conversão de conteúdos contidos em módulos MIB ou PIB em diversos formatos. Estes formatos incluem a representação do módulo numa árvore de nós, os seus tipos ou módulos importados, como a tradução para os formatos SMIv1, SMIv2, SPPI, código C, YANG, entre outros. O Smidump pode também ser usado para a conversão de módulos SMIv2 para SMIng ou vice-versa, ou para o desenvolvimento de *templates* de agentes.

2.8. SUMÁRIO

No inicio deste capitulo foi realizada um breve introdução as tecnologias de gestão de redes sendo descritas algumas das suas características funcionais. Seguidamente foram analisadas as tecnologias para a gestão de redes mais importantes, como o CMIP, o SNMP, o WBEM e em mais particular o protocolo NETCONF e a sua linguagem de modelação de dados YANG.

O protocolo NETCONF define um mecanismo simples através do qual um dispositivo de rede pode ser gerido, os seus dados de configuração recolhidos, e novas configurações especificadas e manipuladas. Este protocolo define um conjunto de nove operações básicas, que permitem gerir as configurações e obter dados de estado do equipamento de rede. Estas operações permitem ler, editar, copiar e apagar os dados de configuração, implementar acesso exclusivo aos dados e mecanismos para terminar sessões NETCONF. O NETCONF possibilita ao cliente a descoberta do conjunto de extensões do protocolo suportado pelo servidor. Estas são realizadas através do mecanismo de troca de capacidades que aumentam as operações primárias do dispositivo, descrevendo operações adicionais e o conteúdo permitido nessas mesmas operações. O protocolo segue uma abordagem *document-oriented* assumindo que a configuração de um dispositivo possa ser representada em um documento, que pode ser obtido, manipulado ou copiado, sendo que o protocolo define o suporte de três tipos de repositórios de configuração, *running*, *startup* e *candidate.*

O YANG é uma linguagem de modelação de dados para o protocolo NETCONF, que permite a descrição da hierarquia dos nós de dados e restrições existentes entre eles. O YANG define o modelo de dados, bem como a forma de manipular esses dados através das operações presentes no protocolo NETCONF. A linguagem incorpora mecanismos de extensibilidade e flexibilidade não presentes em outras linguagens de modelos de dados. Novos módulos podem ser criados para aumentar a hierarquia de dados definida em outros módulos, através da introdução dos dados adicionais pretendidos, nos locais apropriados, na organização de dados já definida. O YANG também permite a extensão da própria linguagem através da definição de novos procedimentos. Os módulos YANG podem ser traduzidos para um formato XML equivalente, bidireccional e reversível, denominado YANG *Independent Notation* (YIN), permitindo que aplicações que usem parsers XML ou *scripts Extensible Stylesheet Language Transformations* (XSLT) possam operar sobre esses módulos.

No fim deste capítulo, foram detalhadas várias implementações de aplicações NETCONF e para o processamento de módulos YANG não comerciais existentes à data da escrita do presente documento, que documentam o trabalho desenvolvido na área em que o presente trabalho se insere.

CAPÍTULO 3.
EDITOR YANG SOBRE ECLIPSE

O objectivo deste trabalho consistiu na criação de uma ferramenta que permite o desenvolvimento integrado de aplicações de gestão distribuídas baseadas na tecnologia NETCONF.

Para a definição do modelo de dados em YANG foi desenvolvido um *plug-in* para o IDE Eclipse, que integrado com outras ferramentas já existentes para o processamento de módulos YANG, conjuntamente com aplicações que permitem a geração e implementação de *Web Services,* proporcionam um sistema de edição de aplicações de gestão de rede baseadas no protocolo NETCONF.

Este capítulo detalha todo o processo de desenvolvimento do *plug-in* Editor YANG, desde a definição dos requisitos propostos, à analise da plataforma Eclipse, desenvolvimento das funcionalidades de edição, integração com as suas ferramentas complementares, e descrição de todo o método de geração do sistema de gestão.

3.1. REQUISITOS

Os requisitos predefinidos para o editor foram:

1. Que implementasse as funcionalidades básicas de um editor normal de uma linguagem de programação; Que incluísse as funcionalidades de realce de sintaxe e de *outline*; Que validasse e efectuasse o de relatório de erros presentes no código.

2. Que permitisse a criação de projectos e módulos YANG.

3. Que permitisse a importação de módulos MIB para a linguagem YANG.

4. Que integrasse funcionalidades de tradução de um módulo YANG para os formatos YIN e XSD.

5. Que proporcionasse um ambiente para o completo desenvolvimento de aplicações de gestão distribuídas, desde a criação do *skeleton*, instalação do serviço e teste do sistema dentro do próprio IDE.

3.2. ESTRUTURA DO ECLIPSE

O Eclipse [21] é um IDE desenvolvido em java numa estrutura extremamente modular, que implementa diversas ferramentas de edição para as mais várias linguagens de

programação. Este IDE inclui um mecanismo de extensão das suas funcionalidades através de um sistema de *plug-ins* e oferece um excelente suporte e documentação, facilitando o desenvolvimento destes componentes por parte dos seus utilizadores.

A plataforma Eclipse fornece um modelo normal interface de utilizador (UI) para integração das suas diversas ferramentas. Foi desenvolvida para poder ser executada em diversos sistemas operativos providenciando uma integração robusta com sistemas operativos distintos que possam estar subjacentes. No seu núcleo encontra-se uma arquitectura para a descoberta dinâmica, carregamento e execução de *plug-ins*. Cada *plug-in* foca um determinado domínio sendo responsável pela realização das suas tarefas inerentes.

A plataforma Eclipse segue um modelo de *workbench* para integração das ferramentas desenvolvidas pelos programadores. Estas ferramentas são anexadas ao *workbench* através de pontos de ligação chamados *extension points*. A plataforma é estruturada em várias camadas de *plug-ins*, onde cada uma define extensões sob a forma de pontos de ligação nas suas camadas inferiores. Este mecanismo de extensões permite que os programadores de *plug-ins* possam facilmente acrescentar uma variedade de funcionalidades às ferramentas básicas da plataforma. Os artefactos de cada ferramenta, como ficheiros e outros dados, são coordenados convenientemente através do modelo de recursos presentes na plataforma. O Eclipse gere internamente a complexidade dos diferentes ambientes de execução, como diferentes sistemas operativos e ambientes de trabalho colaborativos permitindo que os programadores se foquem somente no desenvolvimento das tarefas determinadas, não se preocupando com as possíveis questões de integração.

A estrutura da plataforma Eclipse é composta por diferentes subsistemas que são implementados através de um ou mais *plug-ins*. Estes subsistemas, por sua vez, encontram-se integrados por cima de um motor de execução (*runtime engine*). A Figura 3 ilustra uma vista simplificada do sistema.

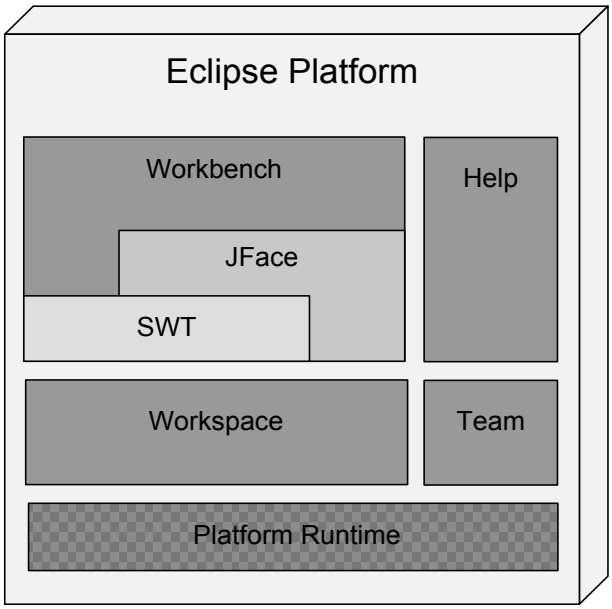

Figura 3: Arquitectura da Plataforma Eclipse

Os *plug-ins* que constituem um subsistema definem os seus diferentes pontos de extensão que permitem o aumento das funcionalidades da plataforma. Seguidamente são descritos os componentes de execução mais importantes que constituem a base da plataforma.

A *runtime platform* define os modelos dos pontos de extensão e dos *plug-ins*. Realiza a descoberta dinâmica de *plug-ins* e mantém informações sobre estes e seus pontos de extensão no seu registo. Um *plug-in* é um componente estruturado que se descreve a si próprio através dos ficheiros OSGi *manifest* (MANIFEST.MF) e *plug-in manifest* (plugin.xml). A plataforma mantém um registo dos *plug-ins* instalados e as funcionalidades que estes providenciam. A plataforma *runtime* é implementada usando um modelo de serviços OSGi onde os *plug-ins* são mais concretamente um OSGi *bundle*. Os *plug-ins* são apenas inicializados quando são necessários, de acordo com as operações realizadas pelo utilizador.

O *workspace* define APIs para criação e gestão de recursos (projectos, ficheiros e pastas) que são originados pelas ferramentas e mantidos no sistema de ficheiros. O *workbench* implementa o ambiente para a navegação na plataforma. Define pontos de extensão para a adição de componentes UI como vistas e menus. Fornece ainda um conjunto adicional de ferramentas (JFace e SWT) para a criação de interfaces. O *Standard Widget Toolkit* (SWT) é uma *framework* independente do sistema operativo de baixo nível que suporta a integração da plataforma e APIs portáteis. O *JFace UI framework* permite a construção de aplicações de alto nível suportando a gestão de diversos componentes como *widgets, wizards, actions, user preferences* e *dialogs*. Os serviços UI são estruturados para que um subconjunto de *plug-ins* UI possam ser usados para a criação de *rich client applications* (RCP) sendo independentes dos modelos *workbench* e *workspace*.

O subsistema de ajuda (*Help)* implementa uma plataforma de serviços de ajuda e uma integração facilitada para a criação e leitura de documentos. Define extensões para que os *plug-ins* possam fornecer ajudas ou documentação como livros pesquisáveis. O subsistema *Team* permite a definição e registo de implementações para uma equipa de desenvolvimento, acesso de repositórios e para a gestão de versões.

O Eclipse SDK também inclui o *plug-in Java Development Tools* (JDT) que estende a plataforma *workbench* especializando-a para a edição, visualização, compilação, *debug* e execução de código java.

Outro componente adicional incluído no Eclipse é o *Plug-in Development Enviroment* (PDE). Este fornece ferramentas que permitem a criação, desenvolvimento, teste, *debug*, compilação e implementação de *plug-ins*, fragmentos, funcionalidades, sites de actualização ou produtos RCP para o Eclipse. O PDE também proporciona uma completa ferramenta para OSGi que confere um ambiente propício ao desenvolvimento de componentes e não apenas só para o desenvolvimento de *plug-ins* Eclipse. Encontra-se subdividido em três componentes principais: o UI, a API *Tooling* e *Build*, que se encontram seguidamente descritos.

O PDE UI providencia um conjunto de ferramentas, tais como editores, *wizards, launchers, views,* entre outras, que implementam um ambiente completo para o desenvolvimento de

funcionalidades e criação de *plug-ins*, fragmentos, funcionalidades, sites de actualização, produtos RCP e OSGi *bundles* par o Eclipse. Nestas ferramentas incluem-se:

- *Form-Based Manifest Editors*: Editores multi-página que gerem centralmente todos os ficheiros manifest de um *plug-in* ou funcionalidade.
- *RCP Tools*: Assistentes e editores *form-based* que permitem a definição, teste e exportação de produtos para plataformas distintas.
- *New Project Creation Wizards*: Que permite a criação de novos *plug-ins*, fragmentos, funcionalidades e sites de actualização.
- *Import Wizards*: Para a importação de *plug-ins* ou funcionalidades do sistema de ficheiros.
- *Export Wizards*: Que implementam assistentes para a compilação, agregação e exportação de *plug-ins*, funcionalidades e produtos.
- *Launchers*: Para o teste e *debug* de aplicações Eclipse e OSGi *bundles*.
- Views: o PDE fornece vistas que auxiliam os programadores na análise de diferentes aspectos do seu ambiente de desenvolvimento.
- *Miscellaneous Tools*: Que implementam assistentes para externalização e reorganização de ficheiros *manifest*.
- *Conversion Tools*: Acrescem assistentes para a conversão de um simples projecto java o ficheiros JAR para projectos de desenvolvimento de *plug-ins*.
- *Integration With JDT*: Onde os ficheiros *manifest* do *plug-in* são integrados no editor java do Eclipse.

O PDE API *Tooling* auxilia a criação de documentação e manutenção de APIs fornecidas pelos *plug-ins* e OSGi *bundles*. Este inclui:

- *Compatibility Analysis*: Identificando aspectos binários relativos as anteriores verões de um *plug-in*.
- API *Restriction Tags*: *Javadoc tags* são fornecidas para explicitamente definir restrições associadas a tipos e membros.
- *Version Number Validation*: Identificando números de versões invalidas relativas as anteriores verões de um *plug-in*.
- *Javadoc @since Tag Validation*: Identificando omitidas ou inválidas *tags @since* em tipos e membros.
- *API Leak Analysis*: Identificando tipos e métodos API que declaram tipos API não identificáveis.

- *Quick Fixes*: Identificam correcções para a adaptação apropriada de versões e *tags* *@since*.

O PDE *Build* facilita a automatização do processo de compilação de *plug-ins*. O PDE *build* processa os *scripts Ant* baseando-se na informação dos ficheiros plugin.xml e build.properties. Os *scripts Ant* gerados podem explorar projectos associados em repositório CVS, jars compilados, *javadocs*, ou pastas *source*, agrupando todos estes componentes num formato pronto a ser enviado para uma localização remota permitindo assim para este possa ser instalado em uma qualquer instancia da plataforma.

3.4. Processo de desenvolvimento do Editor YANG

A presente secção descreve todo o processo que foi necessário para a construção do Editor YANG, desde o desenvolvimento do *parser* e funcionalidades de edição relacionadas, implementação dos assistentes para a criação de projectos e módulos YANG e dos mecanismos para a conversão destes módulos para outros formatos.

3.4.1. *Parser* YANG e Funcionalidades Associadas

O *parser* para a linguagem YANG e todas as outras funcionalidades relacionadas presentes no editor foram desenvolvidos com o auxílio da ferramenta *Xtext* [49]. O *Xtext* é uma *framework* de desenvolvimento de editores para uma determinada linguagem e permite a criação de interpretadores e compiladores totalmente integráveis no Eclipse, possibilitando um desenvolvimento muito mais simplificado mas ao mesmo tempo sofisticado das várias funcionalidades que se pretendem implementar num editor específico para uma determinada linguagem.

O *Xtext* disponibiliza aos seus utilizadores um conjunto de *domain-specific languages* (DSLs) e APIs para a descrição dos diferentes aspectos que constituem uma qualquer linguagem predefinida. Baseado nessas informações é possível a completa implementação dessa linguagem para ser executada em JVM. Os componentes que constituem o compilador são independentes do Eclipse ou OSGi e podem ser usados em qualquer ambiente Java. Nestes incluem-se o *parser, type-safe abstract syntax tree* (AST), o serializador e formatador de código, o *scoping framework* e componentes para *linking*, verificação e validação de código. Para além destes ainda é fornecido um componente para geração de código ou interpretador. Estes *runtime components* para além de totalmente integrados foram eles próprios baseados no *Eclipse Modeling Framework* (EMF), que eficientemente permitem que o *Xtext* possa ser usado conjuntamente com outras plataformas EMF, como por exemplo o *Graphical Modeling Project* (GMF)[50].

Esta arquitectura, ilustrada na Figura 4, implementa também um IDE completo para o Eclipse, totalmente configurado para a linguagem pretendida. São implementada uma grande

variedade de funcionalidades por defeito e com as DSLs e APIs fornecidas é possível a sua fácil configuração ou alteração dos parâmetros que as constituem. Caso isso não seja suficiente é ainda possível a injecção de código, através do *Guice Injection*, para uma maior flexibilidade na alteração do comportamento por defeito de uma implementação desenvolvida em *Xtext*.

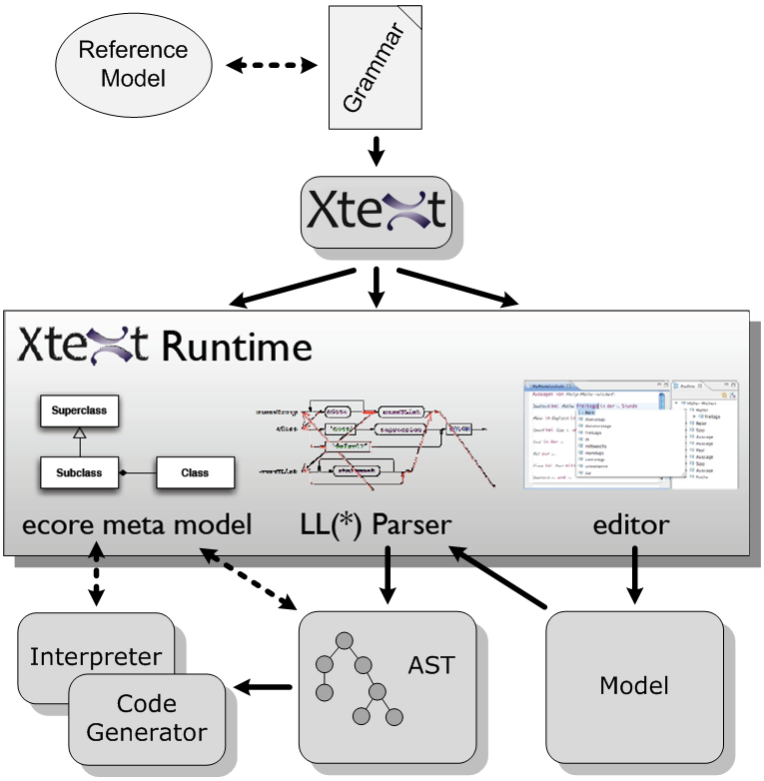

Figura 4: Arquitectura do Xtext

Quando um projecto *Xtext*, que na sua essência representa um projecto em formato de *plug-in* para o Eclipse, é criado, é subdividido em três subprojectos cada um focando uma temática própria, como ilustrado na Figura 5. O primeiro, com o nome do projecto, é vocacionado para a definição da gramática e todos os seus aspectos de execução, como o *linking, scoping* e validação. Os aspectos relacionados com o IDE como o editor, qualquer vista ou construtor incremental do projecto encontram-se no projecto *<nome_do projecto>.ui*. Estes dois componentes incluem classes derivadas da própria gramática conjuntamente com classes para expandir ou adaptar o seu comportamento. Embora o *Xtext* faça uso de geração de código, a maioria do código é escrito como bibliotecas, que são referenciadas por meio de OSGi usando o *Manifest.MF*. O terceiro projecto, *<nome_do projecto>.generator*, tem como funcionalidade conter um gerador de código *Xpand* que utiliza o modelo criado com o editor da DSL. Este recurso é apenas opcional sendo possível o uso de qualquer linguagem de programação que possa ser executada em JVM para implementar um gerador de código para um modelo definido em *Xtext*, como também é opcional a sua não utilização e usar os modelos criados dinamicamente somente na *runtime.*

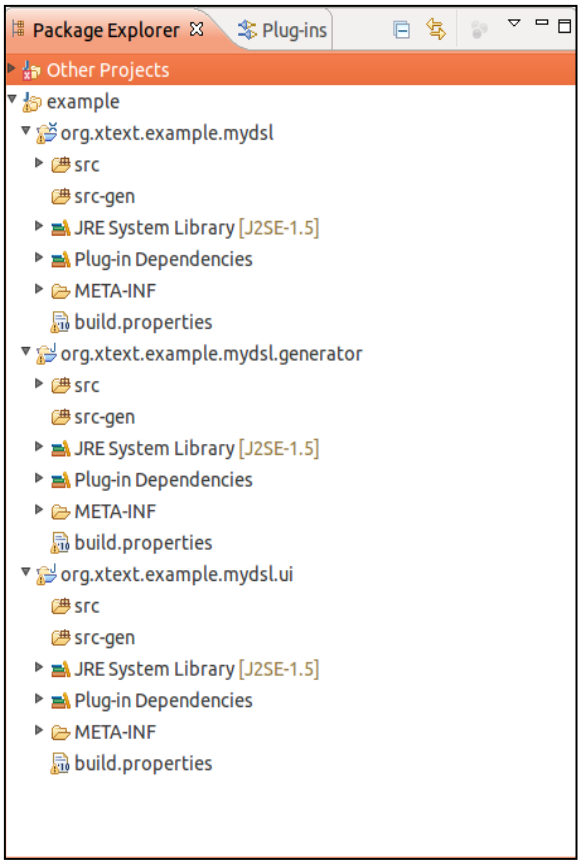

Figura 5: Estrutura de um projecto Xtext

Dentro de cada projecto podem ser encontradas as pastas *src/* e *src-gen/* para a separação do código que é especificado pelo utilizador, daquele que é gerado pela plataforma.

No projecto é fornecido um ficheiro para a definição da gramática. Este ficheiro não tem só a funcionalidade para a descrição concreta da sintaxe da linguagem como também define certas informações de como o *parser* deve estruturar o modelo durante o *parsing*. A gramática não é mais que um conjunto de regras definidas através de um conjunto de expressões com uma sintaxe bastante similar ao formato *Extended Backus-Naur Form.*

A Figura 6: Excerto inicial da gramática YANG implementada no Xtext mostra o excerto inicial da gramática implementada com o auxílio da ferramenta *Xtext*.

```
grammar org.xtext.editor.yang.Yang hidden(WS, ML_COMMENT, SL_COMMENT)

generate yang "http://www.xtext.org/editor/yang/Yang"

import "http://www.eclipse.org/emf/2002/Ecore" as ecore
```

Figura 6: Excerto inicial da gramática YANG implementada no Xtext

A primeira linha da gramática define o seu nome. Este nome deve ser único e tal como as classes públicas de java devem reflectir a localização do ficheiro no seu java *classpath.*

Também é possível definir um conjunto de símbolos terminais (só faz sentido que sejam incluídos terminais sem qualquer significado semântico) para que sejam escondidos das regras do *parser* sendo ignorados quando é realizada a sua identificação durante o *parsing*.

Na segunda linha é declarado o meta modelo a ser derivado. O *parser* do *Xtext* cria em memoria um grafo dos objectos enquanto analisa gramaticalmente o texto. Estes objectos são instâncias do modelo EMF *Ecore* que basicamente consiste num *Epackage* contendo *Eclasses*, *EDataTypes* e *EEnums*. O *Xtext* pode inferir modelos *Ecore* da gramática mas também reutilizar modelos já existentes. Um exemplo desta situação é dado pela terceira linha de código. Este permite a utilização de tipos já definidos nesse *ECore* como *EString*, *EInt*, *Ebool*, *etc*, que são essenciais para a definição dos símbolos terminais. A partir daqui é possível começar a especificar a gramática em si através da definição de *parser* ou *terminal rules*. A especificação destas regras segue a estrutura em pseudocodigo ilustrada na Figura 7.

```
(terminal)? <Nome_da_Regra>: <Especificação_da_Regra>;
```

Figura 7: Pseudocodigo para a definição de regras em Xtext

A especificação de uma regra pode compreender outras regras ou terminais, podendo ser especificada as suas respectivas cardinalidades, palavras reservadas, atribuições ou alternativas. A gramática *Xtext* não só define as regras para serem implementadas no *parser* mas também a estrutura resultante da *abstract syntax tree*, onde cada regra irá corresponder a um novo objecto nessa árvore.

Antes de ser efectuado o *parsing* é realizado o *lexing*, que transforma uma sequência de caracteres em uma sequência de *tokens*. Neste contexto, um *token* é efectivamente um conjunto de caracteres com um significado colectivo. Consiste na conjunção de um ou mais caracteres que correspondem a uma *terminal rule* ou *keyword*, representando assim o seu símbolo atómico. Uma *terminal rule* apenas retorna um único *token*, normalmente num tipo simples de dados, como *String, Integer* ou *Id*. O conjunto de todas as *terminal rules* definidas para a gramática YANG encontram-se descritas na Figura 8.

```
terminal STRING:
    '"' -> '"' | "'" -> "'" ;
terminal ID:
    (('a'..'z'|'A'..'Z'|'_')('a'..'z'|'A'..'Z'|'_'|'-'|'.'|'0'..'9')*);
terminal ML_COMMENT:
     '/*' -> '*/';
terminal SL_COMMENT:
     '//' !('\n'|'\r')* ('\r'? '\n')?;
terminal WS:
     (' '|'\t'|'\r'|'\n')+;
```

Figura 8: Terminal Rules implementadas na gramática YANG

As *token rules* (*terminal* e *parser rules*) são descritas usando expressões um formato similar ao *Extended Backus-Naur*. É possível especificar das diversas cardinalidades designadas a um dado elemento através dos operadores, '*' para zero ou mais, '+' para um ou mais e '?' para zero ou um, a ausência de operador atribuí carnalidade igual a um. Para além das carnalidades as expressões podem ser definidas através do uso de *keywords* (''*<conjunto_de_caracteres>*'') e ranges (*<valor inicial>*'..'*<valor final>*), podendo também ser incluídos nestas definições operadores para negação ('!'), *until* ('->'), *wildcards* ('.') e alternativas ('|').

Ao contrário das *terminal rules*, as *parser rules* não produzem apenas um valor, mas uma árvore composta por *tokens* terminais e não-terminais. Estas permitem a construção da *parse tree*, ou seja, podem ser vistas como um projecto a ser seguido na criação dos *EObjects* que representam o modelo semântico (e.g. AST). Para uma melhor compreensão destes conceitos o código da Figura 9 ilustra um pequeno excerto das primeiras *parser rules* especificadas na gramática YANG.

```
YangFile:
    Module | SubModule;
Module:
    'module' name=STRINGARG
    '{' (statements+=ModuleStatement)* '}';
```

Figura 9: Parser Rules implementadas na gramática YANG

Nas *parser rules* é possível o uso de atribuições. Estas atribuições são usadas para designar um dado atributo ao objecto actual, ou seja, à sua *EClass*. Existem três tipos de operadores para a definição das atribuições. O operador '=' designa uma atribuição simples que é usada em atributos de um só elemento. O operador '+=' especifica um atributo em forma de lista onde múltiplos valores são continuamente adicionados. O ultimo operador, '?=', requer que o atributo seja do tipo *EBoolean* atribuindo o valor *true* caso seja consumido e *false* em caso contrário.

Uma funcionalidade particular do *Xtext* é a habilidade de poder declarar *cross references* na própria gramática. Num compilador tradicional as referências não são estabelecidas durante o processo de *parsing*, mas mais tarde na fase de *linking*. O mesmo acontece num editor criado em *Xtext,* a diferença reside no facto de o *linker* analisar previamente as informações contidas na gramática para a implementar as *cross references* pretendidas.

As *cross references* são especificadas por uma *parser rule* contida dentro de parênteses rectos, como na definição da regra *TypeStatement*, que é ilustrado no excerto de código da Figura 10.

```
TypedefStatement:
    'typedef' name=STRINGARG
    '{' (typedefsubstatemnet+=TypedefSubstatement)* '}';
TypedefSubstatement:
    (TypeStatement | DescriptionStatement | ReferenceStatement |DefaultStatement
    | StatusStatement | UnitsStatement | UnknownStatement);
TypeStatement:
    'type' (type=BuiltInType | (pre=STRINGARG':')? importtype=[TypedefStatement])
    (';' | '{' (typesubstatements+=TypeSubStatement)* '}');
TypeSubStatement:
    (BitStatement | FractionDigitsStatement | DefaultStatement |BaseStatement
   | EnumStatement | LengthStatement | PathStatement |PatternStatement
   | RangeStatement | RequireInstanceStatement |TypeStatement | UnknownStatement);
```

Figura 10: Definição das instruções Type e Typedef na gramática YANG

O código descrito acciona no editor um mecanismo de procura do nome definido numa regra *TypedefStatement*, quer no próprio ficheiro quer em ficheiros importados presentes no mesmo *workspace*. Na verdade a referência não aponta para a regra *TypedefStatement* mas sim à sua *EClass*. O *parser* analisa apenas o nome do elemento referenciado, identificado pelo atributo '*name*', e guarda-o internamente. Mais tarde o *linker* efectua a referenciação baseando-se no nome, na definição do tipo referenciado e nas regras definidas para o *scoping*. Por defeito é aplicado um *scoping* baseado em *namespaces*, o que significa que todas as entidades presentes no mesmo *namespace* são possíveis candidatas a poderem serem referenciadas.

Com a gramática totalmente definida é necessário a execução do gerador para derivar os vários componentes da linguagem para a implementação do editor. O ficheiro que realiza essa tarefa encontra-se em formato *Modeling Workflow Engine 2* (MWE2). O MWE2 é um mecanismo para a geração de código altamente configurável onde os utilizadores podem descrever composições de objectos arbitrários numa sintaxe simples e concisa, permitindo a declaração de instâncias de objectos, atributos e referências. A sua principal utilização é na definição de *workflows* que, usualmente, designam vários de componentes que interagem entre si. Existem componentes para a leitura de recursos EMF, para realização de operações ou transformações nestes e em reescreve-los ou gerar outros artefactos de informação. A execução do MWE2 permite a geração do *parser* e *serializer* assim como outras infraestruturas adicionais do código.

Resumindo, o modelo em *Xtext* é traduzido sob a forma de um recurso EMF. O modelo principal é representado por instâncias denominadas *EClass*, onde elas próprias são declaradas como modelos *ECore*. A *parse tree* também é criada, actuando como uma ponte de ligação entre o modelo e o texto, ou seja, entre a AST e a gramática. O diagrama da Figura 11 ilustra os quatro diferentes tipos de modelos derivados para uma gramática genérica.

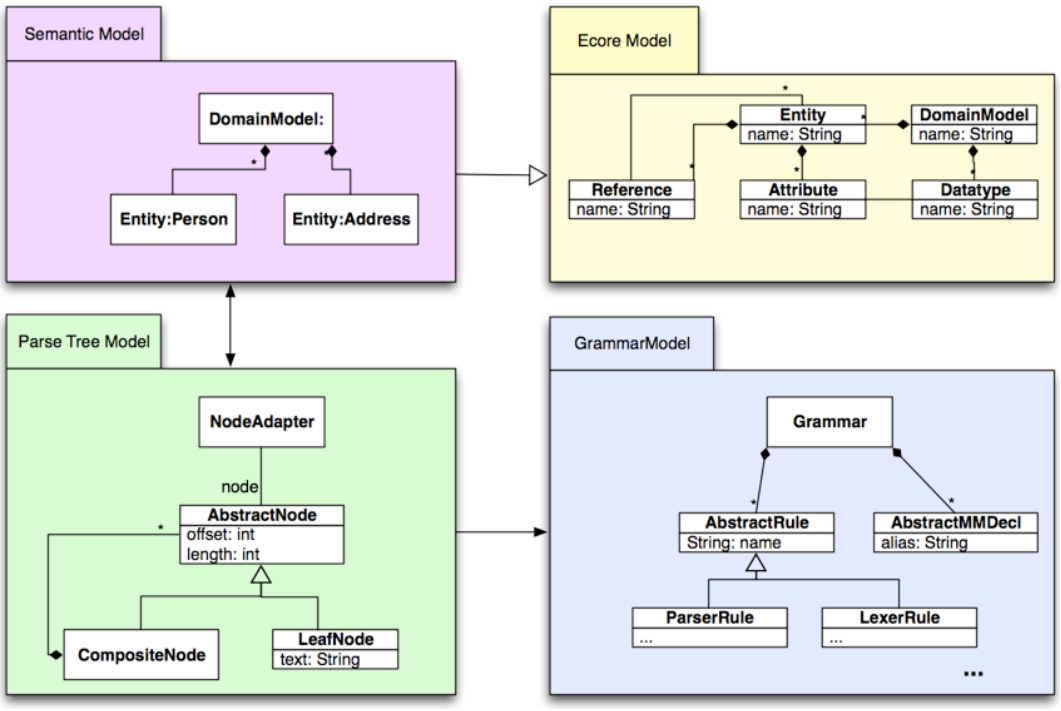

Figura 11: Modelos de dados implementados pelo Xtext

A partir da gramática, o *Xtext* também gera vários componentes para o processamento e validação dos dados introduzidos no editor. Alguns destes elementos são derivados automaticamente da gramática especificada, sendo possível expandir posteriormente essas funcionalidades. Para isso o *Xtext* fornece um API para a escrita de restrições inerentes da própria linguagem que não possam ser inferidos directamente da descrição da gramática. Este API denomina-se *AbstractDeclarativeValidator* e permite, tal como o nome indica, a implementação de restrições ou condições de uma forma declarativa. Em vez de o programador ser obrigado a escrever uma estrutura extensiva de *if-elses* ou *cases*, ou de estender o modelo EMF gerado, este pode simplesmente declarar métodos e adicionar-lhe a notação *@Check* para que sejam invocados automaticamente quando é executada a validação. Estes métodos são invocados consoante os seus parâmetros de entrada o que permite repartir as diferentes condições a implementar por diferentes métodos ou apenas focar estas condições em elementos mais específicos que compõem a linguagem. Caso a condição de erro se verifique esta pode ser facilmente apresentada no editor com o uso dos métodos *error* e *warning*. Estes métodos permitem definir a respectiva mensagem de erro e realçar o componente do elemento que o originou. Na Figura 12 é ilustrado um dos métodos de validação implementados para o editor YANG.

```
@Check
public void checkRevision(RevisionStatement revision){
    String date=revision.getDate();
    if(!date.matches("(\")?[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9](\")?"))
        error("The date must be in form '4DIGIT-2DIGIT-2DIGIT' ",
        YangPackage.REVISION_STATEMENT__DATE);
}
```

Figura 12: Verificação do argumento da instrução YANG Revision

O método anterior verifica se o parâmetro da instrução YANG *Revision* se apresenta no formato de data predefinido pela linguagem. Caso esta condição falhe o editor automaticamente destaca a linha do respectivo erro e assinala o elemento que representa a data na instrução, sendo complementado com a respectiva mensagem de erro. Também é possível a implementação de *quick fixes*, no respectivo API, para promover, ao utilizador, várias alternativas de resolução do erro ou aviso detectado.

O *Xtext* implementa também a funcionalidade de *linking* permitindo a especificação de *cross-references* na própria definição da gramática. O *Xtext* usa um mecanismo de *lazy linking* onde a classe *LazyLinker* cria inicialmente EMF *proxys* e lhes atribui a sua correspondente *EReference*. Um *proxy* EMF é descrito por um URI que aponta para o seu *EObject* real. No caso do *lazy linking* o URI guardado compreende a informação contextual fornecida pelo *parser*, mais concretamente, o *EObject* contido na *cross-reference*, o actual índice da *EReference* e a string que representa o *crosslink* no concreto sintaxe. A classe *IScopeProvider* é responsável pela definição de um *IScope* para um determinado contexto referente a um *EObject* e à sua *EReference*. O *IScope* devolvido pela classe conterá todos os possíveis candidatos de um dado objecto e a sua *cross-reference*. Um *IScope* representa um elemento ligado a uma lista de *scopes*. Isto significa que o *scope* pode ser contido dentro de outro *scope*, mais abrangente. Cada *scope* funciona de forma semelhante a uma tabela ou mapa onde as chaves são strings e os seus valores são denominados por *IEObjectDescription*, que efectivamente representam uma descrição abstracta do *EObject* real. Se um objecto contém na sua definição um atributo com o nome '*name*', um *IEObjectDescription* é criado e exportado, ou seja, é adicionado à lista. Caso um *EObject* não tenha definido nenhum atributo '*name*' este é considerado não referenciável, fora do seu recurso e consequentemente não é indexável.

Apesar de este mecanismo de *linking* ser bastante robusto e de fácil implementação, para certas especificidades mais complexas da linguagem a definição de *cross-references* na gramática pode não ser suficiente. Analisando, por exemplo, o caso da instrução YANG *Augment,* que recebe como argumento um *absolute-nodeid*, que representa um caminho Xpath para um elemento num módulo, será necessário realizar a verificação dos vários componentes que o constituem e não apenas da referência para o nó apontado. Esta análise foi também implementada no API de validação, de forma semelhante ao método de verificação da instrução *Revision* descrita anteriormente. Para a validação de um argumento de uma instrução *Augment* foi necessário a injecção do *IScopeProvider*. Todos os componentes em *Xtext* são construídos por um meio de *Dependency Injection*. Isto significa basicamente que qualquer porção de código que seja necessária para promover uma certa funcionalidade a um outro componente pode ser declarada independentemente, obtendo assim todas as suas características e funcionalidades. O *IScopeProvider* permite a obtenção de um *scope* específico, através do método *getScope(EObject context, EReference reference)*, onde o *context* define o elemento a partir do qual deverá ser obtida a referência e reference que identifica a referência a ser usada para filtrar os elementos. Com o *IScope* obtido é possível a realização de uma procura de um elemento por um determinado parâmetro e obter o seu *IEObjectDescription*. A partir deste é possível obter todo o conteúdo pertencente ao elemento, o seu *inner-scope*, podendo verificar posteriormente cada um dos restantes constituintes do

absolute-nodeid analisando recursivamente a sua AST respectiva. Um excerto da implementação acima descrita encontra-se ilustrado pelo código da Figura 13.

```
@Inject
IScopeProvider provider;

@Check
public void checkAugmentStatement(AugmentStatement aug) {
      String ident = aug.getArg();
      String[] idents = ident.split("/");
      IScope scope = provider.getScope(aug,aug.eContainmentFeature());
      IEObjectDescription con = scope.getContentByName(idents[0]);
      EList<EObject> list = con.getEObjectOrProxy().eContents();
     checkAugmentAbsolutNodeID(list, idents);
}
```

Figura 13: Implementação do ScopeProvider para a referenciação de argumentos complexos

O *Xtext* também deriva automaticamente da gramática diversos componentes relacionados com os aspectos visuais e de aperfeiçoamento da interacção do utilizador com o editor. Entre eles destacam-se o *content assist, outline view, quick outline, hyperlinking* e *syntax highlighting*. Todas estas funcionalidades podem ser personalizadas e estendidas através dos seus APIs. Por exemplo, novas propostas podem ser adicionadas ao *content assist* através do seu registo na classe *ProposalProvider*. A *outline view* auxilia o utilizador a navegar no seu modelo de dados. Por defeito este providencia uma vista hierárquica do modelo e permite que os nós pertencentes à árvore possam ser ordenados alfabeticamente. A selecção de um elemento no *outline* irá destacar o correspondente elemento definido no editor. O *outline* permite que a sua estrutura possa ser alterada com o auxílio da classe *ISemanticModelTransformer*, sendo possível a omissão de um determinado tipo de nó ou a inclusão adicional de nós, podendo estes serem apenas virtuais. Também é possível a implementação de mecanismos de filtragem através da definição e registo dessas capacidades na classe *DeclarativeActionBarContributor*. *O quick outline* rege-se pelos mesmos aspectos do *outline* suportando ainda um mecanismo de procura em profundidade sendo passível a utilização de *wildcards*. O *Xtext* fornece o suporte para *hyperlinking* para os *tokens* correspondentes as *cross-references* na definição da gramática. Quando é activado o editor irá navegar para o respectivo elemento do módulo referenciado, mesmo que este se encontre em um outro recurso, e seleccionar a região de texto correspondente. O *Xtext* providencia também um *ILabelProvider* para a personalização da aparência de elementos a serem apresentados ao utilizador. Este pode influenciar a representação dos nós em distintos componentes do IDE como o *outline view, hyperlinks, content proposals, find dialogs, etc.*

A Figura 14 apresenta algumas das funcionalidades implementadas no editor YANG.

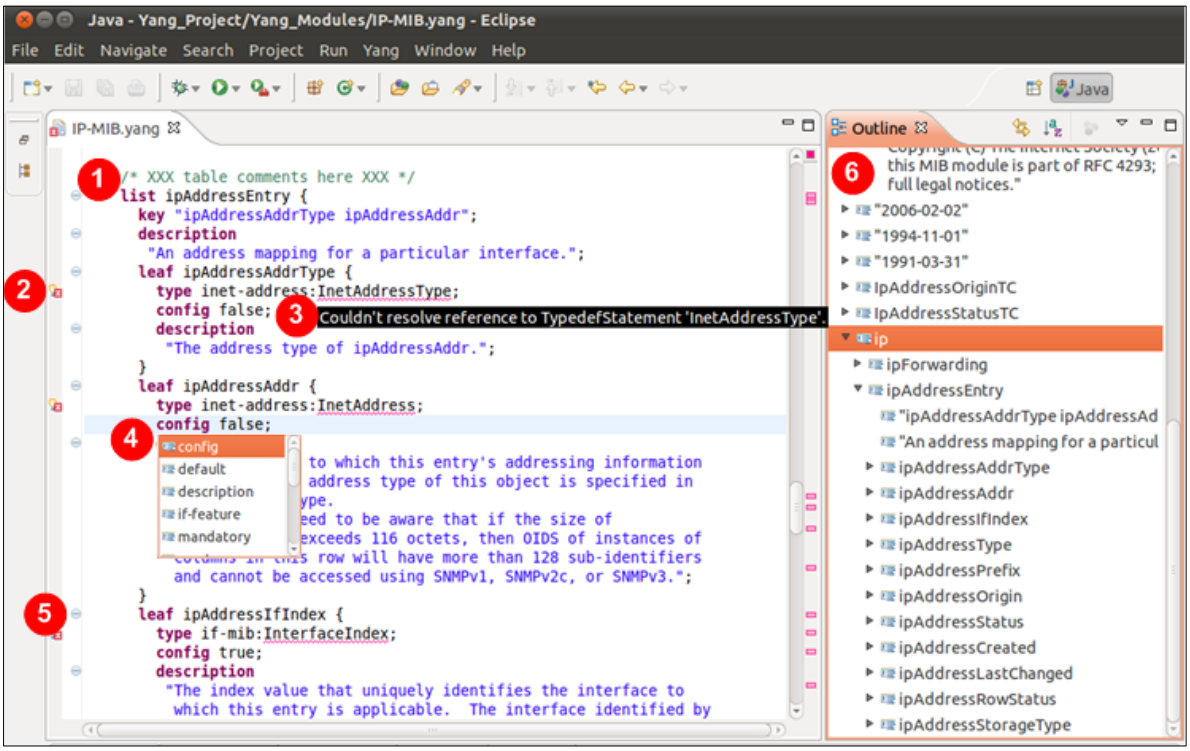

Figura 14: Funcionalidades implementadas no Editor YANG

Legenda:

1. *Parser* do código e realce da sintaxe;
2. Detecção de erros;
3. Mensagem de erro;
4. Assistente de conteúdo;
5. *Folding* do código;
6. *Outline* do código;

3.4.2. Assistentes YANG

Ao editor foram adicionalmente desenvolvidos assistentes para a criação de projectos e de ficheiros YANG. Estes assistentes foram implementados com o auxílio da *framework JFace* UI. Esta providencia várias ferramentas e métodos para a criação de assistentes e diálogos.

Os assistentes (*wizards*) têm a funcionalidade de auxiliar um utilizador na realização de um conjunto sequencial de tarefas. Os *wizards*, no contexto do Eclipse, seguem a arquitectura ilustrada na Figura 15.

A *Wizard Dialog* é a API de diálogo de alto nível contida num *Wizard*. Esta define botões standards e gere o conjunto de páginas a ser facultadas quando esta é activada. Quando é desenvolvida uma extensão *wizard*, o *workbench* providencia automaticamente uma *Wizard*

Dialog para onde o *wizard* é enviado não sendo portanto necessária a sua criação. A *Wizard Dialog* realiza a activação ou desactivação dos botões *Next, Back* e *Finish* dependendo da informação contida no *Wizard* e da correspondente *Wizard Page*.

O *Wizard*, derivado da classe *IWizard*, controla a aparência e comportamento global de um assistente, como por exemplo, o texto presente na barra de título, imagens, botões de ajuda, etc. O *Wizard* pode colectar informações de um *DialogSettings* para definir valores por defeito das opções a serem apresentadas nas *Wizard pages*. A classe *Wizard* implementa muitos dos detalhes necessários na execução de um *wizard*. Tipicamente, apenas é necessário a extensão desta classe para o desenvolvimento das operações específicas do *Wizard* a implementar. Nestas, normalmente incluem-se a criação e adição das páginas do *Wizard* e implementação do procedimento a ocorrer quando é executado o *Finish*.

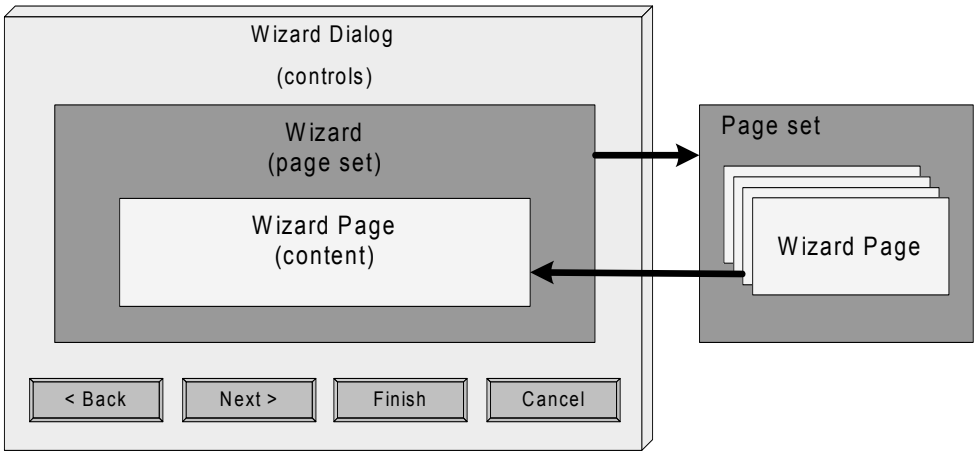

Figura 15: Arquitectura dos Wizards em Eclipse

O *Wizard Page*, derivado da classe *IWizardPage*, define os controlos que podem ser apresentados nas *Wizard Pages*. Este responde aos eventos desencadeados pelo utilizador quando interage com os diversos elementos que o constituem e determina se a pagina se encontra num estado completo. Tipicamente, um *Wizard Page* estende a classe *Wizard Page* e tem como finalidade de definição de controlos SWT que representam a pagina, e determinar se o utilizador fornece a informação necessária para que possa ser possível a transição para outra página.

Mais concretamente o assistente para a criação de projectos YANG foi implementado pela classe *YangWizard* que estende a classe *Wizard* e implementa a *INewWizard* classe. Esta permite a criação de um projecto YANG com a estrutura ilustrada na Figura 16.

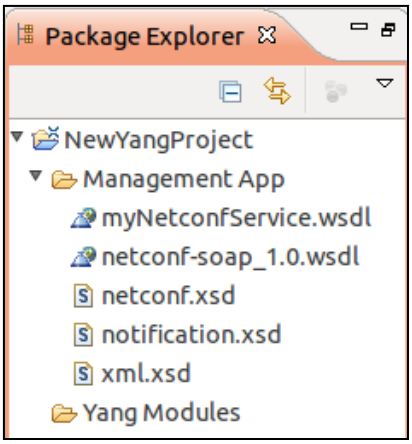

Figura 16: Estrutura de um novo projecto YANG

A pasta *Management_App* tem o intuito de albergar todos os ficheiros necessários para a geração da aplicação de gestão distribuída. São criados, pelo assistente, os ficheiros *myNetconfService.wsdl, netconf-soap_1.0.wsdl, netconf.xsd, notification.xsd* e *xml.xsd*. Estes ficheiros são essenciais para a criação do serviço NETCONF e por esse motivo foram incluídos na *bundle org.xtext.editor.yang.ui* sendo automaticamente replicados sempre que é criado um novo projecto YANG. A pasta *Yang_Modules* tem o propósito de conter os módulos YANG a virem a ser criados e desenvolvidos pelo utilizador. Outra função realizada pelo assistente de criação de projectos YANG é a definição da natureza do projecto, para que, sempre que um novo projecto YANG é criado, seja automaticamente associada ao projecto a natureza relativa ao Editor YANG, que é activado sempre que seja aberto um ficheiro com a extensão '*.yang*' incluído no projecto. A classe *YangWizard* instancia uma pagina *WizardYangNewProjectPage*, que estende a classe *WizardNewProjectCreationPage*. Esta página permite especificar o nome do projecto e a respectiva localização do mesmo no sistema de ficheiro com o aspecto ilustrado na Figura 17 apresentada seguidamente.

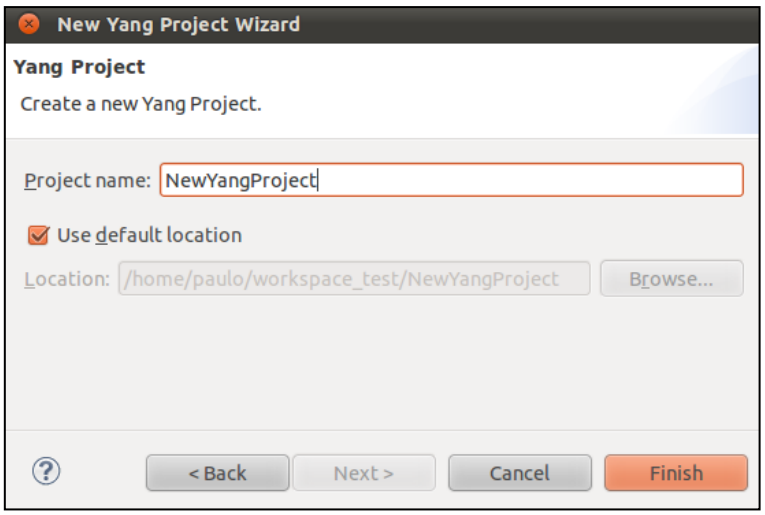

Figura 17: Assistente de criação de um projecto YANG

A página também inclui um método para verificar se o nome do projecto é especificado sem espaços. O utilizador tem de obedecer a esta restrição para poder completar o *Wizard* e

assim criar um novo projecto. O código desta verificação encontra-se descrito seguidamente na Figura 18.

```
@Override
protected boolean validatePage() {
    if(getProjectName().contains(" ")){
        setErrorMessage("The Project Name must not contain 'spaces'.");
        return false;
    }
    return super.validatePage();
}
```

Figura 18: Verificação do nome no assistente de um novo projecto YANG

De forma semelhante ao *Wizard* para criação de projectos YANG foi desenvolvido um assistente para a criação de ficheiros YANG, ilustrado na Figura 19. Este é implementado através da classe *YangFile* que estende a classe *Wizard* e implementa a classe *INewWizard*. A classe *YangFile* instancia uma página *WizardYangNewFileCreationPage* que estende a classe *WizardNewFileCreationPage*. A página permite a definição do local no *workspace* pretendido para a sua criação e o respectivo nome do ficheiro. Quando um novo ficheiro YANG é criado este é automaticamente aberto e exibido no editor.

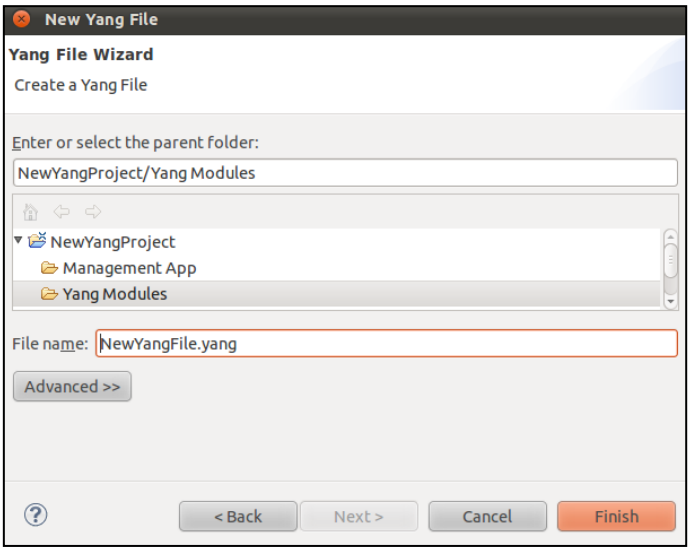

Figura 19: Assistente de criação de um novo ficheiro YANG

Também de forma semelhante aos *wizards* de criação de projectos e ficheiros YANG, foi desenvolvido um *wizard* para a importação de um documento MIB para o formato YANG. A sua implementação foi desenvolvida na classe *ImportMIBWizard* que estende a classe *Wizard* e implementa a interface *IImportWizard*. A classe *ImportMIBWizard* instancia uma página da classe *ImportMIBWizardPage* que, por sua vez, estende a classe *WizardNewFileCreationPage*. O assistente exibe uma página, apresentado na Figura 20, onde é possível procurar e seleccionar uma MIB presente no sistema de ficheiros, e definir o nome e o local no *workspace* onde o módulo YANG traduzido deve ser criado.

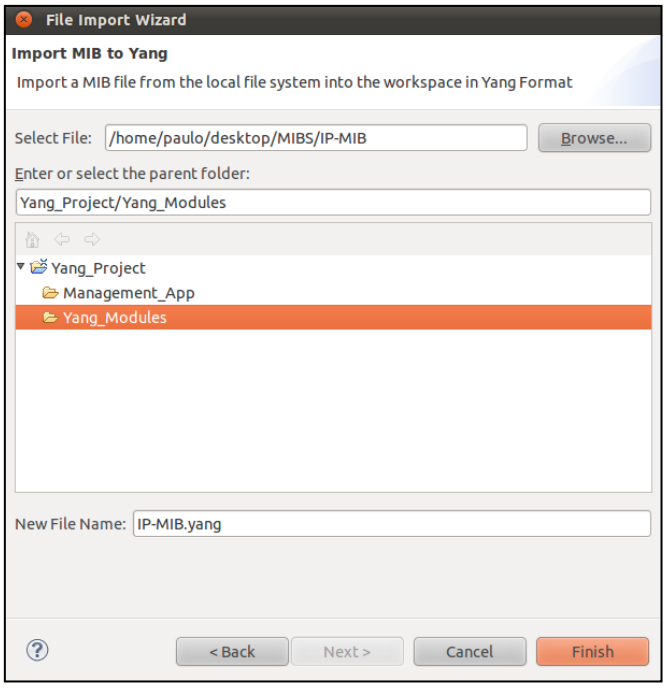

Figura 20: Assistente para importação de uma MIB para YANG

O processamento necessário para obter um módulo YANG traduzido de uma MIB é realizado pela aplicação Smidump. O método *performFinish()* da classe *ImportMIBWizard* tem a particularidade de executar uma *system call* à aplicação Smidump instalada no sistema através do método *Runtime.getRuntime().exec(String[] command)*. Caso haja a ocorrência de um erro na execução desta aplicação é obtido o *error stream* para ser posteriormente exibido na própria consola existente no Eclipse, promovendo um feedback ao utilizador da sua origem e auxiliando-o na sua resolução. Tal como na criação de um novo ficheiro YANG, o módulo importado para YANG, após ser criado é automaticamente aberto e exibido no editor. Seguidamente é ilustrado, na Figura 21, o código do método *performFinish()* implementado no assistente de importação.

```
public boolean performFinish() {
    //create a new file
    IFile file = mainPage.createNewFile();
    if (file == null)
        return false;
    String pathfile = file.getLocation().toString();
    //run smidump to convert MIB to Yang
    //smidump -f yang --yang-indent=3 -k -o [outputfile] [inputfile]
    String[] commands = null;
    commands = new String[]{"smidump","--format=yang", "--yang-indent=3", "--level=1",
                            "--keep-going", "--output="+pathfile, pathfile};
    try {
        Process Findspace = Runtime.getRuntime().exec(commands);
        BufferedReader Resultset = new
            BufferedReader(new InputStreamReader (Findspace.getErrorStream()));
        String line;
        //If ocurrs an error print in the console
        while ((line = Resultset.readLine()) != null) {
            System.out.println(line);
        }
    } catch (IOException eio) {
        eio.printStackTrace();
    }
    //open the new file in the editor
    try {
        IDE.openEditor(_workbench.getActiveWorkbenchWindow().getActivePage(), file);
    } catch (PartInitException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}
```

Figura 21: System Call à aplicação Smidump

Como o assistente depende da aplicação Smidump, foi implementada na classe *ImportMIBWizard*, código para a verificação da sua instalação no sistema. Esta verificação é realizada no método *canFinish(),* exibido no excerto de código ilustrado na Figura 22, onde é verificado qual o sistema operativo em funcionamento e verificado se a aplicação se encontra efectivamente instalada através do comando '*Which'*. Caso qualquer uma das condições falhe é exibido na própria página do assistente a respectiva mensagem de erro informando o utilizador da sua ocorrência, sendo consequentemente desactivado o botão de *Finish* não permitindo a finalização do assistente.

```
	public boolean canFinish() {
		//Verify if the operating system is Linux or Mac
		 if (!((System.getProperty("os.name").equals("Linux")) ||
		System.getProperty("os.name").equals("Mac OS X"))) {
				mainPage.setErrorMessage("Can't be done on
"+System.getProperty("os.name")+"!!!");
				return false;
		}
		//Verify if the Smidump is installed in the system
		String command = "which smidump";
		try {
				Process Findspace = Runtime.getRuntime().exec(command);
				BufferedReader Resultset = new
					BufferedReader(new InputStreamReader (Findspace.getInputStream()));
				String line = "";
				while ((Resultset.readLine()) != null) {
					line = line + (Resultset.readLine());
				}
				if (line.isEmpty()) {
					mainPage.setErrorMessage("This needs Smidump to be correctly instaled in
									our system!!!");

					return false;
				}
		} catch (IOException eio) {
				eio.printStackTrace();
		}
		return true;
	}
```

Figura 22: Verificação da presença da aplicação Smidump no sistema

3.4.3. MENU YANG

Para além destes três assistentes, o editor YANG foi adicionalmente complementado com um novo menu para a realização das tarefas de conversão de módulos YANG para YIN e de conversão de módulos YANG para XSD, implementados através do uso de comandos. Um comando em Eclipse é a descrição declarativa de um componente que é independente dos detalhes de implementação. Os comandos são definidos através do ponto de extensão *org.eclipse.ui.commands*, podendo este ser categorizado, definido onde e como deve ser incluído na interface de utilizador e ser designado um conjunto de teclas de atalho para a sua invocação. A Figura 23 ilustra o menu implementado presente na interface de utilizador do editor.

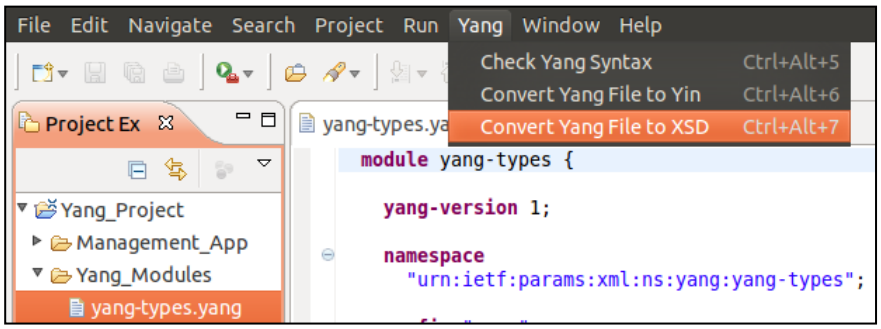

Figura 23: Menu YANG

O comportamento de um comando é definido através de *handlers*. Um *handler* é uma classe que é executada quando o comando é invocado e deve implementar a classe *org.eclipse.core.commands.IHandler*. A classe *org.eclipse.core.commands.AbstractHandler* providencia a implementação por defeito de um interface *IHandler*. O *IHandler* define quatro métodos necessários para a completa implementação de um comando. São eles: *isEnabled*, invocado na instanciação do comando, definindo se este deve estar activo, *isHandled*, que define se um *handler* pode ser invocado, execute, que define o código a ser executado quando é realizada a acção, e o *fireHandlerChanded*, que define o código a ser invocado se o método *isEnabled* é alterado.

O processamento necessário para a realização da tradução de YANG para YIN e de YANG para XSD é efectuado pela aplicação Pyang e, tal como implementado para o procedimento de importação de um documento MIB para YANG, é realizada uma *system call* mas agora à aplicação Pyang, que deve se encontrar a priori instalada no sistema, sendo executada através do método *Runtime.getRuntime().exec(String[] command)*. Esta acção só será realizada se o Pyang estiver correctamente instalado no sistema sendo verificada a sua existência através do comando *'which pyang'*. O código da Figura 24 ilustra um excerto da implementação do método execute para o comando que realiza a tradução de YANG para XSD, onde é especificada a *system call* que, caso haja a ocorrência de erros estes são recolhidos e apresentados na consola do Eclipse.

```
//run pyang to convert yang to xsd
//"pyang --format=xsd --output="+namefileout+" "+pathfilein;
try {
    Process Findspace = Runtime.getRuntime().exec(new
            String[] { "pyang", "-f", "xsd", "-o", namefileout, pathfilein });
    BufferedReader Resultset = new
                    BufferedReader(new InputStreamReader (Findspace.getErrorStream()));
    String line;
    while ((line = Resultset.readLine()) != null) {
            System.out.println(line);
    }
} catch (IOException eio) {
    eio.printStackTrace();
}
```

Figura 24: System Call à aplicação Pyang

Se as operações de tradução de YANG para YIN ou YANG para XSD forem correctamente realizadas os novos documentos YIN ou XSD serão automaticamente abertos e exibidos no editor. Estes novos ficheiros são criados na mesma pasta e com o mesmo nome do módulo YANG de que é derivado, sendo este completado com a extensão correspondente. Se for detectado um ficheiro com o mesmo nome nessa localização é apresentada uma janela de diálogo, antes de ser realizada a tradução, informando o utilizador da ocorrência dessa situação e perguntando-lhe se pretende substituir esse ficheiro. O utilizador pode optar por três opções via três botões: *Cancel*, que aborta a operação, *Yes*, que autoriza a substituição do ficheiro e *No*, que apresenta uma nova janela que permite ao utilizador a definir um novo nome ou localização para o ficheiro YIN ou XSD que se pretende gerar. O código para a

especificação deste diálogo e das suas correspondentes alternativas encontra-se descrito na Figura 25.

```
//Verify is the new file exists
String namefileout = pathfileout+namefile+".xsd";
File fileToOpen = new File(namefileout);
FileDialog dlg = new FileDialog(editor.getSite().getShell(), SWT.SAVE);
if (fileToOpen.exists()) {
// The file already exists; asks for confirmation
MessageBox mb = new MessageBox(dlg.getParent(), SWT.ICON_WARNING
                        | SWT.YES | SWT.NO | SWT.CANCEL);
mb.setMessage(namefile + ".xsd already exists. Do you want to replace it?");
int result = mb.open();
//If choose CANCEL do nothing
if(result == SWT.CANCEL)
    return null;
//If choose NO display SAVEAS dialog
if(result == SWT.NO){
    Shell shell = dlg.getParent();
    FileDialog dialog = new FileDialog(shell, SWT.SAVE);
    dialog.setText("Input the new file name");
    dialog.setFilterNames(new String[] { "Xsd Files", "All Files (*.*)" });
    dialog.setFilterExtensions(new String[] { "*.xsd", "*.*" });
    dialog.setFilterPath(pathfileout);
    dialog.setFileName(namefile+"(new).xsd");
    dialog.open();
    namefileout = dialog.getFilterPath() + "/" + dialog.getFileName();
}
```

Figura 25: Diálogo para a alteração do nome quando é criado um ficheiro XSD

Com o *plug-in* editor YANG totalmente desenvolvido é possível a sua utilização no Eclipse *runtime enviorment* simplesmente copiando o *plug-in* para a pasta *\plugins* existente no directório de plataforma Eclipse. Quando o Eclipse for inicializado, o novo *plug-in* será descoberto, disponibilizando todas as funcionalidades implementadas por este. O PDE, adicionalmente, possibilita a criação de *features* e *update sites*. Estas ferramentas permitem uma mais fácil divulgação e distribuição de *plug-ins* desenvolvidos pelos programadores. As Eclipse *features* são componentes não funcionais que representam um ou mais *plug-ins* encapsulando-os e promovendo uma melhor gestão das suas configurações, podendo ser detalhadas informações das suas funcionalidades e especificadas restrições ou requisitos para a sua instalação. Similarmente aos *plug-ins*, as *features* são descritas em ficheiros *manifest*, denominados *feature.xml*. Estes ficheiros descrevem o conjunto de *plug-ins* e as respectivas versões que a compõem. Adicionalmente a *feature* pode também indicar outros *plug-ins* que lhe são dependentes. As *features* podem ser distribuídas através de *update sites*. Um *update site* é tipicamente um directório Web acessível contendo todas as *features* e *plug-ins* em formato JAR disponíveis para *download* e instalação usando o assistente de instalação de novo software incluído no Eclipse.

Para o *plug-in* Editor YANG foi implementado um *update site* que inclui o *plug-in* editor YANG desenvolvido, na localização *http://atnog.av.it.pt/~ptavares/YangEditor,* disponibilizando a sua instalação, exibida na Figura 26, a qualquer pessoa interessada que pretenda fazer uso das suas funcionalidades.

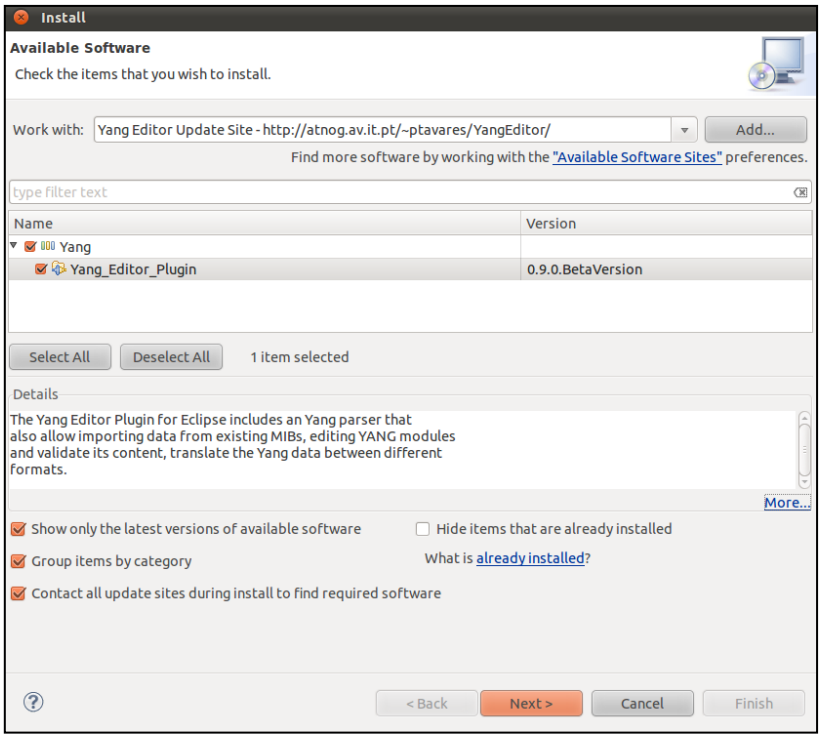

Figura 26: Assistente para a instalação do plug-in Editor YANG

3.4.3. Geração das Aplicações de Gestão

O editor permite a especificação e design de modelos de dados em formato YANG, permitindo igualmente a importação de módulos MIB para YANG e tradução de módulos YANG para YIN ou XSD, permitindo que estes possam ser integrados posteriormente num sistema distribuído de gestão NETCONF. A geração do *skeleton* das aplicações de gestão onde são criadas as *stubs* de comunicação SOAP que permitem a comunicação entre as entidades do sistema é realizada com o auxílio da *framework Eclipse Web Tools Plataform* [51]. O *plug-in Eclipse Web Tools Plataform* (WTP) permite o desenvolvimento de aplicações Web baseadas em J2EE, que inclui tecnologias Web strandards como HTML, XML e serviços Web. A Figura 27 ilustra a estrutura do Eclipse WTP *framework*.

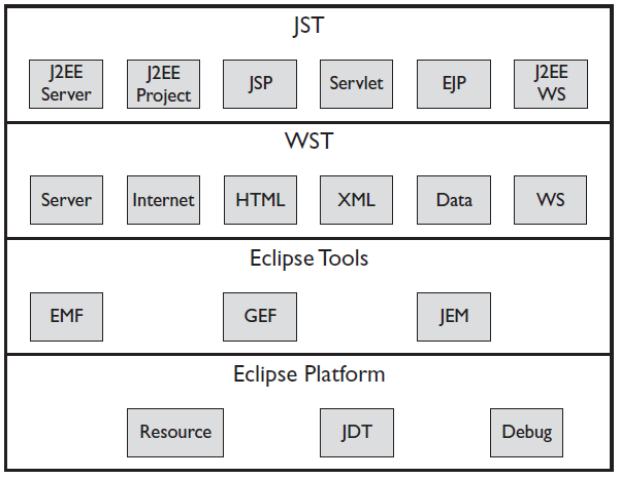

Figura 27: Arquitectura do Eclipse Web Tools Plataform

O WTP é formado por uma colecção de *plug-ins* que dependem entre si de uma forma organizada e controlada. A sua camada inferior encontra-se a plataforma Eclipse. Os projectos J2EE, que podem ser criados pelo WTP, são na sua essência projectos Java especificados numa estrutura amplificada. Por este motivo o WTP usa muitas das APIs presentes nos *plug-ins* Java *Development Tools* (JDT) e as *Debug APIs*. O WTP também aproveita os recursos do *Eclipse Modeling Framework* (EMF) [52] para a definição de modelos, como por exemplo, no desenvolvimento de descritores como *Web.xml* e *application.xml*. O WTP também providencia a visualização gráfica de documentos em formato XSD e WSDL através da *Graphical Editing Framework* (GEF) [53]. O *Java Edit Model* (JEM) faz parte do subprojecto *Visual Editor* e providência APIs de alto nível para o acesso de código fonte Java, estendendo o modelo Java fornecido pelo JDT. Na terceira camada, denominada WST (*Web Standard Tools*), contem varias ferramentas para a manipulação de documentos e normas Web para a criação de páginas Web, ficheiros XML, definição de bases de dados e implementação de serviços Web. A camada superior chamada JST (*J2EE Standard Tools*) estende a camada WST. Por exemplo o editor JSP estende o editor HTML, o J2EE *Web Services Tools* estende o XML *Web service Tools*, e o *J2EE Application Server Support* estende o *Core Server Support*.

O WTP inclui APIs para a adição de novos *server runtime environments* à plataforma Eclipse. Estes são designados por *server adapters* e permitem que o utilizador controle um determinado servidor, sendo possível a sua configuração, arranque, interrupção, *debug* e publicação de aplicações Web desenvolvidas.

Um dos servidores suportados pelo WTP é o *Apache Tomcat Server* [54]. O Tomcat é um servidor em código aberto desenvolvido pela *Apache Software Foundation*. Este implementa um servidor de aplicações Web suportando programação usando *JavaServer Pages* (JSPs) e *servlets*.

O WTP inclui um assistente para a criação de *Web Services* que permite a sua geração e implementação e a geração de clientes relativos a este. Um *Web Service* é um software que implementa um sistema designado para o suporte da interoperabilidade na interacção de diferentes equipamentos dentro de uma rede. Um *Web service* dispõe de uma interface que detalha as especificações do seu serviço, normalmente através de um ficheiro WSDL. Outros sistemas podem interagir com o servidor obedecendo a essas especificações usando mensagens SOAP, tipicamente sobre HTTP, podendo enviar ou receber dados em formato XML. Um ficheiro WSDL descreve os serviços disponibilizados à rede através de uma semântica XML, este providencia a informação de como a comunicação é realizada e o procedimento necessário para que esta comunicação se estabeleça. Enquanto que o SOAP especifica a comunicação entre um cliente e um servidor, o WSDL descreve os serviços oferecidos.

A implementação do serviço através do WTP também depende da presença de um *Web service runtime*, também vulgarmente designado por SOAP *engine*. Um dos *service runtime* suportados pelo WTP é o *Apache Axis2* [55]. O *Apache Axis2* é uma *framework* para o desenvolvimento de *Web Services* escrita em código livre pela *Apache Software Foundation*. O Axis2 usa o *Streaming* API for XML (Stax) para o *parsing* de XML e internamente usa um modelo de objectos *AXIs Object Model* (Axiom) para representar mensagens XML. O Axis2

suporta a geração de código a partir de documentos *Web Services Description Language* (WSDL), este código gerado lida com os detalhes da conversão das estruturas de dados especificados em formato XML usando uma *data binding framework* especificada, permitindo à aplicação acesso directo à estrutura de dados criada. O Axis2 também permite a operação inversa, ou seja, gerar um ficheiro WSDL a partir de código já existente. O Axis2 permite a geração de código quer para o servidor quer para os seus clientes. O código do cliente é criado sob a forma de uma classe *stub* que estende a classe *org.apache.axis2.client.Stub*. O código para o servidor assume a forma de *skeleton* para a implementação do serviço juntamente com uma classe para o processamento das mensagens recebidas que implementa a interface *org.apache.axis2.engine.MessageReceiver*. Ambos, cliente e servidor, são gerados a partir da ferramenta WSDL2Java incluída na *framework Apache Axis2*. Esta ferramenta pode ser directamente executada a partir da classe *org.apache.axis2.wsdl.WSDL2Java* como uma aplicação Java ou através de uma *Ant task*, um *plug-in Maven*, pelo próprio Eclipse ou um IDEA *plug-in* como ocorre no caso do WTP.

O processo de criação do servidor foi baseado no documento WSDL descrito na RFC 4743. O WTP requer que o utilizador crie um novo *Web Service Project* onde pode ser definido o *Tomcat Server* como *runtime server* e o *Apache Axis2* como *o Web service runtime*. O cliente pode ser gerado de forma similar, mas escolhendo um *Web Service Client Project*. O WTP permite duas alternativas para a implementação da *data binding* no processo de criação do *skeleton*: *XMLBeans data binding* e *Axis2 data binding* (ADB). Ambos permitem a geração de código derivado de ficheiros XSD embora o ADB apresente algumas limitações no suporte de *schemas* complexos em relação ao *XMLBeans* que consequentemente podem originar erros no código gerado. O *XMLBeans* apesar de gerar código mais complexo também permite uma melhor flexibilidade de implementação da aplicação uma vez que inclui vários métodos que facilitam a manipulação de código XML.

O processo de geração de código cria um conjunto de classes java para a criação da *stub* no cliente que implementam as respectivas mensagens que podem ser trocadas com o servidor. No servidor é criada uma classe *skeleton* para o processamento das mensagens enviadas pelos clientes e criadas as respectivas mensagens de resposta. O ficheiro WSDL deverá referenciar o ficheiro *netconf.xsd* para que sejam criadas classes que definam operações e tipos de dados do NETCONF e os respectivos métodos para a sua manipulação. Para que o sistema adicionalmente gere os tipos definidos num modelo de dados especificado em YANG este deve ser previamente convertido para o seu formato XSD e a sua importação incluída no WSDL.

Quando o assistente termina, o esqueleto da aplicação de gestão é criada e as classes geradas implementam integramente a comunicação entre o cliente e servidor NETCONF sem que seja necessário nenhuma especificação de código por parte do utilizador. O sistema necessitará apenas de ser completado com a especificação de código que implementa o comportamento que deverá ser realizado pelo servidor e os seus respectivos agentes NETCONF.

3.5. SUMÁRIO

No presente capítulo foi detalhado todo o desenvolvimento levado a cabo para a implementação do Editor YANG. Para a definição do modelo de dados em YANG foi desenvolvido um plug-in para o IDE Eclipse, que integrado com outras ferramentas já existentes para o processamento de módulos YANG e em conjunto com aplicações que permitem a geração e implementação de *Web Services*, proporcionam um sistema de edição de aplicações de gestão de rede baseadas no protocolo NETCONF.

Inicialmente foram descritos os requisitos para o Editor YANG e analisada a estrutura interna da plataforma Eclipse. O Eclipse é um IDE desenvolvido em java numa estrutura extremamente modular, que implementa diversas ferramentas de edição para as mais várias linguagens de programação. Este IDE inclui um mecanismo de extensão das suas funcionalidades através de um sistema de *plug-ins* e oferece um excelente suporte e documentação facilitando o desenvolvimento destes componentes por parte dos seus utilizadores.

Tendo consciência desta estrutura foi detalhada a construção do *parser* YANG e as suas funcionalidades relacionadas através da ferramenta *Xtext*. O *Xtext* é uma *framework* que permite o desenvolvimento de editores para uma determinada linguagem e a criação de interpretadores e compiladores totalmente integráveis no Eclipse, possibilitando um desenvolvimento muito mais simplificado, mas ao mesmo tempo sofisticado das várias funcionalidades que se pretendem implementar num editor específico.

Num projecto *Xtext* é fornecido um ficheiro para a definição da gramática. Este ficheiro não tem só a funcionalidade para a descrição concreta da sintaxe da linguagem como também define determinadas informações de como o *parser* deve estruturar o modelo durante o *parsing*. A gramática não é mais que um conjunto de regras definidas através de um conjunto de expressões com uma sintaxe bastante similar ao formato *Extended Backus-Naur Form*.

Depois de implementada a gramática YANG, o *Xtext* gera vários componentes para o processamento e validação dos dados introduzidos no editor. O *Xtext* deriva automaticamente da gramática diversos componentes relacionados com os aspectos visuais e de aperfeiçoamento da interacção do utilizador com o editor. Entre eles destacam-se o *content assist, outline view, quick outline, hyperlinking* e *syntax highlighting*. Todas estas funcionalidades podem ser personalizadas e estendidas através das suas APIs.

Foi também detalhado o desenvolvimento dos assistentes para o editor que permitem a criação de projectos e módulos YANG e a importação de módulos para o formato SMI para YANG e de um menu YANG para a implementação das funcionalidades de conversão de módulos YANG para os formatos YIN e XSD. Estes assistentes foram implementados com o auxílio da *framework JFace UI*. Esta providencia várias ferramentas e métodos para a criação de assistentes e diálogos. O processamento necessário para obter um módulo YANG traduzido de uma MIB é realizado pela aplicação Smidump. Para isso foi introduzida a execução de uma *system call* à aplicação Smidump no método *performFinish()* da classe ImportMIBWizard.

Ao editor YANG foi adicionalmente complementado com um novo menu para a realização das tarefas de conversão de módulos YANG para YIN e de conversão de módulos YANG para XSD, implementados através do uso de comandos. Os comandos em Eclipse são definidos através do ponto de extensão *org.eclipse.ui.commands*, podendo este ser categorizado, definido onde e como deve ser incluído na interface de utilizador e ser designado um conjunto de teclas de atalho para a sua invocação. O comportamento de um comando é definido através de *handlers*. O processamento necessário para a realização da tradução de YANG para YIN e de YANG para XSD é efectuado pela aplicação Pyang e, tal como implementado para o procedimento de importação de um documento MIB para YANG, é realizada uma *system call* mas agora à aplicação Pyang.

Na parte final do capítulo foi descrito o método para a geração das aplicações gestão baseadas no protocolo NETCONF através da ferramenta WTP integrada com as aplicações *Apache Tomcat Server* e do *Apache Axis 2.* O WTP inclui APIs para a adição de *novos server runtime environments* à plataforma Eclipse. Estes são designados por *server adapters* e permitem que o utilizador controle um determinado servidor. O WTP inclui um assistente para a criação de *Web Services* que permite a sua geração e implementação e a geração de clientes relativos a este. A implementação do serviço através do WTP também depende da presença de um *Web service runtime*, também vulgarmente designado por SOAP *engine*. Um dos service runtime suportados pelo WTP é o *Apache Axis2*. O Apache Axis2 é uma *framework* para o desenvolvimento de *Web Services* escrita em código livre pela *Apache Software Foundation*. O Axis2 suporta a geração de código a partir de documentos WSDL, este código gerado lida com os detalhes da conversão das estruturas de dados especificados em formato XML usando uma *data binding framework* especificada, permitindo à aplicação acesso directo à estrutura de dados criada. O Axis2 permite a geração de código quer para o servidor quer para os seus clientes. O código do cliente é criado sob a forma de uma classe *stub* que estende a classe *org.apache.axis2.client.Stub*. O código para o servidor assume a forma de *skeleton* para a implementação do serviço juntamente com uma classe para o processamento das mensagens recebidas que implementa a interface *org.apache.axis2.engine.MessageReceiver*. Ambos, cliente e servidor, são gerados a partir da ferramenta *WSDL2Java* incluída na *framework Apache Axis2* que é executada pelos assistentes incluídos no WTP para a criação de Web Services e seus respectivos clientes.

Para que o plug-in Editor YANG possa ser mais facilmente instalado e distribuído foi implementado um *update site* que inclui o plug-in editor YANG desenvolvido, na localização http://atnog.av.it.pt/~ptavares/YangEditor.

CAPÍTULO 4.
ANÁLISE DE RESULTADOS

O Editor YANG desenvolvido, detalhado no capítulo anterior, permite a criação simplificada de módulos YANG podendo estes serem derivados de módulos em formato SMI já existentes e a conversão do modelo criado para o formato XML *Schema* ou YIN. O capítulo anterior também descreve a tecnologia necessária, implementada conjuntamente com o editor para a geração das aplicações de gestão NETCONF. Neste capítulo é descrito em pormenor a metodologia necessária para a implementação de sistema de gestão que integre um modelo de dados definido em YANG sendo, no final do capítulo, realizado um teste ao sistema instalado comprovando a sua funcionalidade.

O diagrama de actividade apresentado na Figura 28 ilustra o fluxo completo das acções que podem ser executadas no Editor YANG implementado.

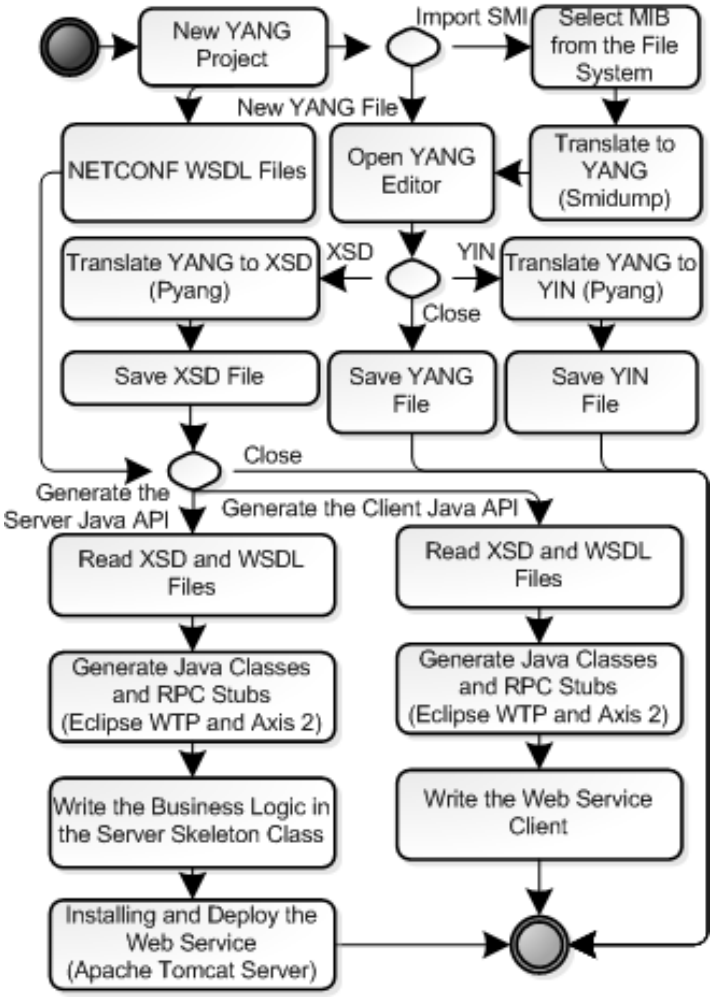

Figura 28: Diagrama de actividade do sistema

No início, depois de ter sido criado um projecto YANG, o utilizador pode optar por importar uma MIB existente para o seu formato YANG correspondente ou definir um módulo YANG a partir do zero. Desde que os módulos YANG se encontrem sintacticamente validados é possível efectuar a sua exportação para os formatos YIN ou XSD. O processo de edição pode ser interrompido a qualquer instante, tendo a possibilidade de poder ser retomado mais tarde através da salvaguarda do modelo de dados em qualquer um dos formatos suportados.

A *framework Eclipse Web Tools Plataform*, integrada com as aplicações *Apache Tomcat Server* e do *Apache Axis2,* permite a geração e implementação de uma aplicação de gestão de rede distribuída baseada no NETCONF, onde são criados os diversos componentes que permitem a implementação de um gestor e dos seus respectivos agentes individualmente.

A aplicação é na sua essência gerada através da definição do NETCONF *Web Service* descrita nos WSDLs presentes na RFC4743. Estes ficheiros WSDLs descrevem a localização, as mensagens e os conteúdos passíveis de serem trocadas na execução do serviço. Para que este serviço implemente um modelo de dados YANG de modo que possa ser integrado nas aplicações de gestão assim como nas operações NETCONF este deve ser previamente traduzido para XSD e a sua importação inserida na própria descrição do serviço.

Quando é realizada a criação do servidor NETCONF são geradas as classes que implementam as *stubs,* sendo necessário o posterior preenchimento do *skeleton* dos métodos *rpc* e *hello* que definam o domínio lógico pretendido a ser implementado pelo equipamento. No caso da criação do cliente NETCONF será necessário a criação de um programa *main* java onde o utilizador pode especificar os conteúdos das mensagens baseados no protocolo NETCONF e no modelo de dados previamente definido a ser enviados pelo *stub* nos respectivos métodos *rpc* e *hello*.

4.1. Criação de uma Aplicação de Gestão

A presente secção descreve o processo de desenvolvimento de uma aplicação de gestão NETCONF utilizando o nosso ambiente de desenvolvimento. O método envolve todo o processo para o desenvolvimento de uma aplicação de gestão distribuída desde a criação de um modelo de dados YANG, à sua tradução para o formato XSD, criação da aplicação de gestão, implementação da lógica de serviço e criação de um programa para teste do serviço num cliente. No processo de desenvolvimento da aplicação de gestão foi seleccionada uma MIB SNMP conhecida e foi precorrido todo o processo de desenvolvimento até à depuração de erros da aplicação de gestão gerada.

4.1.1. Criação do Modelo de Dados

No processo de desenvolvimento foi usado como base o módulo SMI IP-MIB [56]. A sua escolha foi baseada no facto de esta pertencer uma MIB largamente conhecida e a sua informação ser normalmente usada na gestão de operações de elementos de rede.

A MIB IP-MIB depende directa ou indirectamente dos módulos INET-TYPES, SNMPv2-TC, INET-ADDRESS-MIB, IF-MIB, IANAifType-MIB e YANG-SMI. Estes módulos deverão ser agrupados numa pasta onde a sua localização deverá ser atribuída à variável de ambiente SMIPATH. Esta variável é usada pela aplicação Smidump para a pesquisa de módulos importados por outros módulos quando é realizada a conversão de MIB para YANG.

Um novo projecto YANG deverá ser criado no editor e os módulos SMI deverão ser importados sequencialmente para o projecto. O processo de importação de MIB para YANG cria um novo módulo como o mesmo nome e com um novo *namespace*. A estrutura MIB descrita em cada módulo é traduzida de SMI para YANG e é criado um novo módulo YANG sendo definidos, cada um dos seus elementos especificados no módulo SMI, no seu correspondente sintaxe YANG.

Posteriormente, cada um dos novos módulos YANG deverão ser traduzidos para o seu correspondente formato XSD. Para isso deverá ser especificada uma variável de ambiente YANG_MODPATH, que deverá apontar para a pasta que contém os módulos YANG. Esta variável é usada pela aplicação Pyang para descobrir os módulos que são importados pelos módulos a traduzir. Tal como no processo de tradução de MIB para YANG, são criados novos documentos mas agora em formato XSD que incluem os elementos definidos no correspondente módulo YANG.

Foi incluído em anexo o ficheiro IP-MIB que contém a definição da estrutura *ipAddressEntry,* e a suas respectivas conversões para o formato YANG e de YANG para o formato XSD que foram geradas com o auxílio do Editor YANG desenvolvido, apresentados pelos ficheiros *IP-MIB.yang* e *IP-MIB.xsd*.

4.1.2. Geração das Aplicações de Gestão

Para gerar convenientemente o serviço e os seus agentes todos os ficheiros XSD devem ser colocados na pasta *Management_App* juntamente com os ficheiros WSDL e XSD relativos ao NETCONF que são gerados quando é criado um novo projecto YANG. O ficheiro *netconf-soap_1.0.wsdl* deverá ser editado com a definição da importação do ficheiro *IP-MIB.xsd*. Esta definição deve ser realizada após o comentário *<!– Import your custom XSD file here–>* como é ilustrado na Figura 29. Não é necessária a importação dos outros restantes ficheiros XSD uma vez que na criação do serviço qualquer ficheiro referenciado por um *schema* será automaticamente incluído, sendo consequentemente criados os tipos de dados definidos por este.

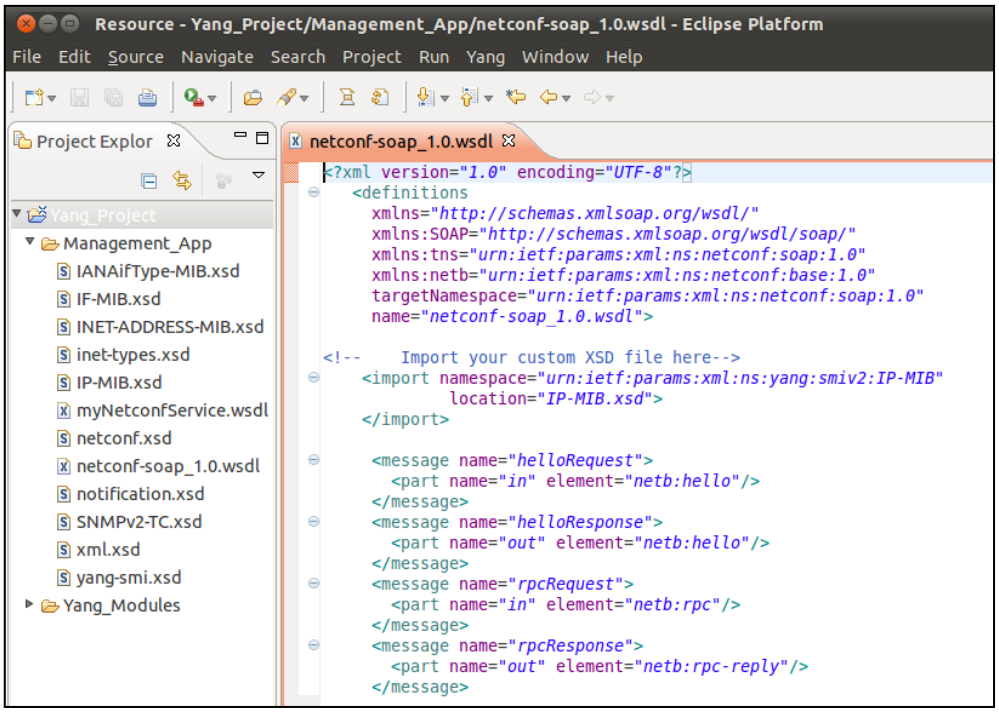

Figura 29: Conteúdo do ficheiro netconf-soap_1.0.wsdl

O ficheiro *myNetconfService.wsdl,* exibido na Figura 30, importa o ficheiro *netconf-soap_1.0.wsdl* e especifica a localização pretendida para o serviço. Por defeito o serviço é criado no próprio equipamento, no *localhost*, mas o utilizador pode optar por editar a *tag SOAP:address* para definir a localização que pretende implementar o serviço.

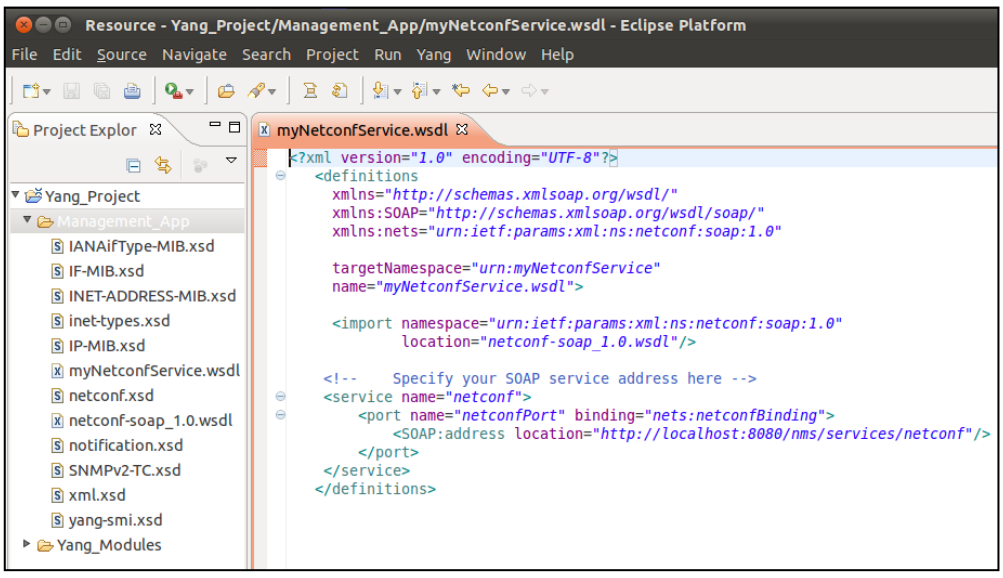

Figura 30: Conteúdo do ficheiro myNetconfService.wsdl

4.1.2.1. GERAÇÃO DO SERVIÇO

A geração do servidor é realizada pelo assistente de criação de *Web Services* incluído no WTP. A configuração do assistente deverá ser similar à apresentada na Figura 31 onde a definição do serviço deverá referenciar o ficheiro *myNetconfService.wsdl* e serem especificados o *server runtime* e o *Web service runtime* respectivamente com o *Tomcat Server* e o *Apache Axis2*.

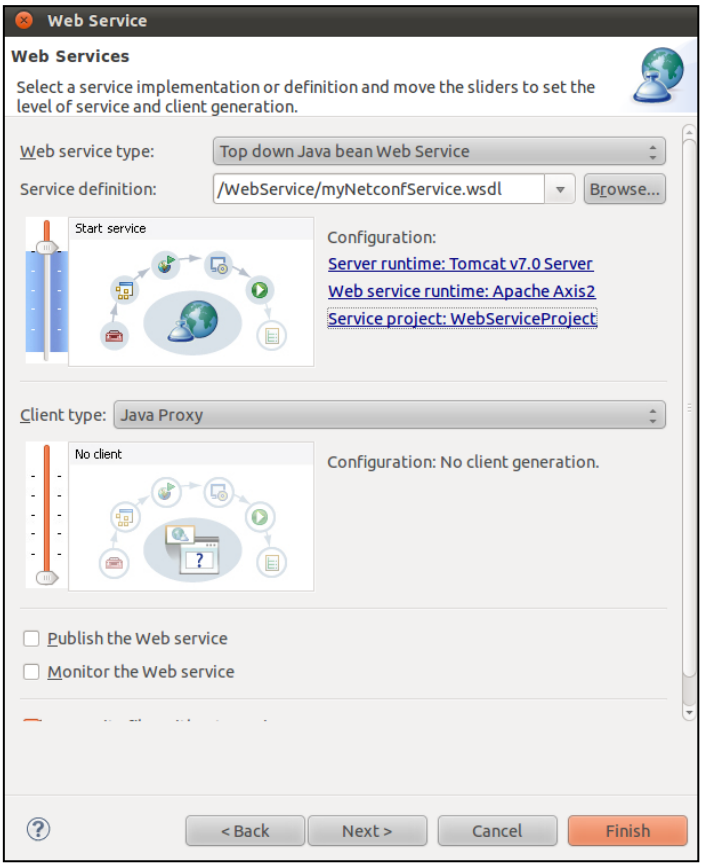

Figura 31: Assistente de criação de um Web Service

O assistente irá criar um novo *Web Service Project* no *workspace* com o nome *WebServiceProject*. Na página seguinte do assistente podem ser configuradas algumas das opções relativas à geração do código do servidor onde deve ser especificado o XMLBeans como *data binding* a ser usado. Outras opções são automaticamente retiradas do ficheiro WSDL como é ilustrado na Figura 32.

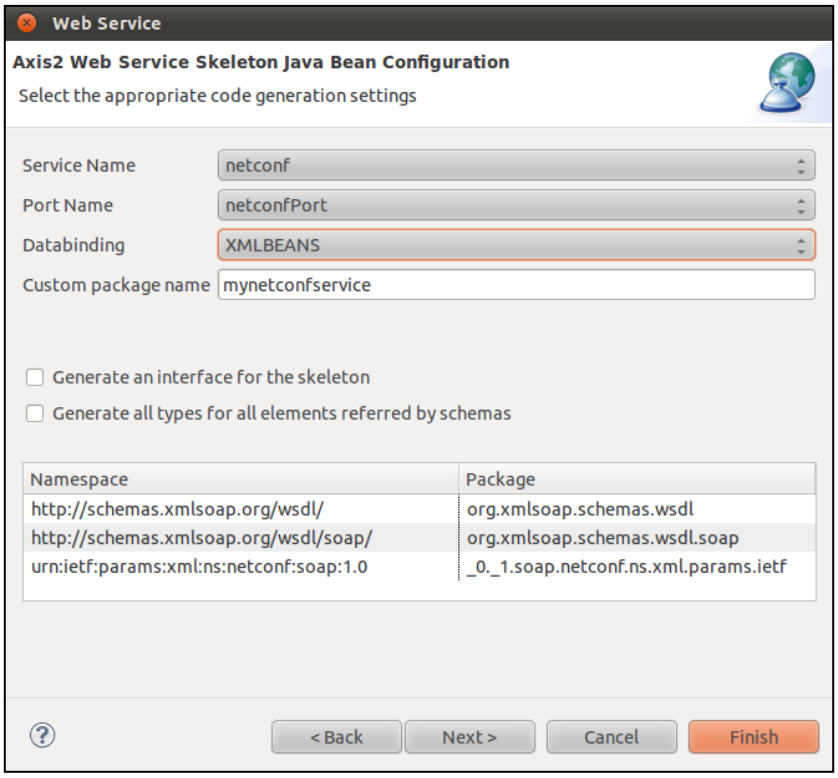

Figura 32: Assistente de configuração das opções para a geração do código do servidor

Caso o servidor *Tomcat* não se encontre em execução o assistente requisitará ao utilizador que o active e quando o assistente termina o novo serviço criado é instalado e iniciado no servidor. O utilizador pode verificar se o serviço se efectivamente encontra em execução ao inserir num browser a localização do serviço onde será apresentada uma pagina com os serviços disponíveis e as respectivas operações, devendo estar incluído o serviço NETCONF e as operações *rpc* e *hello*.

O projecto *WebServiceProject* conterá um *package* chamado *mynetconfservice* que inclui a classe *NetconfSkeleton* que implementa os métodos *rpc* e *hello*. Esta classe deverá ser completada pelo utilizador definindo a lógica de serviço que pretende que seja realizada pelo servidor. A Figura 33 ilustra o editor e em particular a classe *NetconfSkeleton* depois de concluído o processo de geração do esqueleto da aplicação.

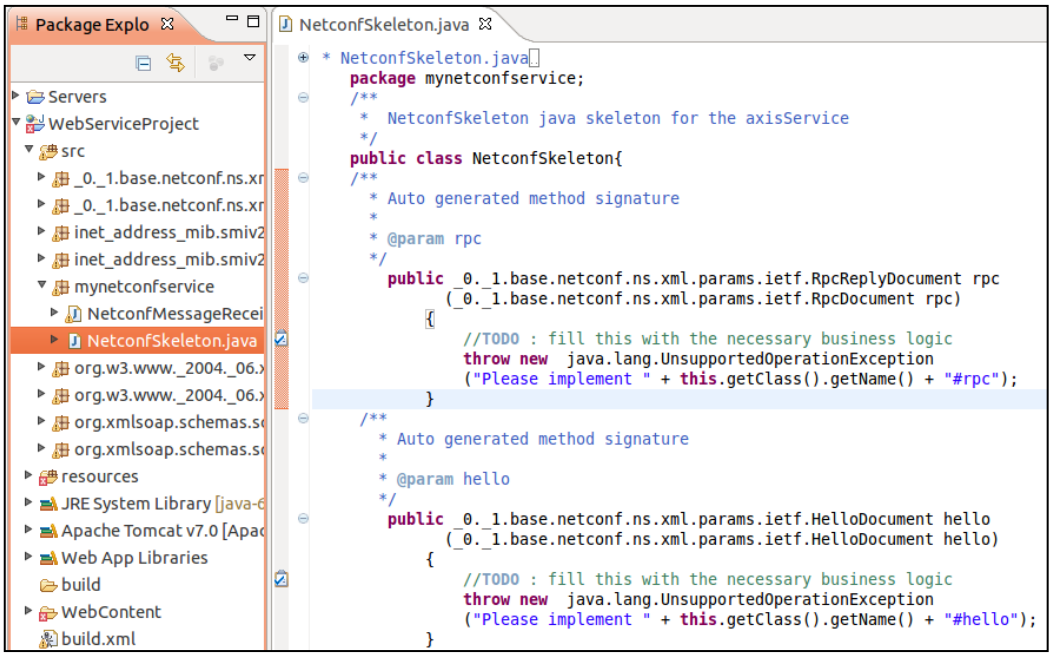

Figura 33: Conteúdo do ficheiro NetconfSkeleton.java

4.1.2.2. Geração do Cliente

O cliente NETCONF é igualmente criado com o auxílio do WTP através do seu assistente para a criação de *Web Service Client*. Se for pretendido implementar o cliente em uma outra máquina será necessário replicar o mesmo projecto YANG contendo a mesmos ficheiros e informações que permitiram a criação do servidor. No assistente de criação do cliente é em tudo semelhante ao processo de criação servidor onde deverá ser novamente referenciado o ficheiro *myNetconfService.wsdl* e designados o *Tomcat Server* como *server runtime* e *o Apache Axis2* como *Web service runtime*. Na secção para a especificação das preferências para a geração do código do cliente deverá ser novamente designado o *XMLBeans* como *data binding* podendo as restantes opções serem deixadas com os valores por defeito definidos pelo assistente.

No final o assistente criará um novo projecto no *workspace* com o nome *WebServiceProjectClient*. Este projecto incluirá o *package mynetconfservice* que inclui a classe *NetconfStub* que implementa a comunicação com o servidor e as mensagens que o cliente pode enviar. A Figura 34 ilustra o editor e a classe *NetconfStub* depois de concluído o processo de geração do cliente.

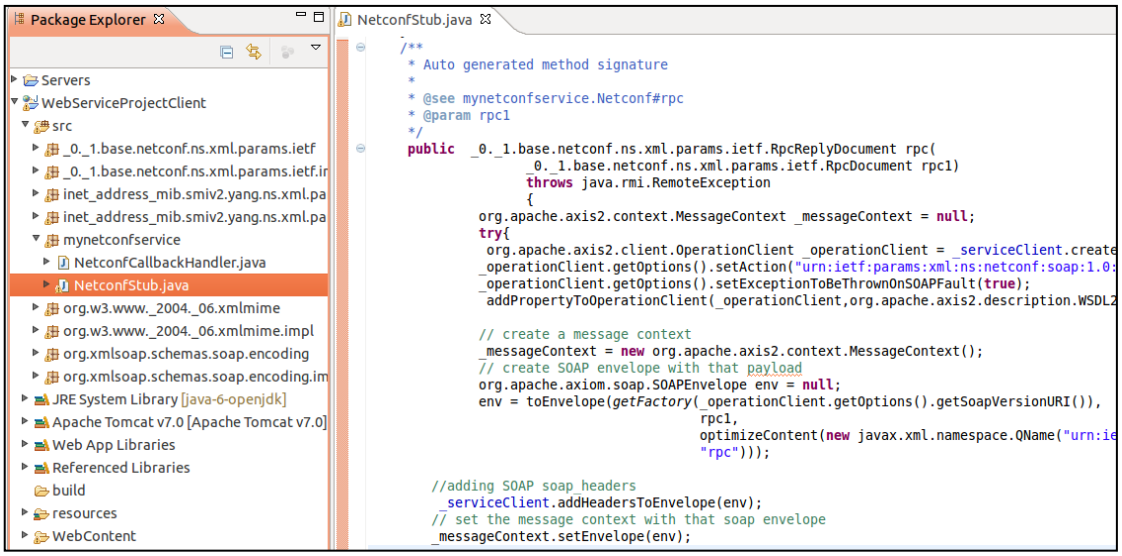

Figura 34: Conteúdo do ficheiro NetconfStub.java

4.1.3. SISTEMA IMPLEMENTADO

O sistema criado encontra-se, em parte, representado pelo diagrama de classes ilustrado na Figura 35. A classe *NetconfStub* implementa a comunicação do cliente com o serviço, designando as operações que este pode realizar. Estas operações, quando enviadas, são processadas no servidor pela classe *NetconfSkeleton*. As classes *NetconfStub* e *NetconfSkeleton* empregam os tipos de dados definidos relativos ao NETCONF onde as classes *ConfigInlineType* e *DataInlineType* derivam tipos de dados definidos nos restantes ficheiros XSD, usados na criação do sistema, como é o caso da classe *IPType*, originada do *schema* IP-MIB, que permite a instanciação do tipo de dados *IPAddressEntryType* definido pela estrutura *ipAddressEntry* incluída no módulo IP-MIB.

Uma vez implementado, o sistema permite a comunicação entre o cliente e o servidor. Para o cliente interagir com o serviço será necessário a criação de um programa principal em Java, que pode ser executado a partir do Eclipse como uma aplicação deste tipo, onde deverá ser instanciada uma variável *NetconfStub* a partir da qual podem ser enviadas as mensagens *hello* ou *rpc*. O servidor responde respectivamente com mensagens *hello* e *rpc-reply*, possibilitando ao cliente a gestão do equipamento. A gestão do equipamento é realizada com o auxílio das operações NETCONF que podem ser incluídas nas mensagens *rpc*, podendo estas operações, por sua vez, incluir tipos de dados derivados de um modelo de dados previamente definido em YANG. O servidor também compreende essas operações e tipos de dados uma vez que implementa exactamente as mesmas classes definidas no cliente, processando as mensagens recebidas e agindo de acordo com a lógica implementada na sua classe *skeleton*.

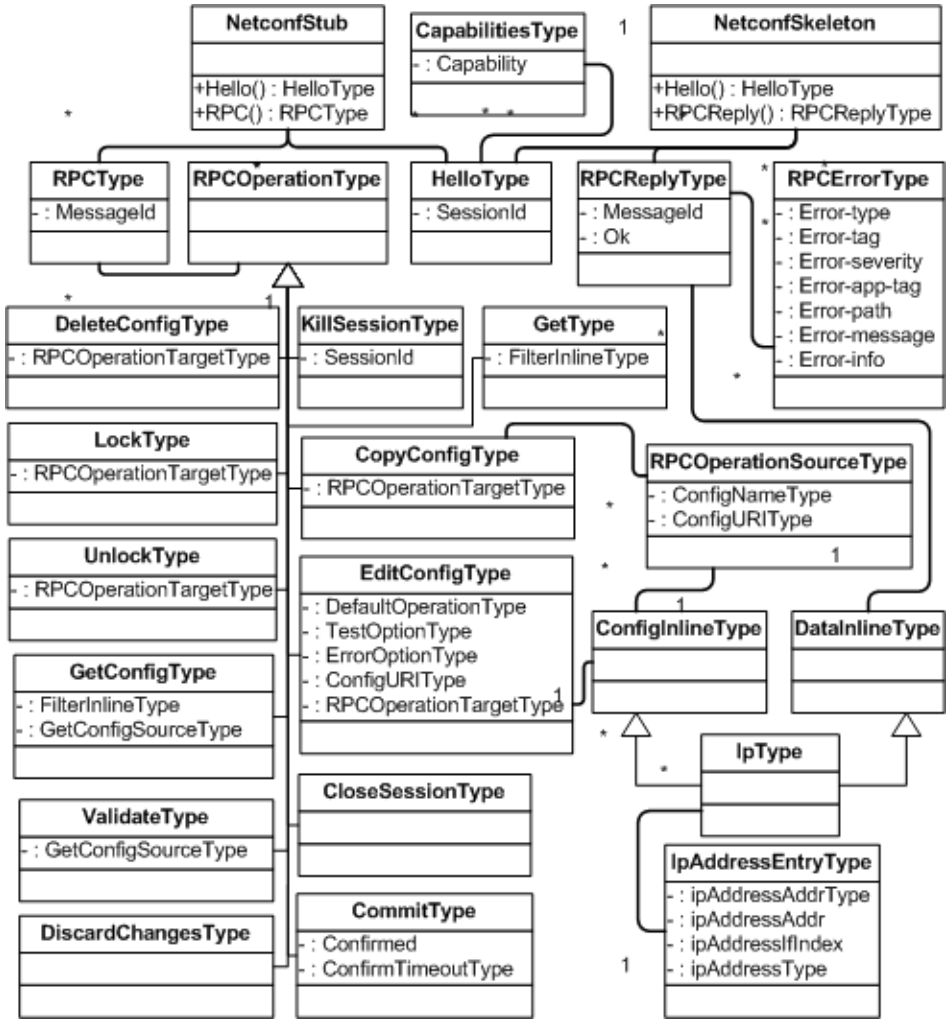

Figura 35: Diagrama de classes do sistema

4.1.4. TESTE DO SISTEMA

Uma interacção possível de ser realizada pelo sistema implementado é apresentada na próxima secção. Os ficheiros que definem esta interacção encontram-se apresentados em anexo, onde os ficheiros *AgenteNetconf.java* e *NetconfSkeleton.java* foram implementados respectivamente no cliente e no servidor. Uma vez que o sistema usa o protocolo SOAP como transporte é possível serem capturadas as mensagens enviadas por ambas as entidades, comprovando a correcta execução do sistema implementado. Para iniciar a sessão NETCONF o cliente envia uma mensagem *hello,* ilustrada na Figura 36, onde inclui a capacidade base do NETCONF identificada pela URI *urn:ietf:params:netconf:base:1.0.*

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
       <urn:hello
          xmlns:urn="urn:ietf:params:xml:ns:netconf:base:1.0">
          <urn:capabilities>
             <urn:capability>
                urn:ietf:params:netconf:base:1.0
                </urn:capability>
             </urn:capabilities>
          </urn:hello>
       </soapenv:Body>
    </soapenv:Envelope>
```

Figura 36: Mensagem Hello enviada pelo cliente NETCONF

O servidor responde com a sua respectiva mensagem *hello,* apresentada na Figura 37, que incluindo a mesma capacidade e define o *id* da sessão com o valor 4.

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <urn:hello
         xmlns:urn="urn:ietf:params:xml:ns:netconf:base:1.0">
         <urn:capabilities>
            <urn:capability>
               urn:ietf:params:netconf:base:1.0
               </urn:capability>
            </urn:capabilities>
         <urn:session-id>
            4
            </urn:session-id>
         </urn:hello>
      </soapenv:Body>
   </soapenv:Envelope>
```

Figura 37: Mensagem Hello enviada pelo servidor NETCONF

Com a sessão estabelecida o cliente pode gerir o equipamento com o envio de mensagens *rpc* especificando as operações NETCONF que pretende realizar. No excerto da mensagem exibida na Figura 38, o cliente envia um *rpc* com um *message-id* igual a '101', contendo a operação *edit-config* onde define o elemento *ipAddressEntry,* definido pelo módulo IP-MIB, com o valor 192.168.1.1.

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
       <urn:rpc
          message-id="101"
          xmlns:urn="urn:ietf:params:xml:ns:netconf:base:1.0">
          <urn:rpcOperation
             xsi:type="urn:editConfigType"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
             <urn:config>
                <urn1:ip
                   xmlns:urn1="urn:ietf:params:xml:ns:yang:smiv2:IP-MIB">
                   <urn1:ipAddressEntry>
                      192.168.1.1
                      </urn1:ipAddressEntry>
                   </urn1:ip>
                </urn:config>
             </urn:rpcOperation>
          </urn:rpc>
       </soapenv:Body>
    </soapenv:Envelope>
```

Figura 38: Mensagem Rpc com a operação <edit-config> enviado pelo cliente NETCONF

O servidor responde com a correspondente mensagem *rpc-reply* ilustrada na Figura 39, contendo o mesmo *message-id* e inclui um elemento <*OK*> confirmando ao cliente a definição desse parâmetro.

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
       <urn:rpc-reply
          message-id="101"
          xmlns:urn="urn:ietf:params:xml:ns:netconf:base:1.0">
          <urn:ok
             xsi:nil="true"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
          </urn:rpc-reply>
       </soapenv:Body>
    </soapenv:Envelope>
```

Figura 39: Mensagem Rpc-Reply confirmando a alteração dos parâmetros do servidor NETCONF

O excerto da mensagem mostrada na Figura 40 representa o envio de um *rpc* por parte do cliente incluindo um *message-id* com o valor '102' requerendo a configuração do equipamento através da operação NETCONF *<get-config>*.

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
       <urn:rpc
          message-id="102"
          xmlns:urn="urn:ietf:params:xml:ns:netconf:base:1.0">
          <urn:rpcOperation
             xsi:type="urn:getConfigType"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
          </urn:rpc>
       </soapenv:Body>
    </soapenv:Envelope>
```

Figura 40: Mensagem Rpc com a operação <get-config> enviada pelo cliente NETCONF

À anterior mensagem o servidor responde com um rpc-reply exibida na Figura 41 contendo o mesmo valor no seu elemento *message-id* e um elemento *<data>* contendo as configurações contidas no equipamento que foram previamente definas pela operação *<edit-config>*.

```
<soapenv:Envelope
      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Body>
         <urn:rpc-reply
            message-id="102"
            xmlns:urn="urn:ietf:params:xml:ns:netconf:base:1.0">
            <urn:data
               xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
               <urn1:ip
                  xmlns:urn1="urn:ietf:params:xml:ns:yang:smiv2:IP-MIB">
                  <urn1:ipAddressEntry>
                     192.168.1.1
                     </urn1:ipAddressEntry>
                  </urn1:ip>
               </urn:data>
            </urn:rpc-reply>
         </soapenv:Body>
      </soapenv:Envelope>
```

Figura 41: Mensagem Rpc-Reply contendo os dados de configuração do servidor NETCONF

4.2. SUMÁRIO

Este capítulo resume as funcionalidades implementadas pelo Editor YANG descrevendo todas as actividades passíveis de poderem ser realizadas por este. É apresentado um exemplo prático para a criação de uma aplicação de gestão que integra um modelo de dados previamente definido em YANG sendo descritas todas as fases do o ciclo de desenvolvimento das aplicações.

Neste exemplo foi usado como base o módulo SMI IP-MIB. A sua escolha foi baseada no facto de esta pertencer uma MIB largamente conhecida e a sua informação ser normalmente usada na gestão de operações de elementos de rede. Depois de ter sido criado um novo projecto YANG o mecanismo de importação permite que a criação de um novo modulo YANG derivado desta MIB.

O processo de importação de MIB para YANG cria um novo módulo com o mesmo nome e com um novo *namespace*. A estrutura MIB descrita no módulo é traduzida de SMI para YANG sendo criado um novo módulo YANG onde são definidos os elementos especificados no módulo SMI no seu correspondente sintaxe YANG. Desde que os módulos YANG se encontrem sintacticamente validados é possível efectuar a sua exportação para os formatos YIN ou XSD que incluem os elementos definidos no correspondente módulo YANG.

A *framework Eclipse Web Tools Plataform*, integrada com as aplicações *Apache Tomcat Server* e do *Apache Axis2*, permite a geração e implementação de aplicações de gestão de rede baseada no NETCONF, onde são gerados os seus diversos componentes que permitem a criação de um gestor e dos seus respectivos agentes individualmente.

As aplicações são, na sua essência, geradas através da definição do NETCONF *Web Service* descrita nos WSDLs presentes na RFC4743. Estes ficheiros WSDLs descrevem a localização, as mensagens e os conteúdos passíveis de serem trocadas na execução do serviço. Para que as aplicações de gestão implementem um modelo de dados YANG assim como as operações NETCONF, estes documentos devem-se encontrar num formato XSD e a sua importação inserida declarativamente na própria descrição do serviço.

Quando é realizada a criação do servidor NETCONF são geradas as classes que implementam as *stubs*, sendo necessário o posterior preenchimento do *skeleton* dos métodos *rpc* e *hello* que definam o domínio lógico pretendido a ser implementado pelo sistema. A classe *NetconfStub* implementa a comunicação do cliente com o serviço, designando as operações que este pode realizar. Estas operações, quando enviadas, são processadas no servidor pelo *NetconfSkeleton*. As classes *NetconfStub* e *NetconfSkeleton* empregam os tipos de dados definidos relativos ao NETCONF onde as classes *ConfigInlineType* e *DataInlineType* derivam tipos de dados definidos nos restantes ficheiros XSD usados na criação do sistema.

Uma vez implementado, o sistema permite a comunicação entre o cliente e o servidor. Para o cliente interagir com o serviço será necessária a criação de um programa principal em Java, que pode ser executado a partir do Eclipse, onde deverá ser instanciada uma variável *NetconfStub* a partir da qual podem ser enviadas as mensagens *hello* ou *rpc*. O servidor responde respectivamente com mensagens *hello* e *rpc-reply*, possibilitando ao cliente a gestão do equipamento. A gestão do equipamento é realizada com o auxílio das operações NETCONF que podem ser incluídas nas mensagens *rpc*, podendo estas operações, por sua vez, incluir tipos de dados derivados de um modelo de dados. O servidor também compreende essas operações e tipos de dados, uma vez que implementa exactamente as mesmas classes definidas no cliente, processando as mensagens recebidas e agindo de acordo com a lógica implementada na sua classe *skeleton*.

Depois de gerados o cliente e o servidor foi criado um programa java no cliente para a realização um teste genérico ao sistema. Neste teste são trocadas mensagens *hello* entre ambas as entidades e posteriormente são enviadas duas mensagens *rpc* pelo cliente, as quais o servidor responde com mensagens *rcp-reply,* para a edição e obtenção das configurações do equipamento comprovando a total funcionalidade do sistema de gestão de rede implementado.

CAPÍTULO 5.
CONCLUSÕES E TRABALHO FUTURO

Esta dissertação descreve a implementação de um ambiente de desenvolvimento integrado para a criação de aplicações de gestão baseadas no protocolo NETCONF. Permite percorrer o ciclo completo de desenvolvimento desde a definição do modelo de dados até ao *debugging* da aplicação final.

Apesar de a tecnologia de gestão NETCONF ser ainda uma tecnologia emergente, esta tem sido progressivamente aceite pela comunidade associada ao domínio de redes, recebendo uma cada vez maior atenção por parte de empresas e institutos ou universidades.

5.1 CONCLUSÕES

O NETCONF é um protocolo de gestão de elementos de rede que utiliza tecnologias Web, tais como SOAP e XML para o transporte de mensagens e para a codificação dos elementos dos dados de gestão. O protocolo define as mensagens que são trocadas entre o agente e o gestor bem como a sua temporização, mas não define o modelo de dados a ser utilizado pelos elementos de gestão. O protocolo tem como base o paradigma RPC onde o cliente, usando uma sessão segura e orientada à ligação, codifica o seu RPC num formato XML e envia-o ao servidor; o servidor responde com um RPC-Reply que é novamente codificado em XML que é devolvido ao cliente.

O NETCONF ultrapassa muitas das limitações impostas pelo SNMP assim como de outros protocolos de gestão, providenciando uma melhor configuração de equipamentos de rede através o uso eficaz da tecnologia XML conjuntamente com outras tecnologias Web. Estas características têm impelido o NETCONF a ser visto como um protocolo promissor, sendo cada vez mais adoptado em novas configurações para a gestão de equipamentos de rede. O protocolo também permite que seja implementado numa variedade de ambientes, onde se inclui a sua implementação sob a forma de um serviço Web que através de ferramentas SOAP que possibilitam a geração automática baseada nas informações especificadas em documentos WSDL.

Para tornar o NETCONF interoperável e capaz de manipular os dados de configurações de forma normalizada foi criada a linguagem de modelação de dados designada YANG. A sintaxe YANG é similar a linguagens de programação C ou C++ promovendo a sua legibilidade e o desenvolvimento de módulos por parte dos administradores de redes e permitindo a modelação da semântica e organização dos dados de estado e configuração, de operações e de notificações a serem manipuladas pelo protocolo NETCONF.

A solução desenvolvida integra um editor para a linguagem YANG sob a forma de um *plug-in* para o Eclipse. O *parser* para a linguagem YANG e todas as outras funcionalidades

relacionadas presentes no editor foram desenvolvidos com o auxílio da ferramenta *Xtext*. O *Xtext* é uma *framework* de desenvolvimento de editores para uma linguagem genérica e permite a criação de interpretadores e de compiladores totalmente integráveis no Eclipse, possibilitando um desenvolvimento muito mais simplificado, mas ao mesmo tempo permitindo implementar um conjunto rico de funcionalidades tipicamente disponíveis editores mais amigáveis e mais avançados. O *Xtext* permite derivar automaticamente da gramática diversos componentes relacionados com os aspectos visuais e de aperfeiçoamento da interacção do utilizador com o editor. O editor YANG desenvolvido inclui todas as normais funcionalidades de edição disponíveis no IDE Eclipse, tais como o realce de sintaxe, detecção de erros, *outline*, assistente de conteúdo, entre outras. Tendo sido algumas destas funcionalidades posteriormente personalizadas e estendidas através das suas APIs. A utilização da ferramenta *Xtext* neste trabalho provou ser uma mais valia uma vez que simplificou a criação do editor ao mesmo tempo que proporcionou um ambiente completo e de aspecto profissional para a edição de módulos em linguagem YANG. O editor ao ter sido implementado nesta tecnologia permite que este possa ser compreendido sem grandes dificuldades por parte de um qualquer programador podendo este ser facilmente actualizado, estendido ou complementado em futuras versões.

O editor foi complementado com assistentes para a criação de projectos YANG, criação de novos módulos e para a importação de módulos SMI para o seu formato YANG correspondente. Estes assistentes facilitam a interacção do utilizador com o editor auxiliando e simplificando as tarefas que este pretende realizar. Em particular o assistente para a criação de um novo projecto YANG inclui automaticamente no projecto todos os ficheiros base necessários para a geração das aplicações de gestão em NETCONF. O editor foi também complementado com um menu YANG permitindo que os módulos YANG, desde que se encontrem sintacticamente validados, ao utilizador efectuar a sua exportação para os formatos YIN ou XSD.

As aplicações *Smidump* e *Pyang,* que permitem a importação de módulos SMI para YANG e a tradução de módulos YANG para YIN ou XSD, respectivamente. Aumentam a interoperabilidade do editor uma vez que unicamente podem ser executadas em ambientes *Unix* ou *MAC*.

O WTP integrado com o *Apache Tomcat Server* e a *framework Apache Axis2*, permite adicionar ao IDE a possibilidade de geração da aplicação de gestão em NETCONF a partir do WSDL retirado da RFC 4743, podendo este integrar um módulo YANG previamente definido no editor convertido no seu formato XSD. A incorporação do WTP conjuntamente com o editor simplifica a criação do *skeleton* e *stubs* para os respectivos servidores e clientes. Todo este processo é automatizado pelos seus assistentes, validando os conteúdos que permitem a geração das aplicações, diminuindo e agilizando a quantidade de tarefas ou consequentes erros caso este processo tivesse de ser realizado manualmente. A ferramenta *Apache Axis2* provou ser fundamental no processo de geração uma vez que permite lidar com o *parsing* do XML, geração de classes Java a partir de documentos WSDL e implementação da comunicação através da camada de transporte SOAP, permitindo que o utilizador apenas se foque na implementação da lógica do serviço da aplicação de gestão.

A integração de funcionalidades para a especificação de um modelo de dados com um ferramentas para o desenvolvimento e geração automática de código permite que todo o ciclo de desenvolvimento possa ser unicamente realizado dentro do próprio Eclipse possibilitando que o processo de criação das aplicações seja mais rápido e de fácil adaptação por parte de um normal utilizador e evitando processos de exportação de dados entre diferentes formatos em diferentes aplicações.

O *plug-in* Editor YANG torna todo o processo de especificação e design de módulos YANG muito mais facilitado dispondo de inúmeras funcionalidades de edição não presentes em mais nenhuma aplicação de processamento de linguagem YANG do género. Para além da especificação de módulos YANG o sistema permite a geração simplificada de toda a aplicação de gestão podendo ser usada por administradores para o desenho e prototipagem de soluções NETCONF, testando modelos de dados YANG e as operações NETCONF antes de as implementar num sistema de rede real.

5.2 Trabalho Futuro

Apesar de serem completamente funcionais, as aplicações que são geradas pelo IDE não suportam o protocolo de transporte SSH definido como obrigatório pelo protocolo NETCONF na RFC 4741. Outra limitação da implementação cinge-se com a falta de um mecanismo de autenticação, que, segundo o artigo [57], também se tornará obrigatório para sessões NETCONF.

O suporte para a monitorização não foi considerado para a implementação do sistema, apesar da sua simplicidade e do baixo número de operações envolvidas. A sua dificuldade de implementação reside no facto de que durante o processo de monitorização o agente e gestor trocam de papéis em relação ao modelo de cliente-servidor que foi implementado. Em mais pormenor o servidor, depois de receber uma mensagem de subscrição de notificações por parte de um cliente, necessita de poder enviar independentemente mensagens *notification* sempre que um determinado evento subscrito ocorre, e o cliente necessita de ter activo um mecanismo para que possa receber e processar essas mensagens.

Também foi considerado a inclusão da aplicação *Pyang* no próprio *plug-in* Editor YANG, pois apesar de esta aplicação ser escrita em *python* já existem interpretadores dessa linguagem para Java. Esta alteração será uma possível evolução do projecto permitindo a eliminação da sua dependência por parte do editor.

O processo de desenvolvimento da aplicação distribuída de gestão também não inclui o suporte para a troca automática de capacidades forçando que esta seja desenvolvida por parte do utilizador depois de criada a aplicação na respectiva classe S*keleton*. Esta limitação poderia ser ultrapassada em futuras versões do editor, através da implementação de um componente de *software* adicional, a ser executado posteriormente à criação do serviço, uma vez que as operações adicionais suportadas por um dispositivo NETCONF podem ser

descritas em módulos YANG e assim o seu anúncio inserido automaticamente no método *hello* da classe *Skeleton*.

6. REFERÊNCIAS

[1] D. C. Verma, *Principles of Computer Systems and Network Management*: Springer, 2009.

[2] *A Simple Network Management Protocol (SNMP),* RFC 1157, Maio de 1990.

[3] *Management Information Base for Network Management of TCP/IP-based internets: MIB-II,* RFC 1213, Março de 1991.

[4] *The Common Management Information Services and Protocols for the Internet (CMOT and CMIP),* RFC 1189, Outubro de 1990.

[5] J. G. D. Slama, and P. Russell, *Enterprise CORBA*: Prentice Hall, Março 1999.

[6] *Hypertext Transfer Protocol - HTTP/1.1,* RFC 2616, Junho de 1999.

[7] (11-05-2011). *Desktop Management Task Force*. Available: http://www.dmtf.org

[8] *Common Information Model (CIM) Infrastructure Specitication,* DSP0004, Outubro de 2005.

[9] DMTF. (02-05-2011). *Web-Based Enterprise Management*. Available: http://www.dmtf.org/standards/wbem

[10] *Representation of CIM in XML,* DSP0201, Setembro de 2009.

[11] *CIM Operations over HTTP,* DSP0200, Setembro de 2009.

[12] *Policy Based Management MIB,* RFC 4011, Março de 2005.

[13] *Overview of the 2002 IAB Network Management Workshop,* RFC 3535, Maio de 2003.

[14] *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER),* X.690, Setembro de 2002.

[15] (11-05-2011). *Juniper Networks*. Available: http://www.juniper.net/

[16] *NETCONF Configuration Protocol,* RFC 4741, Dezembro de 2006.

[17] *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF),* RFC 6020, Outubro de 2010.

[18] M. B. Jan Bergstra, *Handbook of Network and System Administration*: Elsevier, 2007.

[19] C. Huiyang, *et al.*, "Contrast Analysis of NETCONF Modeling Languages: XML Schema, Relax NG and YANG," in *Communication Software and Networks, 2009. ICCSN '09. International Conference on*, 2009, pp. 322-326.

[20] *Using NETCONF over the Simple Object Access Protocol (SOAP),* RFC 4743, Dezembro de 2006.

[21] (05-05-2011). *Eclipse - The Eclipse Foundation open source community website*. Available: http://www.eclipse.org/

[22] ITU-T. (20-04-2010). *Recommendation M.3400*. Available: http://www.itu.int/rec/T-REC-M.3400/en

[23] *The Common Management Information Services and Protocol over TCP/IP (CMOT),* RFC 1095, Abril de 1989.

[24] *Management Framework for Open Systems Interconnection (OSI) for CCITT Applications,* X. 700, 1993.

[25] *Structure and Identification of Management Information for TCP/IP-based Internets,* RFC 1155, Maio de 1990.

[26] *Structure of Management Information Version 2 (SMIv2),* RFC 2578, Abril de 1999.

[27] *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2),* RFC 1902, Janeiro de 2006.

[28] *An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks,* RFC 3411, Dezembro de 2002.

[29] U. AETHIS. (02-05-2011). *Security in SNMPv3 versus SNMPv1 or v2c*. Available: http://www.aethis.com/solutions/snmp_research/snmpv3_vs_wp.pdf

[30] *Introduction to Community-based SNMPv2,* RFC 1901, Janeiro de 2006.

[31] *User-based Security Model for SNMPv2,* RFC 1910, Fevereiro de 1996.

[32] J. Schonwalder, *et al.*, "SNMP Traffic Analysis: Approaches, Tools, and First Results," in *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, 2007, pp. 323-332.

[33] *Using the NETCONF Configuration Protocol over Secure Shell (SSH),* RFC 4742, Dezembro de 2006.

[34] *Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP),* RFC 4744, Dezembro de 2006.

[35] *NETCONF over Transport Layer Security (TLS),* RFC 5539, Maio de 2009.

[36] *Partial Lock Remote Procedure Call (RPC) for NETCONF,* RFC 5717, Dezembro de 2009.

[37] *NETCONF Event Notifications,* RFC 5277, Julho de 2008.

[38] *SMIng - Next Generation Structure of Management Information,* RFC 3780, Maio de 2004.

[39] (09-05-2011). *Yuma - YANG-based Unified Modular Automation Tools*. Available: http://sourceforge.net/projects/yuma/

[40] (02-04-2011). *Netopeer is a platform-neutral configuration system for routers and other devices*. Available: http://www.liberouter.org/netopeer/index.php

[41] M. M. James Clark. (Dezembro de 2001, 18-04-2010). *RELAX NG Specification*. Available: http://www.relaxng.org/spec-20011203.html

[42] R. S. B. Zores, and O. Festor. (02-05-2011). *YENCA*. Available: http://sourceforge.net/projects/yenca/

[43] (11-04-2011). *EnSuite Software Site*. Available: http://ensuite.sourceforge.net/yencap_user_doc.html

[44] G. Palmeri. (02-05-2011). *NETCONF4Android project*. Available: http://code.google.com/p/netconf4android/

[45] S. Bhushan, *et al.*, "NCClient: A Python Library for NETCONF Client Applications," presented at the Proceedings of the 9th IEEE International Workshop on IP Operations and Management, Venice, Italy, 2009.

[46] E. Nataf. (02-05-2011). *JYang (java Yang) a parser for the YANG data model*. Available: http://jyang.gforge.inria.fr/JYang_Home_Page.html

[47] (09-04-2011). *Pyang - An extensible YANG validator and converter in python*. Available: http://code.google.com/p/pyang/

[48] (09-05-2011). *Smidump – dump SMI or SPPI modules in various formats*. Available: http://www.ibr.cs.tu-bs.de/projects/libsmi/smidump.html

[49] (02-02-2011). *Xtext - Language Development Framework*. Available: http://www.eclipse.org/Xtext/

[50] (02-02-2011). *Eclipse Graphical Modeling Project*. Available: http://www.eclipse.org/modeling/gmp/

[51] (04-02-2010). *Web Tools Platform (WTP) Project*. Available: http://www.eclipse.org/webtools/

[52] (09-03-2011). *Eclipse Modeling Project*. Available: http://www.eclipse.org/modeling/

[53] (09-03-2010). *GEF (Graphical Editing Framework)*. Available: http://www.eclipse.org/gef/

[54] (15-12-2010). *Apache Tomcat Server*. Available: http://tomcat.apache.org/

[55] (15-12-2010). *Apache Axis 2*. Available: http://axis.apache.org/axis2/java/core/

[56] *Management Information Base for the Internet Protocol (IP),* RFC 4293, Abril de 2006.

[57] *Network Configuration Protocol Access Control Model,* draft-ietf-netconf-access-control-04, Junho de 2011.

ANEXOS

Ficheiro IP-MIB

```
IP-MIB DEFINITIONS ::= BEGIN

IMPORTS
   MODULE-IDENTITY, OBJECT-TYPE,
   Integer32, Counter32, IpAddress,
   mib-2, Unsigned32, Counter64,
   zeroDotZero                    FROM SNMPv2-SMI
   PhysAddress, TruthValue,
   TimeStamp, RowPointer,
   TEXTUAL-CONVENTION, TestAndIncr,
   RowStatus, StorageType         FROM SNMPv2-TC
   InetAddress, InetAddressType,
   InetAddressPrefixLength,
   InetVersion, InetZoneIndex     FROM INET-ADDRESS-MIB
   InterfaceIndex                 FROM IF-MIB;

ipMIB MODULE-IDENTITY
   LAST-UPDATED "200602020000Z"
   ORGANIZATION "IETF IPv6 MIB Revision Team"
   CONTACT-INFO
         "Editor:
          Shawn A. Routhier
          Interworking Labs
          108 Whispering Pines Dr. Suite 235
          Scotts Valley, CA 95066
          USA
          EMail: <sar@iwl.com>"
   DESCRIPTION
         "The MIB module for managing IP and ICMP implementations, but excluding their management of IP
routes.
          Copyright (C) The Internet Society (2006).  This version ofthis MIB module is part of RFC 4293; see
the RFC itself for full legal notices."

   REVISION      "200602020000Z"
   DESCRIPTION
         "The IP version neutral revision with added IPv6 objects for ND, default routers, and router
advertisements.  As well as  being the successor to RFC 2011, this MIB is also the successor to RFCs 2465
and 2466.  Published as RFC 4293."

   REVISION      "199411010000Z"
   DESCRIPTION
         "A separate MIB module (IP-MIB) for IP and ICMP management objects.  Published as RFC 2011."

   REVISION      "199103310000Z"
   DESCRIPTION
         "The initial revision of this MIB module was part of MIB-II, which was published as RFC 1213."
   ::= { mib-2 48}

--
-- The textual conventions we define and use in this MIB.
--
IpAddressOriginTC ::= TEXTUAL-CONVENTION
   STATUS     current
   DESCRIPTION
         "The origin of the address.
          manual(2) indicates that the address was manually configured to a specified address, e.g., by user
configuration.
          dhcp(4) indicates an address that was assigned to this system by a DHCP server.
          linklayer(5) indicates an address created by IPv6 stateless auto-configuration.
          random(6) indicates an address chosen by the system at random, e.g., an IPv4 address within
169.254/16, or an RFC 3041 privacy address."
   SYNTAX     INTEGER {
      other(1),
      manual(2),
      dhcp(4),
```

```
      linklayer(5),
      random(6)
   }

IpAddressStatusTC ::= TEXTUAL-CONVENTION
   STATUS     current
   DESCRIPTION
         "The status of an address.  Most of the states correspond to states from the IPv6 Stateless Address
Autoconfiguration protocol.
         The preferred(1) state indicates that this is a valid address that can appear as the destination or
source address of a packet.
         The deprecated(2) state indicates that this is a valid but deprecated address that should no longer
be used as a source address in new communications, but packets addressed to such an address are
processed as expected.
         The invalid(3) state indicates that this isn't a valid address and it shouldn't appear as the
destination or source address of a packet.
         The inaccessible(4) state indicates that the address is not accessible because the interface to which
this address is assigned is not operational.
         The unknown(5) state indicates that the status cannot be determined for some reason.
         The tentative(6) state indicates that the uniqueness of the address on the link is being verified.
Addresses in this state should not be used for general communication and should only be used to determine
the uniqueness of the address.
         The duplicate(7) state indicates the address has been determined to be non-unique on the link and
so must not be used.
         The optimistic(8) state indicates the address is available for use, subject to restrictions, while its
uniqueness on a link is being verified.
         In the absence of other information, an IPv4 address is always preferred(1)."
   REFERENCE "RFC 2462"
   SYNTAX     INTEGER {
      preferred(1),
      deprecated(2),
      invalid(3),
      inaccessible(4),
      unknown(5),
      tentative(6),
      duplicate(7),
      optimistic(8)
   }

--
-- the IP general group
-- some objects that affect all of IPv4
--

ip       OBJECT IDENTIFIER ::= { mib-2 4 }

ipForwarding OBJECT-TYPE
   SYNTAX     INTEGER {
                 forwarding(1),    -- acting as a router
                 notForwarding(2)  -- NOT acting as a router
              }
   MAX-ACCESS read-write
   STATUS     current
   DESCRIPTION
         "The indication of whether this entity is acting as an IPv4 router in respect to the forwarding of
datagrams received by, but not addressed to, this entity.  IPv4 routers forward datagrams.  IPv4 hosts do
not (except those source-routed via the host).
         When this object is written, the entity should save the change to non-volatile storage and restore
the object from non-volatile storage upon re-initialization of the system.
         Note: a stronger requirement is not used because this object was previously defined."
   ::= { ip 1 }

ipAddressEntry OBJECT-TYPE
   SYNTAX     IpAddressEntry
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
         "An address mapping for a particular interface."
   INDEX { ipAddressAddrType, ipAddressAddr }
```

```
   ::= { ipAddressTable 1 }

IpAddressEntry ::= SEQUENCE {
       ipAddressAddrType InetAddressType,
       ipAddressAddr InetAddress,
       ipAddressIfIndex InterfaceIndex,
       ipAddressType INTEGER,
       ipAddressPrefix RowPointer,
       ipAddressOrigin IpAddressOriginTC,
       ipAddressStatus IpAddressStatusTC,
       ipAddressCreated TimeStamp,
       ipAddressLastChanged TimeStamp,
       ipAddressRowStatus RowStatus,
       ipAddressStorageType StorageType
   }

ipAddressAddrType OBJECT-TYPE
   SYNTAX InetAddressType
   MAX-ACCESS not-accessible
   STATUS current
   DESCRIPTION
         "The address type of ipAddressAddr."
   ::= { ipAddressEntry 1 }

ipAddressAddr OBJECT-TYPE
   SYNTAX InetAddress
   MAX-ACCESS not-accessible
   STATUS current
   DESCRIPTION
         "The IP address to which this entry's addressing information pertains. The address type of this
object is specified in ipAddressAddrType.
          Implementors need to be aware that if the size of ipAddressAddr exceeds 116 octets, then OIDS of
instances of columns in this row will have more than 128 sub-identifiers and cannot be accessed using
SNMPv1, SNMPv2c, or SNMPv3."
   ::= { ipAddressEntry 2 }

ipAddressIfIndex OBJECT-TYPE
   SYNTAX InterfaceIndex
   MAX-ACCESS read-create
   STATUS current
   DESCRIPTION
         "The index value that uniquely identifies the interface to which this entry is applicable. The interface
identified by a particular value of this index is the same interface as identified by the same value of the IF-
MIB's ifIndex."
   ::= { ipAddressEntry 3 }

ipAddressType OBJECT-TYPE
   SYNTAX INTEGER {
              unicast(1),
              anycast(2),
              broadcast(3)
   }
   MAX-ACCESS read-create
   STATUS current
   DESCRIPTION
         "The type of address. broadcast(3) is not a valid value for IPv6 addresses (RFC 3513)."
   DEFVAL { unicast }
   ::= { ipAddressEntry 4 }

ipAddressPrefix OBJECT-TYPE
   SYNTAX RowPointer
   MAX-ACCESS read-only
   STATUS current
   DESCRIPTION
         "A pointer to the row in the prefix table to which this address belongs. May be { 0 0 } if there is no
such row."
   DEFVAL { zeroDotZero }
   ::= { ipAddressEntry 5 }
```

```
ipAddressOrigin OBJECT-TYPE
    SYNTAX IpAddressOriginTC
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
            "The origin of the address."
    ::= { ipAddressEntry 6 }

ipAddressStatus OBJECT-TYPE
    SYNTAX IpAddressStatusTC
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
            "The status of the address, describing if the address can be used for communication.
             In the absence of other information, an IPv4 address is always preferred(1)."
    DEFVAL { preferred }
    ::= { ipAddressEntry 7 }

ipAddressCreated OBJECT-TYPE
    SYNTAX TimeStamp
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
            "The value of sysUpTime at the time this entry was created.
             If this entry was created prior to the last re-initialization of the local network management
subsystem, then this object contains a zero value."
    ::= { ipAddressEntry 8 }

ipAddressLastChanged OBJECT-TYPE
    SYNTAX TimeStamp
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
            "The value of sysUpTime at the time this entry was last updated. If this entry was updated prior to
the last re-initialization of the local network management subsystem, then this object contains a zero
value."
    ::= { ipAddressEntry 9 }

ipAddressRowStatus OBJECT-TYPE
    SYNTAX RowStatus
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
            "The status of this conceptual row.
             The RowStatus TC requires that this DESCRIPTION clause states under which circumstances other
objects in this row can be modified. The value of this object has no effect on whether other objects in this
conceptual row can be modified.
             A conceptual row can not be made active until the ipAddressIfIndex has been set to a valid index."
    ::= { ipAddressEntry 10 }

ipAddressStorageType OBJECT-TYPE
    SYNTAX StorageType
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
            "The storage type for this conceptual row. If this object has a value of 'permanent', then no other
objects are required to be able to be modified."
    DEFVAL { volatile }
    ::= { ipAddressEntry 11 }

END
```

Ficheiro IP-MIB.yang

```
/*
 * This module has been generated by smidump 0.4.8:
 *
 *      smidump -f yang IP-MIB
 *
 * Do not edit. Edit the source file instead!
 */

module IP-MIB {

  /*** NAMESPACE / PREFIX DEFINITION ***/
  namespace "urn:ietf:params:xml:ns:yang:smiv2:IP-MIB";
  prefix "ip-mib";

  /*** LINKAGE (IMPORTS / INCLUDES) ***/
  import IF-MIB          { prefix "if-mib"; }
  import INET-ADDRESS-MIB { prefix "inet-address"; }
  import SNMPv2-TC       { prefix "smiv2"; }
  import inet-types      { prefix "inet"; }
  import yang-types      { prefix "yang"; }

  /*** META INFORMATION ***/
  organization
   "IETF IPv6 MIB Revision Team";

  contact
   "Editor:
    Shawn A. Routhier
    Interworking Labs
    108 Whispering Pines Dr. Suite 235
    Scotts Valley, CA 95066
    USA
    EMail: <sar@iwl.com>";

  description
   "The MIB module for managing IP and ICMP implementations, but excluding their management of IP routes.
    Copyright (C) The Internet Society (2006).  This version of this MIB module is part of RFC 4293; see the RFC itself for full legal notices.";

  revision "2006-02-02" {
    description
     "The IP version neutral revision with added IPv6 objects for ND, default routers, and router advertisements.  As well as being the successor to RFC 2011, this MIB is also the successor to RFCs 2465 and 2466.  Published as RFC 4293.";
  }
  revision "1994-11-01" {
    description
     "A separate MIB module (IP-MIB) for IP and ICMP management objects.  Published as RFC 2011.";
  }
  revision "1991-03-31" {
    description
     "The initial revision of this MIB module was part of MIB-II, which was published as RFC 1213.";
  }

  /*** TYPE DEFINITIONS ***/
  typedef IpAddressOriginTC {
```

```
    type enumeration {
      enum other     { value 1; }
      enum manual    { value 2; }
      enum dhcp      { value 4; }
      enum linklayer { value 5; }
      enum random    { value 6; }
    }
    description
     "The origin of the address.
      manual(2) indicates that the address was manually configured to a specified address, e.g., by user
configuration.
      dhcp(4) indicates an address that was assigned to this system by a DHCP server.
      linklayer(5) indicates an address created by IPv6 stateless auto-configuration.
      random(6) indicates an address chosen by the system at random, e.g., an IPv4 address within
169.254/16, or an RFC 3041 privacy address.";
   }

   typedef IpAddressStatusTC {
    type enumeration {
      enum preferred    { value 1; }
      enum deprecated   { value 2; }
      enum invalid      { value 3; }
      enum inaccessible { value 4; }
      enum unknown      { value 5; }
      enum tentative    { value 6; }
      enum duplicate    { value 7; }
      enum optimistic   { value 8; }
    }
    description
     "The status of an address.  Most of the states correspond to states from the IPv6 Stateless Address
Autoconfiguration protocol.
      The preferred(1) state indicates that this is a valid address that can appear as the destination or source
address of a packet.
      The deprecated(2) state indicates that this is a valid but deprecated address that should no longer be
used as a source address in new communications, but packets addressed to such an address are processed
as expected.
      The invalid(3) state indicates that this isn't a valid address and it shouldn't appear as the destination or
source address of a packet.
      The inaccessible(4) state indicates that the address is not accessible because the interface to which this
address is assigned is not operational.
      The unknown(5) state indicates that the status cannot be determined for some reason.
      The tentative(6) state indicates that the uniqueness of the address on the link is being verified.
Addresses in this state should not be used for general communication and should only be used to determine
the uniqueness of the address.
      The duplicate(7) state indicates the address has been determined to be non-unique on the link and so
must not be used.
      The optimistic(8) state indicates the address is available for use, subject to restrictions, while its
uniqueness on a link is being verified.
      In the absence of other information, an IPv4 address is always preferred(1).";
    reference
     "RFC 2462";
   }

   container ip {

    leaf ipForwarding {
      type enumeration {
        enum forwarding    { value 1; }
        enum notForwarding { value 2; }
      }
```

```
      config true;
      description
       "The indication of whether this entity is acting as an IPv4 router in respect to the forwarding of
datagrams received by, but not addressed to, this entity.  IPv4 routers forward datagrams.  IPv4 hosts do
not (except those source-routed via the host).
        When this object is written, the entity should save the change to non-volatile storage and restore the
object from non-volatile storage upon re-initialization of the system.
        Note: a stronger requirement is not used because this object was previously defined.";
    }

    /* XXX table comments here XXX */
    list ipAddressEntry {
      key "ipAddressAddrType ipAddressAddr";
      description
       "An address mapping for a particular interface.";
      leaf ipAddressAddrType {
        type inet-address:InetAddressType;
        config false;
        description
         "The address type of ipAddressAddr.";
      }
      leaf ipAddressAddr {
        type inet-address:InetAddress;
        config false;
        description
         "The IP address to which this entry's addressing information pertains.  The address type of this object
is specified in ipAddressAddrType.
          Implementors need to be aware that if the size of ipAddressAddr exceeds 116 octets, then OIDS of
instances of columns in this row will have more than 128 sub-identifiers and cannot be accessed using
SNMPv1, SNMPv2c, or SNMPv3.";
      }
      leaf ipAddressIfIndex {
        type if-mib:InterfaceIndex;
        config true;
        description
         "The index value that uniquely identifies the interface to which this entry is applicable.  The interface
identified by a particular value of this index is the same interface as identified by the same value of the IF-
MIB's ifIndex.";
      }

      leaf ipAddressType {
        type enumeration {
          enum unicast   { value 1; }
          enum anycast   { value 2; }
          enum broadcast { value 3; }
        }
        config true;
        description
         "The type of address.  broadcast(3) is not a valid value for IPv6 addresses (RFC 3513).";
      }

      leaf ipAddressPrefix {
        type smiv2:RowPointer;
        config false;
        description
         "A pointer to the row in the prefix table to which this address belongs.  May be { 0 0 } if there is no
such row.";
      }

      leaf ipAddressOrigin {
```

```
        type ip-mib:IpAddressOriginTC;
        config false;
        description
         "The origin of the address.";
      }

      leaf ipAddressStatus {
        type ip-mib:IpAddressStatusTC;
        config true;
        description
         "The status of the address, describing if the address can be used for communication.
          In the absence of other information, an IPv4 address is always preferred(1).";
      }

      leaf ipAddressCreated {
        type yang:timestamp;
        config false;
        description
         "The value of sysUpTime at the time this entry was created.
          If this entry was created prior to the last re-initialization of the local network management
subsystem, then this object contains a zero value.";
      }

      leaf ipAddressLastChanged {
        type yang:timestamp;
        config false;
        description
         "The value of sysUpTime at the time this entry was last updated.  If this entry was updated prior to
the last re-initialization of the local network management subsystem, then this object contains a zero
value.";
      }

      leaf ipAddressRowStatus {
        type smiv2:RowStatus;
        config true;
        description
         "The status of this conceptual row. The RowStatus TC requires that this DESCRIPTION clause states
under which circumstances other objects in this row can be modified.  The value of this object has no effect
on whether other objects in this conceptual row can be modified.
          A conceptual row can not be made active until the ipAddressIfIndex has been set to a valid index.";
      }

      leaf ipAddressStorageType {
        type smiv2:StorageType;
        config true;
        description
         "The storage type for this conceptual row.  If this object has a value of 'permanent', then no other
objects are required to be able to be modified.";
      }
    }
  }

} /* end of module IP-MIB */
```

Ficheiro IP-MIB.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:yin="urn:ietf:params:xml:schema:yang:yin:1"
           targetNamespace="urn:ietf:params:xml:ns:yang:smiv2:IP-MIB"
           xmlns="urn:ietf:params:xml:ns:yang:smiv2:IP-MIB"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified"
           version="2006-02-02"
           xml:lang="en"
           xmlns:if-mib="urn:ietf:params:xml:ns:yang:smiv2:IF-MIB"
           xmlns:smiv2="urn:ietf:params:xml:ns:yang:smiv2:SNMPv2-TC"
           xmlns:inet-address="urn:ietf:params:xml:ns:yang:smiv2:INET-ADDRESS-MIB"
           xmlns:ip-mib="urn:ietf:params:xml:ns:yang:smiv2:IP-MIB"
           xmlns:yang="urn:ietf:params:xml:ns:yang:yang-types"
           xmlns:inet="urn:ietf:params:xml:ns:yang:inet-types">

  <xs:import namespace="urn:ietf:params:xml:ns:yang:smiv2:IF-MIB"
             schemaLocation="IF-MIB.xsd"/>
  <xs:import namespace="urn:ietf:params:xml:ns:yang:smiv2:INET-ADDRESS-MIB"
             schemaLocation="INET-ADDRESS-MIB.xsd"/>
  <xs:import namespace="urn:ietf:params:xml:ns:yang:smiv2:SNMPv2-TC"
             schemaLocation="SNMPv2-TC.xsd"/>
  <xs:import namespace="urn:ietf:params:xml:ns:yang:inet-types"
             schemaLocation="inet-types.xsd"/>
  <xs:import namespace="urn:ietf:params:xml:ns:yang:yang-types"
             schemaLocation="yang-types.xsd"/>

  <xs:annotation>
    <xs:documentation>
      This schema was generated from the YANG module IP-MIB
      by pyang version 1.0.

      The schema describes an instance document consisting of the entire configuration data store,
operational data, rpc operations, and notifications.
      This schema can thus NOT be used as-is to validate NETCONF PDUs.
    </xs:documentation>
  </xs:annotation>

  <xs:annotation>
    <xs:documentation>
      The MIB module for managing IP and ICMP implementations, but excluding their management of IP
routes.
      Copyright (C) The Internet Society (2006).  This version of this MIB module is part of RFC 4293; see the
RFC itself for full legal notices.
    </xs:documentation>
  </xs:annotation>

  <!-- YANG typedefs -->

  <xs:simpleType name="IpAddressOriginTC">
    <xs:annotation>
      <xs:documentation>
        The origin of the address.
        manual(2) indicates that the address was manually configured to a specified address, e.g., by user
configuration.
        dhcp(4) indicates an address that was assigned to this system by a DHCP server.
        linklayer(5) indicates an address created by IPv6 stateless auto-configuration.
```

```
      random(6) indicates an address chosen by the system at random, e.g., an IPv4 address within
169.254/16, or an RFC 3041 privacy address.
     </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:string">
     <xs:enumeration value="other"/>
     <xs:enumeration value="manual"/>
     <xs:enumeration value="dhcp"/>
     <xs:enumeration value="linklayer"/>
     <xs:enumeration value="random"/>
    </xs:restriction>
   </xs:simpleType>

   <xs:simpleType name="IpAddressStatusTC">
    <xs:annotation>
     <xs:documentation>
      The status of an address.  Most of the states correspond to states from the IPv6 Stateless Address
Autoconfiguration protocol.
      The preferred(1) state indicates that this is a valid address that can appear as the destination or
source address of a packet.
      The deprecated(2) state indicates that this is a valid but deprecated address that should no longer be
used as a source address in new communications, but packets addressed to such an address are processed
as expected.
      The invalid(3) state indicates that this isn't a valid address and it shouldn't appear as the destination
or source address of a packet.
      The inaccessible(4) state indicates that the address is not accessible because the interface to which
this address is assigned is not operational.
      The unknown(5) state indicates that the status cannot be determined for some reason.
      The tentative(6) state indicates that the uniqueness of the address on the link is being verified.
Addresses in this state should not be used for general communication and should only be used to determine
the uniqueness of the address.
      The duplicate(7) state indicates the address has been determined to be non-unique on the link and so
must not be used.
      The optimistic(8) state indicates the address is available for use, subject to restrictions, while its
uniqueness on a link is being verified.
      In the absence of other information, an IPv4 address is always preferred(1).
     </xs:documentation>
    </xs:annotation>

    <xs:restriction base="xs:string">
     <xs:enumeration value="preferred"/>
     <xs:enumeration value="deprecated"/>
     <xs:enumeration value="invalid"/>
     <xs:enumeration value="inaccessible"/>
     <xs:enumeration value="unknown"/>
     <xs:enumeration value="tentative"/>
     <xs:enumeration value="duplicate"/>
     <xs:enumeration value="optimistic"/>
    </xs:restriction>
   </xs:simpleType>

   <xs:element name="ip">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="ipForwarding" minOccurs="0">
       <xs:annotation>
        <xs:documentation>
```

```
      The indication of whether this entity is acting as an IPv4 router in respect to the forwarding of
      datagrams received by, but not addressed to, this entity.  IPv4 routers forward datagrams.  IPv4
      hosts do not (except those source-routed via the host).
      When this object is written, the entity should save the change to non-volatile storage and restore
      the object from non-volatile storage upon re-initialization of the system.
      Note: a stronger requirement is not used because this object was previously defined.
    </xs:documentation>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="forwarding"/>
      <xs:enumeration value="notForwarding"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="ipAddressEntry" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation>
      An address mapping for a particular interface.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ipAddressAddrType"  type="inet-address:InetAddressType">
        <xs:annotation>
          <xs:documentation>
            The address type of ipAddressAddr.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="ipAddressAddr"  type="inet-address:InetAddress">
        <xs:annotation>
          <xs:documentation>
            The IP address to which this entry's addressing information pertains.  The address type of this
      object is specified in ipAddressAddrType.
            Implementors need to be aware that if the size of ipAddressAddr exceeds 116 octets, then
      OIDS of instances of columns in this row will have more than 128 sub-identifiers and cannot be
      accessed using SNMPv1, SNMPv2c, or SNMPv3.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="ipAddressIfIndex" minOccurs="0"  type="if-mib:InterfaceIndex">
        <xs:annotation>
          <xs:documentation>
            The index value that uniquely identifies the interface to which this entry is applicable. The
      interface identified by a particular value of this index is the same interface as identified by the
      same value of the IF-MIB's ifIndex.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="ipAddressType" minOccurs="0">
        <xs:annotation>
          <xs:documentation>
            The type of address.  broadcast(3) is not a valid value for IPv6 addresses (RFC 3513).
          </xs:documentation>
        </xs:annotation>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="unicast"/>
            <xs:enumeration value="anycast"/>
```

```
      <xs:enumeration value="broadcast"/>
    </xs:restriction>
  </xs:simpleType>
 </xs:element>
 <xs:element name="ipAddressPrefix" minOccurs="0"  type="smiv2:RowPointer">
  <xs:annotation>
   <xs:documentation>
    A pointer to the row in the prefix table to which this address belongs.  May be { 0 0 } if there
is no such row.
   </xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="ipAddressOrigin" minOccurs="0"  type="ip-mib:IpAddressOriginTC">
  <xs:annotation>
   <xs:documentation>
    The origin of the address.
   </xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="ipAddressStatus" minOccurs="0"  type="ip-mib:IpAddressStatusTC">
  <xs:annotation>
   <xs:documentation>
    The status of the address, describing if the address can be used for communication.
    In the absence of other information, an IPv4 address is always preferred(1).
   </xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="ipAddressCreated" minOccurs="0"  type="yang:timestamp">
  <xs:annotation>
   <xs:documentation>
        The value of sysUpTime at the time this entry was created.
        If this entry was created prior to the last re-initialization of the local network management
        subsystem, then this object contains a zero value.
   </xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="ipAddressLastChanged" minOccurs="0"  type="yang:timestamp">
  <xs:annotation>
   <xs:documentation>
    The value of sysUpTime at the time this entry was last updated.  If this entry was updated
prior to the last re-initialization of the local network management subsystem, then this object
contains a zero value.
   </xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="ipAddressRowStatus" minOccurs="0"  type="smiv2:RowStatus">
  <xs:annotation>
   <xs:documentation>
    The status of this conceptual row.
    The RowStatus TC requires that this DESCRIPTION clause states under which circumstances
other objects in this row can be modified.  The value of this object has no effect on whether other
objects in this conceptual row can be modified.
    A conceptual row can not be made active until the ipAddressIfIndex has been set to a valid
index.
   </xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="ipAddressStorageType" minOccurs="0"  type="smiv2:StorageType">
  <xs:annotation>
   <xs:documentation>
```

```
                The storage type for this conceptual row.  If this object has a value of 'permanent', then no
            other objects are required to be able to be modified.
                </xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:any minOccurs="0" maxOccurs="unbounded"
                  namespace="##other" processContents="lax"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:any minOccurs="0" maxOccurs="unbounded"
            namespace="##other" processContents="lax"/>
    </xs:sequence>
  </xs:complexType>
    <xs:key name="key_ip_ipAddressEntry">
      <xs:selector xpath="ip-mib:ipAddressEntry"/>
      <xs:field xpath="ip-mib:ipAddressAddrType"/>
      <xs:field xpath="ip-mib:ipAddressAddr"/>
    </xs:key>
 </xs:element>

</xs:schema>
```

Ficheiro AgenteNetconf.java

```
package mynetconfservice;

import inet_address_mib.smiv2.yang.ns.xml.params.ietf.InetAddress;
import ip_mib.smiv2.yang.ns.xml.params.ietf.IpDocument;
import ip_mib.smiv2.yang.ns.xml.params.ietf.IpDocument.Ip.IpAddressEntry;

import java.net.UnknownHostException;
import java.rmi.RemoteException;
import org.apache.axis2.AxisFault;
import org.apache.xmlbeans.XmlObject;
import snmpv2_tc.smiv2.yang.ns.xml.params.ietf.DisplayString;
import _0._1.base.netconf.ns.xml.params.ietf.ConfigInlineType;
import _0._1.base.netconf.ns.xml.params.ietf.EditConfigType;
import _0._1.base.netconf.ns.xml.params.ietf.GetConfigType;
import _0._1.base.netconf.ns.xml.params.ietf.HelloDocument;
import _0._1.base.netconf.ns.xml.params.ietf.RpcDocument;
import _0._1.base.netconf.ns.xml.params.ietf.RpcType;

public class ClientMain {

	@SuppressWarnings("deprecation")
	public static void main(String[] args) {

	try {
		NetconfStub stub = new
			NetconfStub("http://localhost:8080/WebServiceProject/services/netconf");

		//SEND HELLO
		HelloDocument hellodoc = HelloDocument.Factory.newInstance();
		HelloDocument.Hello hello = hellodoc.addNewHello();
		HelloDocument.Hello.Capabilities cap = hello.addNewCapabilities();
		cap.addCapability("urn:ietf:params:netconf:base:1.0");
		hello.setCapabilities(cap);
		hellodoc.setHello(hello);
		hellodoc = stub.hello(hellodoc);

		//SEND RPC WITH EDIT-CONFIG
		RpcDocument rpc = RpcDocument.Factory.newInstance();
		RpcType rpctype = rpc.addNewRpc();
		EditConfigType editconfigtype = EditConfigType.Factory.newInstance();

		IpDocument ipdoc = IpDocument.Factory.newInstance();
		IpDocument.Ip ip = ipdoc.addNewIp();
		IpAddressEntry ipaddr = ip.addNewIpAddressEntry();
		DisplayString str = DisplayString.Factory.newInstance();
		str.setStringValue("192.168.1.1");
		ipaddr.set(str);

		ConfigInlineType config = editconfigtype.addNewConfig();
		config.set(ipdoc);
		editconfigtype.setConfig(config);

		rpctype.setRpcOperation(editconfigtype);
		rpctype.setMessageId("101");
		rpc.setRpc(rpctype);
		stub.rpc(rpc);

		//SEND RPC WITH GET-CONFIG
		RpcDocument rpc2 = RpcDocument.Factory.newInstance();
		RpcType rpctype2 = rpc2.addNewRpc();
		GetConfigType getconfigtype = GetConfigType.Factory.newInstance();
		rpctype2.setRpcOperation(getconfigtype);
		rpctype2.setMessageId("102");
		rpc2.setRpc(rpctype2);
		stub.rpc(rpc2);
```

```
                System.out.println("Finished!!!");

        } catch (AxisFault e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        } catch (RemoteException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

        }
}
```

Ficheiro NetconfSkeleton.java

```
/**
 * NetconfSkeleton.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis2 version: 1.5.4  Built on : Dec 19, 2010 (08:18:42 CET)
 */
package mynetconfservice;

import java.io.File;
import java.io.IOException;
import org.apache.xmlbeans.XmlException;
import org.apache.xmlbeans.XmlObject;
import _0._1.base.netconf.ns.xml.params.ietf.ConfigInlineType;
import _0._1.base.netconf.ns.xml.params.ietf.DataInlineType;
import _0._1.base.netconf.ns.xml.params.ietf.EditConfigType;
import _0._1.base.netconf.ns.xml.params.ietf.GetConfigType;
import _0._1.base.netconf.ns.xml.params.ietf.HelloDocument;
import _0._1.base.netconf.ns.xml.params.ietf.RpcOperationType;
import _0._1.base.netconf.ns.xml.params.ietf.RpcReplyDocument;
import _0._1.base.netconf.ns.xml.params.ietf.RpcReplyType;
import _0._1.base.netconf.ns.xml.params.ietf.RpcType;

   /**
    *  NetconfSkeleton java skeleton for the axisService
    */
   public class NetconfSkeleton{
      /**
       * Auto generated method signature
       *
       * @param rpc
       */
       public _0._1.base.netconf.ns.xml.params.ietf.RpcReplyDocument rpc
              (_0._1.base.netconf.ns.xml.params.ietf.RpcDocument rpc) {
                XmlObject ok = XmlObject.Factory.newInstance();
                ok.setNil();
                String filePath = "/config/configdata.txt";
                File file = new File(filePath);
                RpcType rpctype = rpc.getRpc();

                RpcReplyDocument rpcreply = RpcReplyDocument.Factory.newInstance();
                RpcReplyType rpcreplytype = rpcreply.addNewRpcReply();
                RpcOperationType op = rpctype.getRpcOperation();

                if (op instanceof EditConfigType){
                    EditConfigType ed = (EditConfigType) op;
                    ConfigInlineType config = ConfigInlineType.Factory.newInstance();
                    config.set(ed.getConfig());

                    try {
                              config.save(file);
                    } catch (IOException e) {
                              // TODO Auto-generated catch block
                              e.printStackTrace();
                    }
                    rpcreplytype.setOk(ok);
                }

                if ((op instanceof GetConfigType) ){
                     File inputXMLFile = new File(filePath);
                     DataInlineType data;
                    try {
                              data = DataInlineType.Factory.parse(inputXMLFile);
                              rpcreplytype.setData(data);
                    } catch (XmlException e) {
```

```
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

            else{
                rpcreplytype.setOk(ok);
            }

            rpcreplytype.setMessageId(rpctype.getMessageId());
            rpcreply.setRpcReply(rpcreplytype);
            return rpcreply;
    }


    /**
     * Auto generated method signature
     *
     * @param hello
     */
    public _0._1.base.netconf.ns.xml.params.ietf.HelloDocument hello
          (_0._1.base.netconf.ns.xml.params.ietf.HelloDocument hello)
    {
                HelloDocument hellodoc = HelloDocument.Factory.newInstance();
                HelloDocument.Hello hello2 = hellodoc.addNewHello();
                HelloDocument.Hello.Capabilities cap = hello2.addNewCapabilities();
                cap.addCapability("urn:ietf:params:netconf:base:1.0");
                hello2.setCapabilities(cap);
                hello2.setSessionId(4);
                hellodoc.setHello(hello2);
                System.out.println("hello");
                return hellodoc;
    }

}
```