



**Universidad de la República**  
Facultad de Ingeniería  
Instituto de Computación  
**Uruguay**

---

# **Ingeniería Dirigida por Modelos aplicada a la configuración de redes de computadoras**

Marco Centurión  
Maximiliano Kotvinsky

---

Proyecto de Grado  
Ingeniería en Computación  
Universidad de la República

Montevideo, Uruguay, Febrero de 2019

Supervisores:      Dr. Ing. Daniel Calegari  
                             Dr. Ing. Andrea Delgado  
                             Universidad de la República



El manejo de la configuración en el contexto de las tareas de administración de sistemas es una actividad que se ha vuelto esencial en los últimos años y permite indicar, muchas veces de forma puramente declarativa, el estado que se desea de un sistema en particular. El paradigma de Ingeniería Dirigida por Modelos (MDE por sus siglas en inglés) propone la construcción de software basado en una abstracción de su complejidad a través de la definición de modelos y en un proceso de construcción (semi)automático guiado por transformaciones de estos modelos.

En esta tesis se estudia la posibilidad de utilizar el paradigma de MDE para buscar una solución al problema planteado. Para hacer esto posible, se cuenta con un modelo de representación de la estructura de red y de sus componentes, así como de componentes de software relacionados, e incluyendo parámetros de configuración de los mismos. A su vez, se busca cierta automatización que permita generar scripts de configuración a partir de necesidades expresadas en este modelo de visualización a más alto nivel de abstracción.

Se comienza con un estudio del estado del arte, mencionando las características similares proporcionadas por herramientas existentes, donde se puede ver aproximaciones a las diferentes características por separado, existiendo varias herramientas de generación y simulación de redes por un lado, y de modelado de topologías por otro, pero ninguna que posea ambas, es decir, la capacidad de modelar una topología deseada, y asistir en el manejo de la configuración en dicha topología.

A partir de esto, se opta por definir un perfil UML con el fin de extender el lenguaje con el conjunto de componentes necesarios. La solución descrita en dicho perfil se conforma por componentes físicos necesarios para la configuración de una topología, como lo son routers, switches, computadoras, servidores, y periféricos; componentes lógicos como sistemas operativos y una selección de componentes de software; y finalmente el componente de configuración, que es la base del valor adicional aportado por este trabajo.

Una vez conformado el perfil UML, se trabaja sobre las transformaciones de modelo a texto, que en este caso serán archivos de configuración para el sistema en cuestión. Dicha configuración se realizará utilizando como soporte sistemas de manejo de la configuración (Configuration Manager Tools), como lo son Puppet, Chef o Ansible, por nombrar algunos ejemplos.

Por último se analiza un caso de estudio para mostrar la viabilidad y utilidad de la solución planteada, utilizando como partida el modelo de una topología tipo, con los diferentes componentes definidos, y obteniendo un conjunto de scripts de configuración aplicables a la herramienta de manejo de la configuración.

Las conclusiones de la tesis consisten en un conjunto de fortalezas y debilidades, destacando que la solución brinda un punto de entrada al manejo de la configuración utilizando herramientas de MDE, además de permitir una solución a la administración de sistema de red, brindado a través del perfil UML. Al mismo tiempo, es un punto de partida para que futuros trabajos puedan extender los componentes definidos en este trabajo, así ofreciendo mayor libertad al momento de administrar un sistema.

**Palabras clave:** Model driven engineering, UML, UML profile, Configuration Manager, Eclipse, Papyrus, Acceleo, Puppet, Chef, Ansible, network, metamodel, network configuration, model to text.

# Contenido

<b>1</b>	<b>Introducción</b>	<b>1</b>
<b>2</b>	<b>Marco Teórico</b>	<b>3</b>
2.1	Estado del Arte	3
2.1.1	Herramientas de simulación de redes de computadoras	3
2.1.2	Herramientas de generación de topologías de red	4
2.1.3	Herramientas de visualización de redes de computadoras	4
2.2	Sistemas de Manejo de la Configuración	5
2.2.1	Puppet	6
2.2.2	Chef	6
2.2.3	Ansible	8
2.3	Model Driven Engineering	8
2.3.1	Modelo	10
2.3.2	Metamodelo	10
2.3.3	Transformación	11
2.3.4	Diagramas de deployment	12
2.3.5	Perfiles UML	14
2.4	Herramientas de MDE	16
2.4.1	Eclipse Modeling Framework	16
2.4.2	Papyrus	16
2.4.3	Acceleo	18
2.4.4	ATL	18
2.5	MDE en redes de computadoras	19
2.5.1	MDE en sensores de red inalámbricos	19
2.5.2	YANG	19
2.5.3	Metamodelo para Configuraciones en Dispositivos de Redes	19
<b>3</b>	<b>Análisis de requerimientos</b>	<b>23</b>
3.1	Requerimientos del Proyecto	23
3.1.1	RQ1 - Lenguaje de especificación de la estructura de red	23
3.1.2	RQ2 - Estandarización de modelos creados	24
3.1.3	RQ3 - Transformación de modelo a una configuración	24
3.1.4	RQ4 - Aplicación de configuración generada	24
3.1.5	RQ5 - Detección de red existente	24
3.1.6	RQ6 - Detección de cambios en la red	25

3.1.7	RQ7 - Distribución de archivos generados entre los nodos . . . . .	25
3.2	Requerimientos del Lenguaje . . . . .	25
3.3	Requerimientos de la Herramienta . . . . .	26
3.4	Alcance del Proyecto . . . . .	27
<b>4</b>	<b>Diseño de la Solución</b>	<b>29</b>
4.1	Definición de Dominio Específico . . . . .	29
4.1.1	Nodos Físicos . . . . .	30
4.1.2	Nodos Lógicos . . . . .	30
4.1.3	Configuración . . . . .	31
4.2	Lenguaje de Modelado del Dominio . . . . .	32
4.2.1	Lenguaje a Utilizar . . . . .	32
4.3	CMT a Utilizar . . . . .	33
4.4	Transformación M2T . . . . .	34
<b>5</b>	<b>Implementación de la Solución</b>	<b>37</b>
5.1	Implementación del modelo UML . . . . .	37
5.1.1	Modelado de la Topología . . . . .	38
5.1.2	Modelado de la Configuración . . . . .	41
5.1.3	Modelado de la Solución . . . . .	42
5.2	Implementación del Perfil UML . . . . .	43
5.2.1	Nodos Físicos . . . . .	44
5.2.2	Nodos Lógicos . . . . .	44
5.2.3	Configuración . . . . .	46
5.2.4	Restricciones . . . . .	46
5.2.5	Implementación del Perfil como plugin . . . . .	47
5.3	Implementación de la Transformación . . . . .	49
5.3.1	Generación de Información . . . . .	52
5.3.2	Generación de Scripts . . . . .	53
5.3.3	Resumen de transformaciones . . . . .	60
<b>6</b>	<b>Caso de Estudio</b>	<b>63</b>
<b>7</b>	<b>Conclusiones</b>	<b>73</b>
7.1	Fortalezas . . . . .	73
7.2	Debilidades . . . . .	74
7.3	Trabajo Futuro . . . . .	74

# 1

## Introducción

---

La configuración de una red de computadoras es el proceso mediante el cual se busca que los elementos de una red (switches, routers, PCs, hasta impresoras) tengan un estado determinado. El significado de estado puede variar dependiendo del dispositivo y del objetivo de los administradores, pudiendo ir desde una simple configuración de red hasta asegurar que un programa esté instalado o un archivo de configuración esté presente. Todo esto es un proceso costoso y proclive a errores debido a la continua evolución y diversidad de componentes conectados a la red.

Este proceso de configuración se suele realizar a través de un proceso de codificación de scripts a bajo nivel que se torna repetitivo y, sin protocolos consensuados entre los integrantes del grupo de administración de dicha red, muchas veces caótico.

Algunas herramientas permiten visualizar información general de una red (ej.: LanFlow, Network Notepad y Cade) y de esta forma lograr una comprensión global de su estructura y componentes, pero esto no soluciona los problemas inherentes a impactar una configuración de forma manual, por lo cual si bien se tiene una mejor comprensión de la red, el proceso sigue siendo repetitivo y caótico.

Una potencial solución a estos problemas es contar con un modelo que represente simultáneamente la estructura de la red, sus componentes y la configuración de los mismos. Esto sería acompañado de cierta automatización que permita generar scripts de configuración a partir de necesidades expresadas en el modelo de visualización a más alto nivel de abstracción. Con esta solución no sólo se soluciona el problema de comprensión global sino que simplifica la comunicación entre los integrantes del grupo y finalmente se elimina la necesidad de generar scripts de configuración manualmente.

Teniendo en cuenta esta solución potencial, la Ingeniería Dirigida por Modelos (MDE) parece ser una excelente herramienta para lograr implementarla. MDE propone partir de una abstracción del problema (un modelo del problema) y mediante sucesivas transformaciones construir

el producto de software deseado. Para lograr la abstracción necesaria es fundamental contar con un lenguaje de modelado apropiado que permita representar los elementos involucrados.

El objetivo de este proyecto consistirá entonces, en la definición de un lenguaje de modelado de la estructura de la red y de sus componentes, así como la generación automática de los scripts de configuración de la red. En particular, los objetivos específicos que se desean analizar son los siguientes:

- Definir o adaptar un lenguaje de modelado de aspectos de una red de computadoras.
- Definir transformaciones de modelos para la generación automática de scripts de configuración.
- Desarrollar un prototipo funcional de herramienta de configuración capaz de integrar el lenguaje y transformaciones definidos.
- Desarrollar un caso de estudio que permita poner en práctica las definiciones realizadas y evidenciar la viabilidad técnica de la solución a través del uso del prototipo desarrollado.

El documento se organizará de la siguiente manera, como primer paso se realizará un Marco Teórico en el Capítulo 2 con información de las herramientas actuales y de interés para el desarrollo del proyecto. Luego se procederá al Análisis de Requerimientos en el Capítulo 3, donde se plantean que características deben cumplir tanto el lenguaje que se utilizará, como la herramienta de generación de scripts de configuración. Una vez definidos los requerimientos, en el Capítulo 4 se llevará a cabo el Diseño de la Solución, que consiste en detallar a más bajo nivel la solución propuesta, y brindar una respuesta a alto nivel de como se llevará a cabo, así como seleccionar las tecnologías a utilizar para resolver el problema. El Capítulo 5 describirá la Implementación de la Solución, esto consiste en mostrar como se plasman las soluciones de la sección anterior desde el punto de vista tecnológico. Finalmente se presentarán Casos de Estudio, y Conclusiones obtenidas del proyecto en los Capítulos 6 y 7 respectivamente.



# 2

En este capítulo se hará una revisión de conceptos básicos de la ingeniería dirigida por modelos, perfiles UML y diagramas de deployment. Se introducirá el concepto de sistema de manejo de la configuración, junto con algunos ejemplos de los mismos, y se mostrarán ejemplos de herramientas de simulación y visualización de redes, relacionadas a la temática. Finalmente se encontrará una breve introducción a herramientas de modelado, en particular Eclipse Modeling Tools, y algunos ejemplos de MDE aplicado a redes de computadoras.

## 2.1 Estado del Arte

### 2.1.1 Herramientas de simulación de redes de computadoras

#### M5 Simulator

La herramienta de código abierto M5 Simulator es utilizado para permitir la investigación en redes TCP/IP. Provee las características necesarias para simular redes de hosts, incluyendo todas las funcionalidades de un sistema, un detallado sistema de I/O, y la capacidad de simular múltiples sistemas en red. Se describe como un simulador de arquitecturas de propósito general, con un enfoque hacia networking. [Binkert et al., 2006]

Si bien esta herramienta es capaz de simular redes, no provee un editor gráfico para realizar diagramas de las mismas, lo mismo sucede con las herramientas de generación de topologías que se verán a continuación.

### 2.1.2 Herramientas de generación de topologías de red

Existen varias herramientas para la generación de topologías de red, que permiten hacer simulaciones sobre las topologías generadas. Estas herramientas, sin embargo, están enfocadas en la estructura de la topología, y por lo tanto, se basan en la generación de nodos y su estructura a un alto nivel, por lo que no son utilizadas para diferenciar dispositivos de red en particular. Hay diferentes clases de generadores, en base a la forma en que generan los grafos, dos de los más conocidos son los siguientes [Tangmunarunkit et al., 2002]:

**Tiers:** es un generador *estructural*, esta herramienta se enfoca en generar topologías multi-nivel (multi-tier), e implementa simulación de modelos que imitan la estructura de red de internet.

**Waxman:** este generador es de la clase de *grafos aleatorios*, genera grafos aleatorios en base al modelo de Erdos-Renyi, que asigna una probabilidad uniforme para crear una conexión entre cada par de nodos. Waxman extiende este modelo asignando los nodos, de forma aleatoria, a posiciones en un plano, y basando la probabilidad de creación de conexión entre un par nodos en la distancia que existe entre ellos.

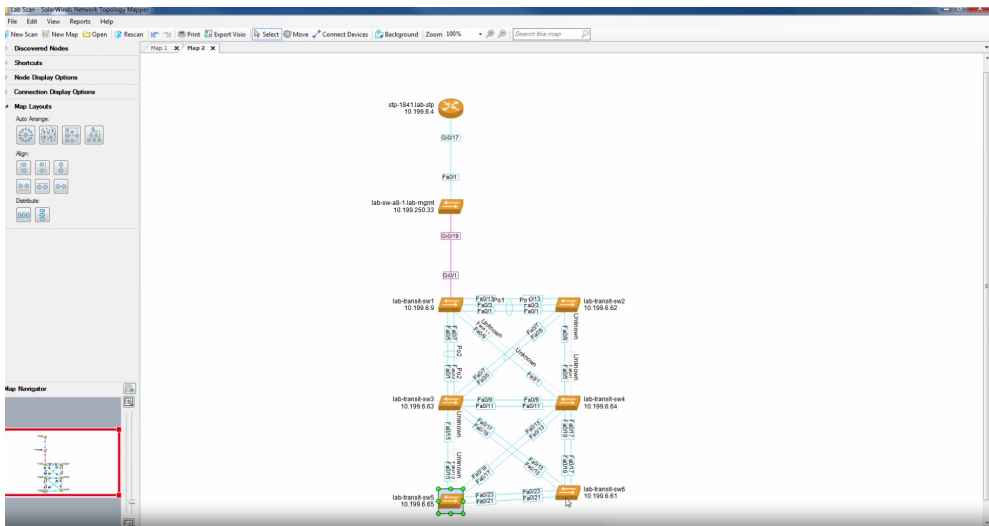
### 2.1.3 Herramientas de visualización de redes de computadoras

**SolarWinds: Network Topology Mapper** [SolarWinds, 2019]

SolarWinds es una herramienta utilizada para escanear y mapear la infraestructura de la red, incluyendo dispositivos de red, servidores, y hosts virtualizados. Algunas de sus características principales son:

- La capacidad de detectar y delinear automáticamente la topología de la red, y generar diagramas de red completos.
- Ofrece la posibilidad de elegir diferentes métodos de detección.
- Permite exportar los diagramas en diferentes formatos.
- Brinda informes robustos sobre puertos de conmutación, VLAN, subredes e inventario.
- Aborda directamente el cumplimiento con PCI, FIPS 140-2, etc. que requieren mantenimiento de un diagrama de red actualizado.

En la Figura 2.1 se puede ver un ejemplo de un diagrama de red creado utilizando la herramienta. Es importante notar que esta herramienta se basa en la detección y generación de diagramas de red, sin embargo, no ofrece la posibilidad de generar archivos de configuración a partir de estos diagramas.



## 2.2 Sistemas de Manejo de la Configuración

En cualquier contexto las computadoras cambian continuamente, software es agregado, removido, o actualizado, y sus configuraciones cambian constantemente, lo que se vuelve un problema al tener que manejar decenas o cientos de computadoras al mismo tiempo. De este problema surgen los Sistemas de Manejo de la Configuración.

Existen diferentes herramientas pero todas comparten ciertas características fundamentales [Cotton, 2016]:

**Aplicación (Enforcement) de configuración:** Es una de las características más importantes, dado que asegura que una computadora siempre este configurada según su especificación, es decir, que se encuentre en su estado deseado en cada momento dado.

Esto significa que el manejador evitará que se produzcan desvíos en la configuración por medio de actualizaciones, o cambios aplicados por terceros (técnicos, trabajadores).

**Permite la cooperación:** Facilitan el trabajo en equipo al tener un solo punto de acceso a toda la configuración. Esto se hace indispensable a medida que la cantidad de computadoras a configurar aumenta, dado que realizar la configuración de cada una será propenso a fallos, y al mismo tiempo, detectarlos se volverá más dificultoso.

**Permite versionado:** La mejor manera de permitir la colaboración en equipo es utilizando un sistema de versionado, una ventaja es que la mayoría de las herramientas en cuestión basan su configuración en archivos de texto, lo que lleva a que la integración con

cualquier sistema de versionado sea inmediata, y obteniendo todos sus beneficios ya conocidos.

**Permite control de cambios:** Dado que estas herramientas se basan en archivos de texto, y pueden ser versionadas, también se puede realizar sobre ellas un sistema de control de cambios, donde los usuarios envían los cambios a aplicar, se comparan las diferencias, y se lleva a cabo revisión de código antes de aprobar los mismos. Esto también permite llevar registro de diferentes versiones, en caso de necesitar volver a un estado previo.

**Permite la abstracción de los sistemas:** Al tener cierta cantidad de computadoras que mantener, es probable que se tenga que gestionar diferentes sistemas operativos, o al menos, diferentes distribuciones de un mismo sistema. Con herramientas de manejo de la configuración esto deja de ser un problema, ya que se encargarán de abstraer la implementación de configuraciones específicas de los diferentes sistemas o distribuciones.

En las siguientes secciones se presentará una selección de herramientas para el manejo de la configuración. En particular se hará una breve introducción a Puppet, Chef, y Ansible. La elección se basa en que dichas herramientas son las más populares en el área, son de fácil acceso, y poseen un buen nivel de documentación. Además, estas herramientas poseen características similares entre sí, lo que será beneficioso al momento de realizar la implementación, o como consideración para trabajos futuros.

### 2.2.1 Puppet

Puppet es una herramienta de automatización de configuración de hosts y servidores. Utiliza un lenguaje propio, declarativo, para definir el estado deseado de los distintos dispositivos. En sus mejores prácticas utiliza diferentes modelos (rol, perfil, módulo), lo que confiere una gran flexibilidad al momento de configurar nodos.

### 2.2.2 Chef

Según su propia definición, Chef es una plataforma de automatización que transforma infraestructura en código. Independientemente de si se opera en servicio Cloud, con servidores locales, o un híbrido, Chef permite automatizar como la infraestructura es configurada, desplegada, y manejada a través de una red, sin importar el tamaño. [Chef, 2019]

En la Figura 2.3 se muestran los diferentes elementos que participan en un ambiente de Chef, incluyendo los nodos, servidores, y estaciones de trabajo. Por otro lado, en la Figura 2.4 se puede ver como se representa el manejo de la configuración en Chef.

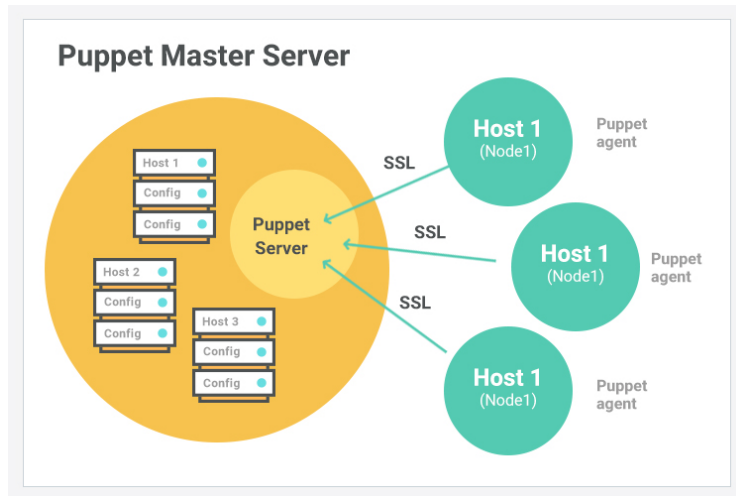


Figure 2.2: Representación del manejo de la configuración a través de Puppet.[Yigal, 2017]

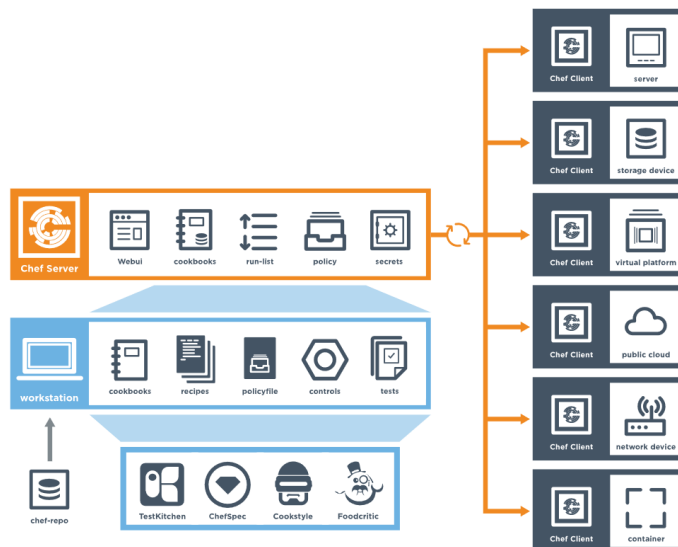


Figure 2.3: Elementos presentes en el ambiente de Chef. [Chef, 2019]

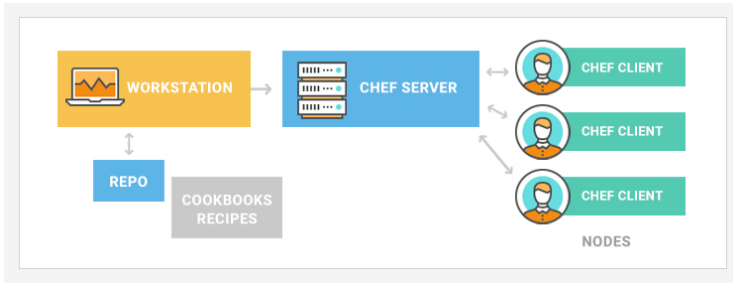


Figure 2.4: Representación del manejo de la configuración en Chef. [Yigal, 2017]

### 2.2.3 Ansible

Ansible es una herramienta de automatización de IT que permite automatizar diferentes necesidades como aprovisionamiento en servicios cloud, manejo de la configuración, despliegue de aplicaciones, y organización de servicios, entre otros. Además, es capaz de modelar la infraestructura de IT describiendo como todos los sistemas involucrados se conectan entre sí, en lugar de manejar cada sistema por separado. Otra característica es que no requiere agentes o una infraestructura de seguridad particular, y utiliza un lenguaje simple como YAML que permite describir los procesos de automatización de una forma similar al inglés. En la Figura 2.5 se puede apreciar el flujo de configuración mediante Playbooks de Ansible, mientras que la Figura 2.6 muestra un ejemplo de su arquitectura. [Ansible, 2019]

## 2.3 Model Driven Engineering

A medida que los sistemas de software han evolucionado, los desarrolladores se han encontrado con nuevos desafíos en el ejercicio de su profesión. Para [Clark et al., 2015] los más importantes son:

**Complejidad** Debido a las necesidades de los negocios y usuarios, los sistemas de software son cada vez más complejos, lo cual trae problemas como tiempos de desarrollo más largos y mayor dificultad de mantener y testear código.

**Diversidad** La diversidad de dominios, requerimientos y tecnologías dificulta el proceso de desarrollo de software.

**Cambio** El cambio es la constante en los procesos de desarrollo de software: los requerimientos cambian, las tecnologías cambian, las funcionalidades cambian y este cambio dificulta y enlentece el desarrollo.

Es en este contexto que surge el concepto de la Ingeniería Dirigida por Modelos. La Inge-

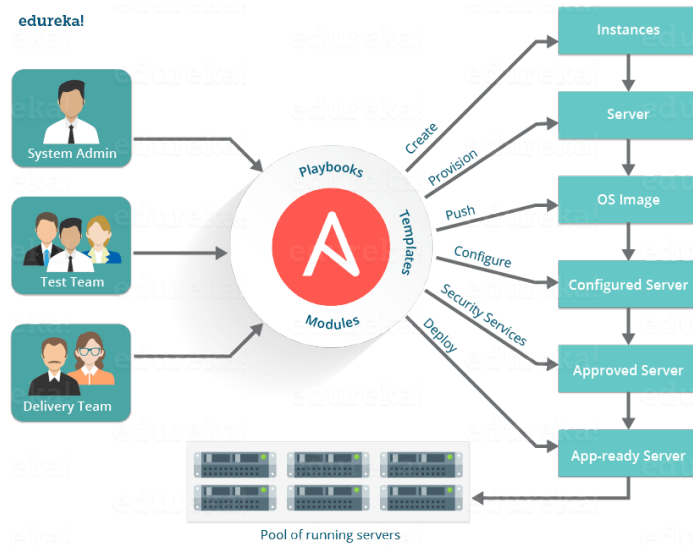


Figure 2.5: Representación del manejo de configuración de Ansible. [Ahmed, 2018]

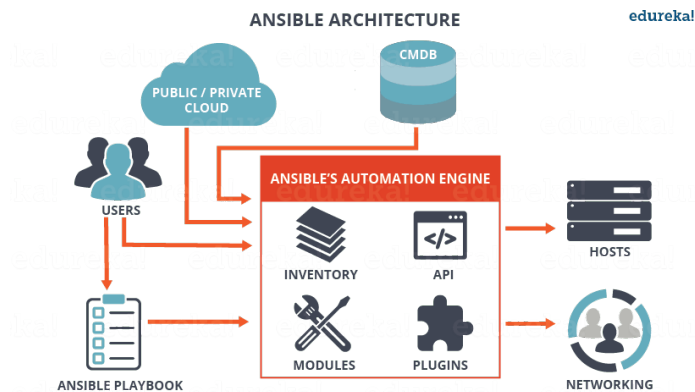


Figure 2.6: Representación de la arquitectura de Ansible. [Ahmed, 2018]

*nería Dirigida por Modelos* (a partir de ahora MDE, por sus siglas en inglés) es un paradigma en el cual el uso de modelos toma un papel fundamental en todas las actividades ingenieriles [Kent, 2002]. Es así que MDE como disciplina se basa en la idea de que se puede llegar a resolver un problema a partir de la confección de un modelo del mismo.

En una primera instancia sería fácil ver MDE simplemente como generación de software a partir de la creación de un modelo. Esto se denomina Desarrollo Dirigido por Modelos (MDD), pero la disciplina no se limita a esto ya que esta es sólo una de las actividades de la ingeniería, y el uso de modelos puede utilizarse para comprender mejor un problema, para realizar ingeniería inversa, para mejorar la comunicación entre miembros de un equipo o entre equipos, entre otras cosas.

Para entender el proceso de MDE hay que comprender ciertos conceptos fundamentales, como Modelo, Metamodelo y Transformación. A continuación se podrán encontrar definiciones y ejemplos para ayudar a entender claramente dichos conceptos.

### 2.3.1 Modelo

La Real Academia Española define modelo como [Española, 2017]:

- 4 m. Esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja, [...] que se elabora para facilitar su comprensión y el estudio de su comportamiento.

Esta definición da lugar a los dos roles más importantes que cumple un modelo:

**Reducción** Un modelo debe reflejar solamente una selección de las propiedades originales, de modo de enfocarse en los aspectos de interés.

**Mapeo** Los modelos son basados en un individuo original, el cual es tomado como prototipo de una categoría de individuos y es abstraído y generalizado en un modelo.

[Brambilla et al., 2012]

Entonces se puede decir que un modelo, en el contexto de MDE, es una abstracción simplificada del sistema o entorno que estudia.

### 2.3.2 Metamodelo

Algunas definiciones de metamodelo lo describen como “un modelo que define un lenguaje para expresar un modelo”; o como “un modelo de un lenguaje de modelos”. Esto quiere decir que un metamodelo describe completamente al lenguaje de modelado en cuestión: su sintaxis concreta, abstracta y semántica [Clark et al., 2015].



La relación entre metamodelo y modelo, se puede pensar como que un modelo “Se conforma a” un metamodelo. [Bézivin, 2005] Por lo tanto, un metamodelo es un modelo de una especificación para el cuál los sistemas estudiados son modelos, en un determinado lenguaje de modelado. La representación visual de un metamodelo en relación con el modelo y sistema se puede apreciar en la Figura 2.7. [Da Silva, 2015]

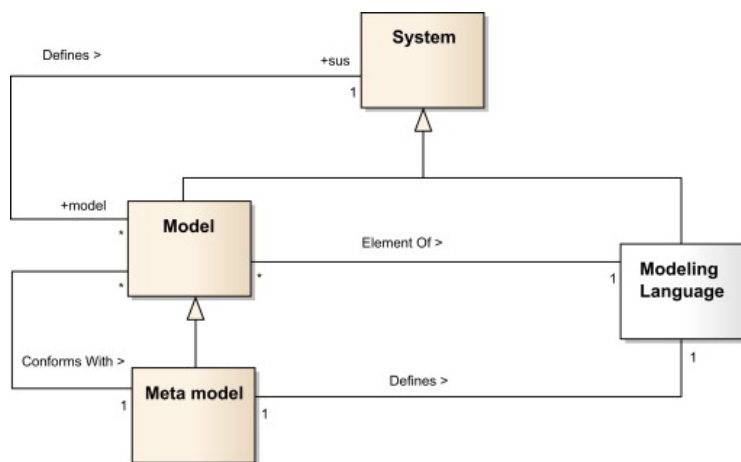


Figure 2.7: Relación entre metamodelo, modelo y sistema. [Da Silva, 2015]

La relación de un modelo y un metamodelo se puede explicar con tres propiedades de ambos y el lenguaje de modelado. Primero, a través de la relación *ElementOf* entre Modelo y Lenguaje de Modelado, un lenguaje de modelado es un conjunto de modelos, (o un modelo es parte de un lenguaje de modelado). Luego, a través de la relación *Defines* entre Metamodelo y Lenguaje de Modelado, un metamodelo es un modelo de una estructura de lenguaje de modelado (o un lenguaje de modelado es definido por un metamodelo). Finalmente, un metamodelo es un conjunto de modelos o modelo de modelos.

### 2.3.3 Transformación

Una Transformación de Modelo es el proceso de convertir uno o varios modelos (modelos origen) en un modelo salida (modelo destino/objetivo) del mismo sistema.

Kleppe [Kleppe et al., 2003] definía una transformación como la generación automática de un modelo destino a partir de un modelo origen, en base a una definición. Una definición de transformación es un conjunto de reglas de transformación que en conjunto describen como un modelo del lenguaje origen puede ser transformado en un modelo del lenguaje destino. Por último, una regla de transformación es una descripción de como uno o varios conceptos del lenguaje origen pueden ser transformados en uno o varios conceptos del lenguaje destino.

Las transformaciones también pueden ser aplicadas a múltiples modelos origen, obteniendo múltiples modelos destino, mientras que también se pueden generar transformaciones de modelo a texto [Mens and Van Gorp, 2006].

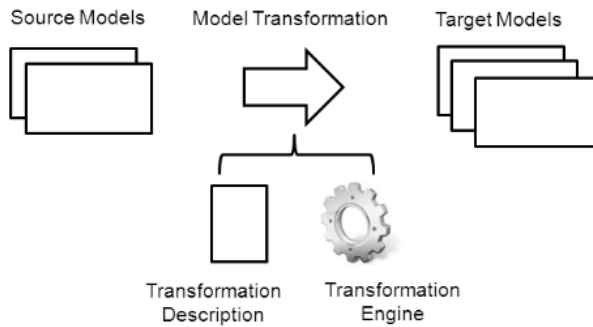


Figure 2.8: Componentes de una transformación de modelo. [Biehl, 2010].

De la misma forma en que existen transformaciones para obtener modelos de salida a partir de los modelos de origen, conocidas como transformaciones Model-To-Model, también se pueden generar transformaciones de modelo a texto, conocidas como Model-To-Text. El estándar de la transformación de modelo a texto MOF (mof2text) describe como pasar de un modelo a varios artefactos de texto, ya sea código, especificaciones de configuración, reportes, documentos, etc. En esencia, describe como obtener una representación en texto a partir de un modelo. Una forma de realizar esta transformación es mediante el uso de plantillas (*templates*), donde el texto que se genera a partir de los modelos se especifica a partir de un conjunto de plantillas de texto, que son parametrizadas con los elementos del modelo en cuestión.

En este enfoque un *template*, o plantilla, especifica un texto con marcadores a rellenar con datos que serán extraídos de los modelos. Estos marcadores son expresiones que se describen utilizando entidades del metamodelo y consultas (**queries**) para seleccionar y extraer los valores de los modelos. Estos valores son entonces convertidos a texto utilizando como base un lenguaje de expresiones, y alguna librería de manejo de *strings*. [Group, 2018]

### 2.3.4 Diagramas de deployment

Un Diagrama de *deployment* de UML es una representación del estado actual de la configuración de los nodos y los componentes que se ejecutan en esos nodos. Estos diagramas muestran el hardware del sistema, el software instalado en dicho sistema, y el middleware que conecta las máquinas entre sí. [Ambler, 2018]

En general se utilizan para describir sistemas no triviales, mostrando las diferentes interacciones entre sus componentes, por ejemplo aplicaciones que se ejecutan en varias computadoras, servidores utilizando servicios de firewall, arquitecturas de servicios web, o arquitecturas de sistemas embebidos, mostrando como el hardware y el software funcionan en conjunto.

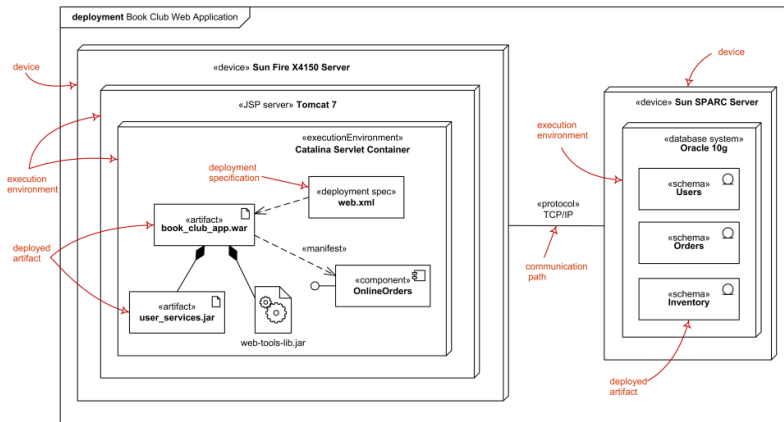


Figure 2.9: Diagrama de *deployment* a nivel de especificación. [Diagrams, 2018a]

Los diagramas de deployment se suelen crear durante la etapa de implementación en el ciclo de desarrollo, de forma de mostrar como se distribuyen los nodos en un sistema distribuido, qué artefactos se encuentran en qué nodos, y los componentes y otros elementos que los artefactos implementan. Los nodos representan dispositivos de hardware como computadoras, sensores, impresoras, y demás dispositivos que puedan soportar un ambiente de ejecución. Al mismo tiempo, las relaciones de deploy (o despliegue) y los enlaces de comunicación modelan las conexiones en el sistema. [IBM, 2019]

Estos diagramas son útiles para visualizar, especificar, y documentar diferentes tipos de sistemas, algunos ejemplos son los siguientes: [IBM, 2019]

- Sistemas embebidos que utilizan hardware que es controlado por señales externas, por ejemplo una pantalla que se controla por cambios de temperatura
- Sistemas cliente/servidor que distinguen su interfaz de usuario y los datos persistentes del sistema
- Sistemas distribuidos con múltiples servidores y que poseen diferentes versiones de artefactos de software, que pueden ser migrados de nodo a nodo

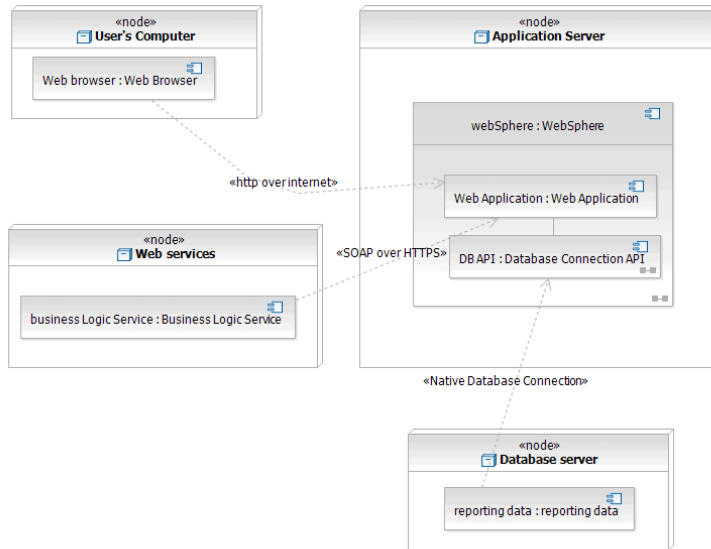


Figure 2.10: Ejemplo de un diagrama de deployment. [IBM, 2019]

### 2.3.5 Perfiles UML

Los diagramas de Perfiles en UML brindan un mecanismo genérico de extensión con el fin de personalizar modelos UML para ciertos dominios o plataformas. Estos mecanismos de extensión permiten refinar la semántica estándar de una forma aditiva, lo que evita conflictos con dicha semántica. Los perfiles se definen utilizando Estereotipos (stereotypes), Definiciones de Valores (tagged values definitions), y Restricciones (constrains), los cuales se aplican a elementos específicos del modelo como Clases, Atributos, Operaciones, y Actividades. Un perfil es entonces, una colección de estas extensiones, que en conjunto personalizan UML para un dominio particular [Paradigm, 2018]:

**Stereotypes:** Los estereotipos permiten incrementar el vocabulario de UML, agregando o creando nuevos elementos del modelo a partir de los existentes, pero que tienen nuevas propiedades específicas acorde a las necesidades particulares del dominio.

**Tagged Values:** Los valores etiquetados se utilizan para extender las propiedades UML, agregando información adicional en la especificación de un elemento del modelo. Estas se agregan especificando pares de clave-valor a un modelo, donde las claves son los atributos.

**Constrains:** Las restricciones son las propiedades que especifican la semántica o condiciones

que se deben cumplir en todo momento. Permiten extender la semántica de construcción de UML agregando nuevos protocolos.

En la Figura 2.11 se puede ver un ejemplo de como se define un perfil UML. En este caso se quiere definir el elemento Servers, por lo que extiende la metaclass Device con el estereotipo de dicho nombre, de esta forma se tendrá el elemento Server, que tendrá el comportamiento de la metaclass Device, y además poseerá los atributos que fueron definidos en dicho estereotipo: Vendor, CPU, y Memory.

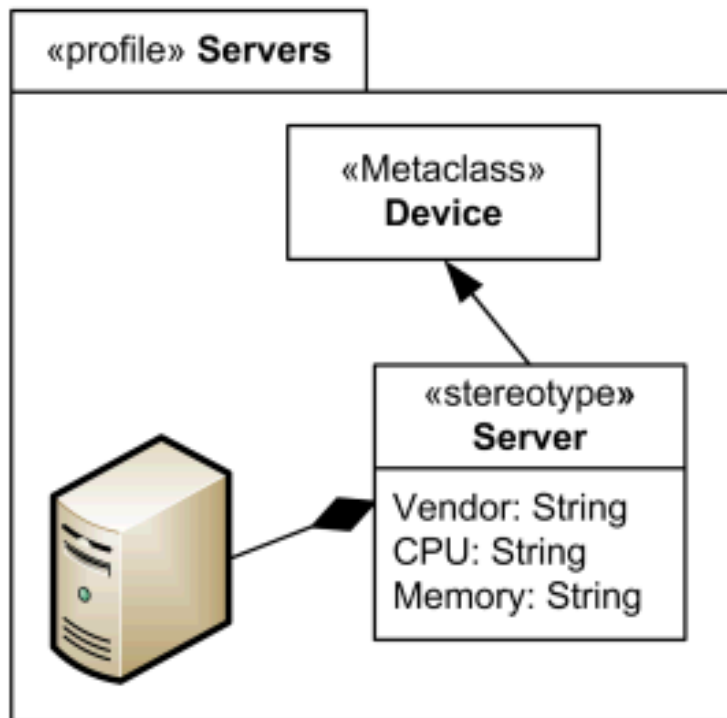


Figure 2.11: Ejemplo de perfil UML Servers. [Diagrams, 2018b]

El perfil podría utilizar todos, o parte de otros perfiles ya definidos, además, multiples perfiles pueden ser aplicados a un mismo modelo.

## 2.4 Herramientas de MDE

### 2.4.1 Eclipse Modeling Framework

Eclipse Modeling Framework (EMF) es un framework de modelado y generación de código diseñado para construir herramientas y otras aplicaciones basadas en un modelo de datos estructurado. A partir de una especificación de modelo en XMI, permite producir un conjunto de clases java para el modelo, así como un conjunto de clases adaptador que permiten visualizar, y utilizar comandos para editar el modelo. EMF también provee un editor básico para trabajar sobre el modelo y las clases de java.

En particular, EMF Core es un estándar para modelado de datos, por lo que muchas tecnologías y frameworks se basan en dicho estándar. EMF Core consiste de tres partes fundamentales [Steinberg et al., 2008]:

**EMF:** Incluye un meta modelo (Ecore) para describir modelos y soporte en tiempo de ejecución para los modelos incluyendo notificación de cambios, persistencia con serialización XML, y una API para manejar objetos EMF.

**EMF.Edit:** Incluye clases genéricas reusables, para construir editores de modelos EMF.

**EMF.Codegen:** La generación de código de EMF permite generar cualquier elemento necesario para construir un editor de modelos EMF completo. Incluye una interfaz gráfica (GUI) donde se pueden especificar opciones de generación, y llamar a los generadores.

### 2.4.2 Papyrus

Papyrus es una herramienta de modelado gráfico para UML2 basado en Eclipse. Al ser una herramienta de MDE, provee generación de código y permite interacción o conexión con herramientas externas de forma que los modelos sean los artefactos principales del proceso de desarrollo. Papyrus también posee muchas características para diseño de Domain Specific Modelling Languages (DSMLs) utilizando el concepto de perfiles UML, y por defecto incluye un conjunto de plug-ins de extensiones (perfiles UML 2) predefinidos dedicados a aplicaciones embebidas en tiempo real, como los estándares OMG (Object Management Group) SysML, MARTE, CCM, y LwCCM.

Algunas de las características de Papyrus descritas en su propia página web, son representadas en la Figura 2.12.

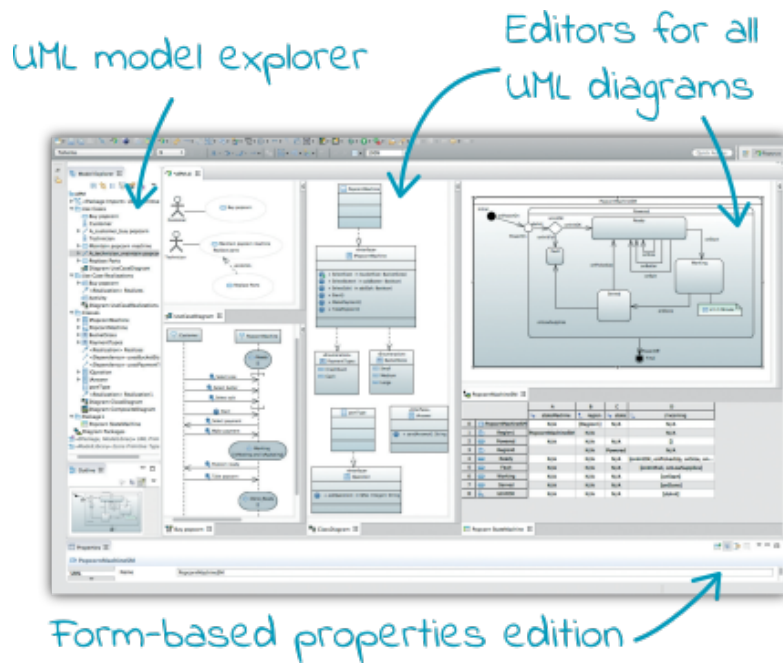


Figure 2.12: Características del editor de Papyrus. [Foundation, 2019d]

### 2.4.3 Acceleo

Acceleo es una herramienta de generación de código que implementa la especificación del estándar para transformación de modelo a texto MOF Model to Text Language del Object Management Group (OMG). Algunas de las características que el IDE provee a los desarrolladores son: sintaxis simple, generación de código eficiente, y herramientas avanzadas equivalentes a las de Java Development Tools. Su propósito es el de asistir al desarrollador durante el ciclo de vida de sus generadores de código. [Foundation, 2019a]

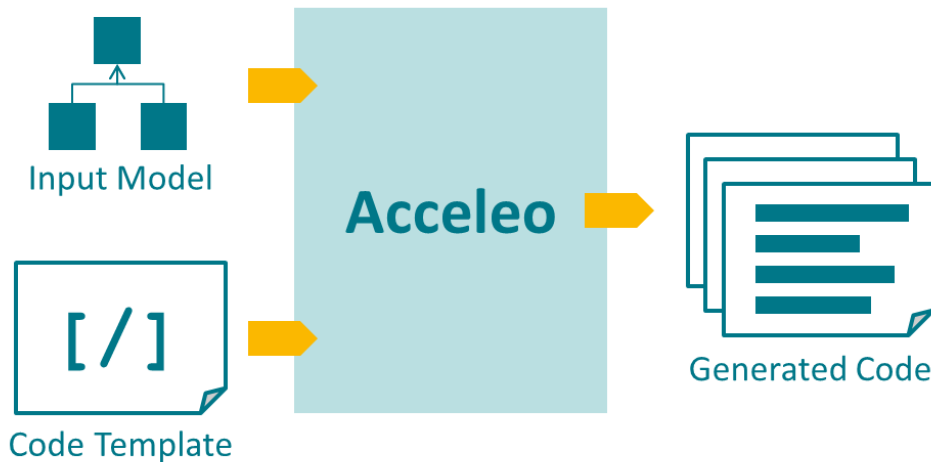


Figure 2.13: Proceso de transformación de Acceleo.

### 2.4.4 ATL

ATL (ATL Transformation Language) es un lenguaje y herramienta de transformación de modelos. Para MDE, ATL provee formas de producir un conjunto de modelos destino, a partir de un conjunto de modelos origen. Una transformación de ATL se compone de reglas que definen como los elementos de los modelos de origen se obtienen y navegan para crear e inicializar los elementos de los modelos destino. [Foundation, 2019b]

Una transformación de modelo a modelo se corresponderá con un módulo ATL. Estos módulos se encargan de definir dicha transformación, y están compuestos por los siguientes elementos [Foundation, 2019c]:

- una sección **header** (encabezado) que define algunos atributos relativos al módulo de la transformación



- una sección opcional **import**. que permite importar librerías existentes de ATL
- un conjunto de helpers que se pueden ver como el equivalente de ATL a los métodos de Java
- un conjunto de reglas que definen cómo los modelos de destino serán generados a partir de los modelos de origen

## 2.5 MDE en redes de computadoras

### 2.5.1 MDE en sensores de red inalámbricos

El artículo [Essaadi et al., 2017] presenta un enfoque de ingeniería dirigida por modelos para llevar a cabo el desarrollo de aplicaciones de Sensores de Red Inalámbricos. Es necesario manejar la complejidad en aumento de estos sistemas, y se enfoca en aumentar la flexibilidad y reusabilidad del diseño. Los tres niveles de abstracción definidos permiten construir: modelos de dominio específico, descripciones de arquitectura basada en componentes, y modelos específicos a la plataforma. También se definen transformaciones entre estos tres niveles.

### 2.5.2 YANG

YANG es un lenguaje de modelado utilizado para modelar configuración y estado de datos utilizados por el protocolo de configuración de redes NETCONF (Network Configuration Protocol), así como sus llamadas a procedimientos remotos, y notificaciones. Los modelos de datos definidos en un módulo de YANG se representan en XML, y las operaciones NETCONF se utilizan para manipular los datos. A su vez, cada módulo define la jerarquía de datos que pueden ser utilizados para operaciones basadas en NETCONF, incluyendo configuración, estado de datos, llamadas a procedimientos remotos (RPCs), y notificaciones. Esto permite una descripción completa de todos los datos enviados entre un cliente y servidor NETCONF. [Bjorklund, 2010]

### 2.5.3 Metamodelo para Configuraciones en Dispositivos de Redes

En la tesis con nombre Metamodelo para Configuraciones en Dispositivos de Redes como Estándar Soportado en la Ingeniería Dirigida por Modelos, se busca desarrollar una herramienta que estandarice y facilite la administración de redes, de forma de poder configurar cualquier protocolo, independientemente del proveedor, para lo que se desarrolló un metamodelo que permita generar dicha herramienta.

En este trabajo se describe el metamodelo, desarrollando un DSL para la configuración de dispositivos de red, sin considerar la configuración de computadoras o servidores. Para esto se definen los componentes necesarios, que serán implementados mediante el DSL. En primer lugar se tienen los componentes Router y Switch, asociados a un componente de Interfaz, luego se definen los componentes Brand (Marca) y Protocol (Protocolo), y finalmente se tiene el componente de Configuración que une todos los conceptos. En la Figura 2.14 se puede ver el diagrama de clases utilizado para dicha implementación, mientras que en la Figura 2.15 se tiene un ejemplo del resultado de un archivo de configuración para un router. Finalmente, no define ni busca integrarse con otra herramienta, sino que propone las bases con el metamodelo definido, que estandarizará y facilitará la configuración de redes. [Higuera Castro et al., 2016]

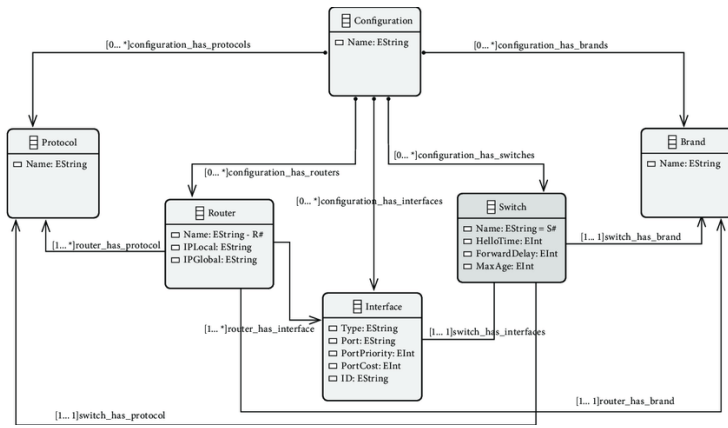
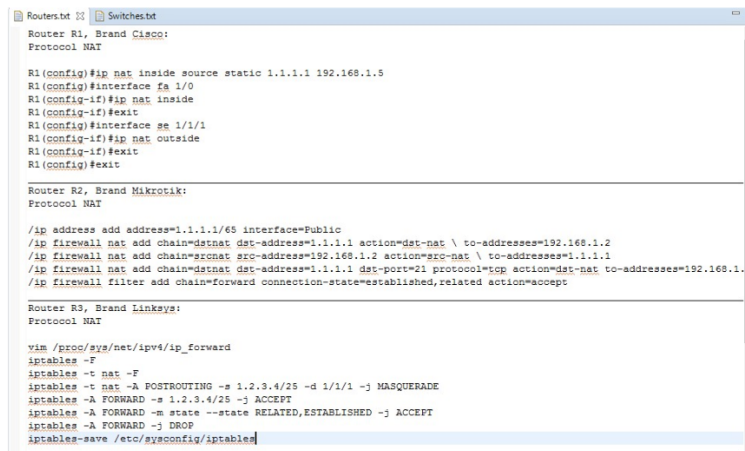


Figure 2.14: Diagrama de Eclases del modelo de dominio. [Higuera Castro et al., 2016]



```
Router.txt  Switch.txt
Router R1, Brand Cisco:
Protocol NAT

R1(config)#ip nat inside source static 1.1.1.1 192.168.1.5
R1(config)#interface fa 1/0
R1(config-if)#ip nat inside
R1(config-if)#exit
R1(config)#interface se 1/1/1
R1(config-if)#ip nat outside
R1(config-if)#exit
R1(config)#exit

Router R2, Brand Mikrotik:
Protocol NAT

/ip address add address=1.1.1.1/65 interface=Public
/ip firewall nat add chain=dstnat dst-address=1.1.1.1 action=dst-nat \ to-addresses=192.168.1.2
/ip firewall nat add chain=srcnat src-address=192.168.1.2 action=src-nat \ to-addresses=1.1.1.1
/ip firewall nat add chain=dstnat dst-address=1.1.1.1 dst-port=21 protocol=tcp action=dst-nat to-addresses=192.168.1.
/ip firewall filter add chain=forward connection-state=established,related action=accept

Router R3, Brand Linksys:
Protocol NAT

vim /proc/sys/net/ipv4/ip_forward
iptables -F
iptables -t nat -F
iptables -t nat -A POSTROUTING -s 1.2.3.4/25 -d 1/1/1 -j MASQUERADE
iptables -A FORWARD -s 1.2.3.4/25 -j ACCEPT
iptables -A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -j DROP
iptables-save /etc/sysconfig/iptables
```

Figure 2.15: Ejemplo de salida para la configuración de routers. [Higuera Castro et al., 2016]



# 3

## Análisis de requerimientos

---

La definición de requerimientos se comienza por aquellas necesidades a alto nivel que se buscan cubrir con la propuesta, es decir los requerimientos generales del proyecto, que identificaremos como **RQ**.

A su vez, el análisis se puede separar en base a las dos clases de requerimientos que se pueden ver. Por un lado se tienen los requerimientos del lenguaje a definir, ya sea creado desde cero utilizando un DSL, o extendiendo UML, para que cumpla con el objetivo planteado; y por otro lado, se tienen los requerimientos de la herramienta que se utilizará, esto es, cuales son las características que debe proveer la herramienta para trabajar sobre el lenguaje. Estos requerimientos no necesitarán una notación particular, ya que todos los mencionados serán considerados en el alcance del proyecto.

### 3.1 Requerimientos del Proyecto

#### 3.1.1 RQ1 - Lenguaje de especificación de la estructura de red

Comenzando por los requerimientos necesarios para obtener una prueba de concepto, es fundamental contar con un lenguaje de especificación de la estructura de la red, y de los dispositivos tanto de red como de usuario, obteniendo una base sólida para la herramienta de modelado.

El lenguaje de especificación debe tener flexibilidad para la incorporación de cambios, o la introducción de nuevos dispositivos. Este requerimiento es clave en el contexto de trabajo para el que podría ser utilizado, donde se podrían realizar cambios en la red frecuentemente, y por lo tanto es imprescindible evitar el re-trabajo con cada cambio en la configuración de la red o dispositivo.

### **3.1.2 RQ2 - Estandarización de modelos creados**

También es necesario contar con la capacidad de almacenar los modelos en base a un estándar, por ejemplo XML. Esto es sumamente importante si se busca la posibilidad de utilizar o integrar el resultado con herramientas de terceros.

### **3.1.3 RQ3 - Transformación de modelo a una configuración**

Dado que la herramienta busca una aplicación práctica, es necesario el pasaje de los modelos a una configuración aplicable a un sistema de computadoras, con esto en mente es claro que se necesita una generación automática de los archivos de configuración de los dispositivos a partir de la especificación de la estructura de red.

### **3.1.4 RQ4 - Aplicación de configuración generada**

De la misma forma que es necesario generar los archivos de configuración en base al modelo, también es necesario contar con la capacidad de aplicar esta configuración generada de forma automática y centralizada. Algunas de las ventajas que pretende brindar esta herramienta son la de reducir el trabajo necesario para realizar las configuraciones de forma manual, y facilitar el acceso a este tipo de configuraciones a técnicos (o interesados en el campo) que no tengan un conocimiento avanzado en sistemas de estas características. Por lo que se debe ser capaz de implementar, o instalar, los archivos de configuración generados de una forma sencilla.

En este último punto se puede incluir esta funcionalidad como uno de los objetivos, u optar por una herramienta existente que cumpla con este propósito. El formato o estructura del archivo de configuración generado definirá en gran medida la decisión de esta herramienta.

### **3.1.5 RQ5 - Detección de red existente**

Una característica que se encuentra presente en algunos productos (Véase [2.1.3](#)) es la capacidad de detectar una red existente, y generar un modelo a partir de la misma. Este descubrimiento de modelos a partir de la información de la red existente podría ser de gran utilidad al momento de realizar la configuración inicial, ya que se evitaría la necesidad de crear el modelo desde cero. En su lugar, se detectaría la topología actual, y luego se trabajaría sobre esta base. Esta funcionalidad crece en importancia a medida que el tamaño de la topología inicial es mayor, e incluso podría ser imprescindible en casos donde el sistema o red objetivo sean del orden de cientos de dispositivos.

### 3.1.6 RQ6 - Detección de cambios en la red

De forma similar a lo mencionado anteriormente, también sería deseable que la herramienta contara con un sistema de detección de cambios, es decir, que sea capaz de detectar cambios en la red o en los dispositivos, y sea capaz de actualizar el modelo correspondiente adaptándolo de forma acorde.

En este punto cabe mencionar una herramienta que posee las características mencionadas, la cuál se vio en la Sección 2.1.3, Network Topology Mapper (mapeador de topología de red). Como se vio anteriormente, esta herramienta esta enfocada a la administración de redes, y es capaz de escanear y mapear la infraestructura de red, incluyendo dispositivos de red, servidores, y hosts virtualizados.

### 3.1.7 RQ7 - Distribución de archivos generados entre los nodos

Finalmente, una característica a considerar es la capacidad de enviar los archivos de configuración generados a los nodos correspondientes. Esto es fundamental para lograr una centralización con la herramienta, y va de la mano con uno de los objetivos mencionados, que es el de configurar los dispositivos de manera centralizada. Sin embargo, esto se puede lograr con diferentes herramientas, y dependiendo de la decisión que se tome en cuanto a la configuración de los nodos, se podría lograr de forma automática.

Un caso particular a considerar son los manejadores de configuración, o CMTs (por sus siglas en inglés Configuration Manager Tools), los cuales son servidores centralizados capaces de realizar configuraciones remotas a partir de scripts de configuración, en estos casos, la distribución de los archivos de configuración a los nodos no sería necesaria, dado que sólo se necesitaría generar un script de configuración a ser utilizado por el CMT en su servidor correspondiente. Algunos ejemplos de estas herramientas se vieron en la Sección 2.2.

En caso de optar por no utilizar una herramienta de este estilo, sería necesario implementar la distribución y aplicación de las configuraciones generadas a partir del modelo, de forma que la propagación de cambios sea centralizada, y no se tenga la necesidad de aplicar la configuración a cada dispositivo, lo cual iría en contra del objetivo del proyecto.

## 3.2 Requerimientos del Lenguaje

Una vez analizados los requerimientos a alto nivel, se puede comenzar a entrar en detalle sobre los mismos. Para ello se separan en dos clases, por un lado se tienen las características que deberá proveer el lenguaje, y por otro lado las que deberá cumplir la herramienta de configuración.

Comenzando por los requerimientos del lenguaje (ya sea DSL, UML2, etc), se deben describir los que se consideran de importancia para poder definir los modelos necesarios. Partiendo de los requerimientos **RQ1** (3.1.1) y **RQ2** (3.1.2), uno de los más importantes es poder describir diferentes tipos de elementos, dado que se contemplarán los diferentes tipos de nodos que se encuentran en una red, como un PC, un servidor, o dispositivos de red (routers, switches). Esto quiere decir que los componentes disponibles a modelar deberán ser definidos de antemano, de forma de saber que elementos se van a agregar al lenguaje.

También es importante poder describir las características de los diferentes nodos. Cada componente tendrá diferentes especificaciones, que podrán cambiar en el tiempo, y éstas deben ser reflejadas en el modelo, ya sea de forma gráfica o en forma de propiedades en texto plano. De forma análoga a los componentes, estas características o propiedades que sean necesarias deberán ser definidas de forma previa, para que sean incluidas en el lenguaje.

Por último, siguiendo la línea del requerimiento **RQ3** (3.1.3), se debe contar con un diagrama que represente el lenguaje de especificación de la red (LER). Es decir, se debe definir un lenguaje gráfico, describiendo como se mapean los elementos al diagrama, y definiendo los diferentes elementos que serán utilizados en el diagrama, con sus respectivos atributos.

Estos diagramas serán la base para la herramienta de generación de código. También es importante que el diagrama pueda estar basado en un estándar existente, de forma que su extensión e integración con otras herramientas sea posible.

### 3.3 Requerimientos de la Herramienta

En cuanto a los requerimientos de la herramienta, si bien algunos estarán ligados a la definición del lenguaje, es posible destacar los más importantes para analizar.

Primero es necesario definir que tareas serán realizadas por la herramienta, y que tareas se van a delegar a otras, como la configuración de dispositivos y red en sí, es decir, si se utilizarán herramientas como soporte durante el proceso de transformar un modelo a un script de configuración. Esto hace referencia al requerimiento **RQ4** (3.1.4) en particular, aunque dependiendo de la herramienta y del alcance del proyecto, también podría cubrir **RQ5** (3.1.5), **RQ6** (3.1.6), y **RQ7** (3.1.7).

Una posibilidad es utilizar un Configuration Manager Tool (CMT) para realizar la configuración de los nodos y de la red. La elección del CMT definirá el formato de salida del archivo de configuración, el cual será utilizado por esta herramienta, asimismo, se puede analizar la posibilidad de generar un script de configuración genérico, por ejemplo en formato XML, que luego se pueda adaptar a varios CMTs, o tener la posibilidad de generar cada script en diferentes formatos para cada CMT correspondiente. Por otro lado, se debe definir el flujo de trabajo que tendrá la herramienta, esto es, cuáles son las etapas de uso que tendrá. Una descripción a alto nivel de los pasos a seguir puede ser la que se describe a continuación.



Primero se debe crear el diagrama del estado actual de la red, y sus respectivos nodos, por lo tanto se necesitará de un editor que permita utilizar el lenguaje definido para crearlo. Una vez que se tiene dicho diagrama, es posible editarlo, para representar el estado deseado de la red. Con el diagrama definido, se debe poder exportar, ya sea a un formato estándar como XML, o directamente a un formato adecuado para la utilización de un CMT. En caso de que se exporte a un formato estándar, también se deberá definir la transformación de este estado, a los diferentes CMTs. Finalmente, una vez generado y exportado el script al formato adecuado, se debe aplicar la configuración a los diferentes nodos y dispositivos de red. Este último paso puede ser realizado de forma directa con las herramientas mencionadas previamente.

### 3.4 Alcance del Proyecto

Analizando los requerimientos mencionados anteriormente, surge que aquellos fundamentales son **RQ1** (3.1.1), **RQ2** (3.1.2), **RQ3** (3.1.3), y **RQ4** (3.1.4), ya que son indispensables para tener un producto funcional, y que permita realizar el proceso de configuración en su totalidad, esto es, definir el modelo del sistema estudiado, generar los archivos de configuración mediante la transformación, y aplicar dicha configuración al sistema en cuestión. De la misma forma, tanto los requerimientos del lenguaje (3.2) como de la herramienta (3.3) se consideran dentro del alcance, debido a que son esenciales para tener una prueba de concepto.

Por otro lado, dada la complejidad de los requerimientos **RQ5** (3.1.5), **RQ6** (3.1.6), y **RQ7** (3.1.7), se optó por dejarlos fuera del alcance. Más aún, se considera que la funcionalidad de descubrimiento y detección de una red podría abarcar un proyecto por sí misma. Sin embargo, se podría considerar la posibilidad de integración con una herramienta con estas características. Esto sucede en el caso del requerimiento **RQ7**, dado que la distribución de archivos de configuración vendrá dada por la integración con el CMT que se utilice, como se mencionó en dicha sección.



# 4

## Diseño de la Solución

---

En base a los requerimientos analizados en la sección anterior, en esta sección se detallan las decisiones tomadas de forma de brindar una respuesta a los mismos. Primero se describen las decisiones con respecto a los elementos del dominio, esto es, que elementos se van a considerar para los siguientes pasos, a partir de esto se describe el modelo UML que será la base del proyecto, y finalmente se definen las herramientas que se utilizarán como soporte para poder cumplir con el resto de requerimientos, es decir, la transformación y aplicación de archivos de configuración.

### 4.1 Definición de Dominio Específico

Partiendo de que el objetivo del proyecto es ofrecer la capacidad de modelar la configuración de una red de dispositivos, se tiene que los conceptos claves a modelar son los dispositivos y la configuración aplicada a los mismos. Esto también se puede ver como la topología de la red, es decir, los nodos que se encuentran en la red a modelar, y la configuración aplicada a dicha red y sus respectivos dispositivos o nodos.

La topología de red puede verse desde dos enfoques: el enfoque físico y el lógico, de los cuales se desprenden los conceptos de nodo físico y nodo lógico. Se consideran nodos físicos a los dispositivos en sí, como lo son las computadoras o routers, con el fin de representar de forma precisa los diferentes dispositivos físicos que se encuentra en la red. Por otro lado, se consideran nodos lógicos a los componentes que se encuentran en los nodos físicos, estos van desde sistemas operativos hasta programas de software instalados, ya que el objetivo es representar el estado del dispositivo de forma tan precisa como sea posible. Con estos elementos se obtiene una imagen o descripción general de los nodos presentes en la infraestructura de la red.

### 4.1.1 Nodos Físicos

Desde el punto de vista de los nodos físicos, los componentes a modelar serán los dispositivos de red, es decir, los routers y switches, los servidores, las computadoras personales (pc o workstation), y por último el resto de dispositivos que se puedan conectar a una red, como puede ser una impresora o un escáner de red. A estos últimos los denominaremos con el término genérico de "dispositivos".

Se consideran estos componentes por ser los básicos y más usuales que se encontrarán en una red, cubriendo así la mayoría de los escenarios posibles. Los dispositivos de red son la base para un sistema de este estilo, por lo que es imprescindible que **Router** y **Switch** sean representados. Luego es necesario hacer una distinción entre **Servidor** y **PC**, ya que cumplen funciones diferentes e interesa representar esta diferencia en el lenguaje. Además en la mayoría de los casos se encontrarán en diferentes cantidades. Finalmente se tiene el componente **Dispositivo** para encapsular al resto de componentes y poder representar de forma más fiel la realidad del sistema estudiado.

### 4.1.2 Nodos Lógicos

En el caso de los nodos lógicos, interesa saber el **Sistema Operativo** de los **Servidores** y **PC** por un lado, y el **Firmware** de los nodos de red por otro lado. Asimismo, se debe tener en cuenta el software instalado en cada componente, en particular en los **Servidores** y **PCs**, por lo que deberán ser contemplados en el modelo.

Los componentes de software considerados para modelar son los siguientes:

- un componente de **Firewall**
- un ambiente de ejecución (**Runtime**), en este caso se consideró el de **Java**
- un servidor de aplicaciones (**Application Server**), **Tomcat** en este caso
- un servidor HTTP (**HTTP Server**), para el que se utilizará **Apache**
- motores de bases de datos (**Database Engines**), para los cuales se representará **MySQL** y **PostgreSQL**

La decisión de estos componentes se basa en tener diferentes clases de componentes, y que a la vez sean normalmente utilizados por CMTs (Configuration Manager Tools), la importancia de esto se explicará en detalle en la sección **CMTs a Utilizar**. También es necesario acotar la cantidad de componentes de software ya que estos deberán ser modelados, y siendo esta una primer iteración del objetivo, tendrá más importancia obtener una prueba de concepto, que luego pueda ser extendida fácilmente con más componentes.

### 4.1.3 Configuración

La configuración es el punto más importante, y que mayor valor da al proyecto, ya que la transformación se basará principalmente en este componente y en los componentes lógicos de software para generar los scripts de configuración, los que finalmente serán utilizados para aplicar el estado a la red. Por esto, es importante que se contemplen la mayor cantidad de escenarios posibles de configuración. Al igual que con los componentes de software, parte de la decisión se basa en el uso de CMTs (Configuration Manager Tools), que se explicará en detalle en la sección **CMTs a Utilizar**.

En este caso se definen tres clases de configuración en función de las características y la relación con el resto de los elementos. Primero se tiene la **configuración de software**, esta clase se corresponderá con los componentes de software definidos como nodos lógicos. Los elementos de este tipo son específicos a cada componente de software ya que representan los aspectos configurables particulares de cada software que se desea configurar.

Luego se define la **configuración de archivo**, nombrada de esta forma por ser análoga a la función que cumple dicha configuración en los CMTs, esto es, acciones discretas sobre archivos del sistema, por ejemplo: verificación de presencia o contenido, copiado o borrado de dichos archivos, etc. Este tipo de elemento es muy importante ya que, en ausencia de un elemento de software o configuración del mismo se puede recurrir a manipular directamente archivos para llevar a cabo una configuración.

Finalmente, se tiene una configuración libre, cuyo propósito es la de dar la oportunidad de registrar cualquier tipo de configuración que no pueda ser aplicada de forma automática por las herramienta, pero que podría interesar guardar un estado, como lo son los componentes **Dispositivos**. De esta forma, se pueden modelar opciones de configuración que no estén soportadas por un CMT, o cualquier comentario que se desee guardar en el correspondiente nodo. Un ejemplo de esto puede ser la configuración de una impresora: las impresoras en general no proveen un sistema de configuración automática y se debe realizar la misma de manera manual. Con un elemento de configuración libre se puede asociar a la impresora una lista de los distintos valores a configurar, un tutorial o un link a una guía de configuración.

De forma similar, pero sólo aplicable en el caso de sistemas operativos Windows, se tiene una cuarta clase de configuración llamada **Registro (Registry)**, que como el nombre lo indica, son configuraciones que pueden ser aplicadas al registro de Windows. Esta última clase es necesaria ya que una gran cantidad de configuraciones en Windows se realizan a través de sus registros.

## 4.2 Lenguaje de Modelado del Dominio

### 4.2.1 Lenguaje a Utilizar

Antes de comenzar con la implementación del modelo, es necesario elegir el lenguaje que se va a utilizar. En este sentido hay dos grandes opciones, por un lado se tienen los lenguajes de dominio específicos o DSL por sus siglas en inglés (Domain Specific Languages), y por otro lado se tienen los perfiles UML.

Los DSL son utilizados para resolver un problema en particular, y representan un problema o dominio específico, como su nombre lo indica. Los perfiles UML, tal como se detalló en la introducción, se utilizan para extender modelos UML para ciertos dominios o plataformas. En base a estas descripciones, se puede ver que ambos lenguajes serían apropiados para cumplir con el fin de representar la realidad planteada.

En este caso, luego de analizar diferentes ventajas y desventajas de ambos, se optó por utilizar el perfil UML. La principal razón de la decisión se basa en que una gran parte de lo que se quiere representar, como el caso de la topología, ya posee contraparte en UML, por lo que sería favorable partir de esta base.

Sucede lo mismo con los diagramas de deployment de UML, su representación de nodos y componentes que se ejecutan en dichos nodos, se ajusta a la descripción del problema planteado, por lo que se puede tomar ventaja de dichos elementos y reutilizarlos para la implementación de la solución.

Otra razón para esto es la estandarización de UML comparado a un DSL. Al extender UML se puede hacer uso de todas las herramientas ya existentes para dicho lenguaje, mientras que para un DSL se deberían implementar todo un ecosistema análogo. Al elegir UML como base ya se tienen editores, mecanismos de extensión y una sintaxis en la cual basarse.

Finalmente, si se piensa en la implementación es posible ver una analogía entre elementos existentes de UML y el problema que se intenta resolver, que servirán al momento de definir la extensión, esto es, el concepto de **Nodo** y **Deployment** que posee UML.

En primer lugar, según la especificación formal de UML [Group, 2015], el elemento **Nodo** se divide en dos especializaciones, por un lado **Device** representa componentes de máquinas físicas, que coincide con el concepto de nodo físico; y por otro lado **ExecutionEnvironment**, se asignan a componentes de Device, y representan sistemas de software estándar, lo que es análogo a lo que se definió como nodo lógico.

Por otro lado, el elemento **Deployment** representa las relaciones entre elementos físicos y lógicos de un sistema, así como características asignadas a estas relaciones. En este caso, el nodo lógico que se menciona en la especificación coincide con nuestro concepto de configuración.

La representación de UML para el elemento **Node** es capturada en la Figura 4.1, mientras que la representación de **Deployment** es capturada en la Figura 4.2.

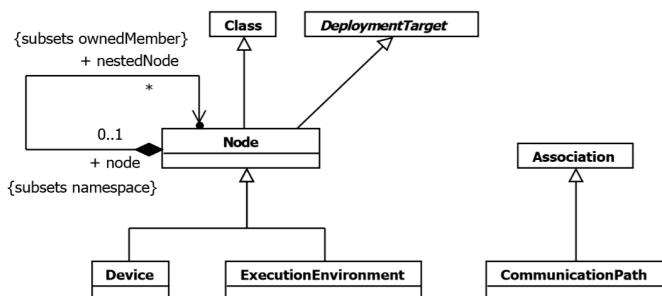


Figure 4.1: Representación de UML de Node.

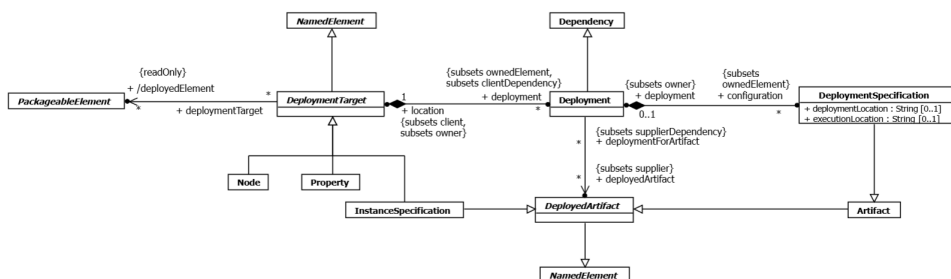


Figure 4.2: Representación de UML de Deployment.

El conjunto de estas ventajas, y en particular las similitudes encontradas entre los componentes definidos por UML y la realidad que se desea resolver, llevaron a optar por la creación de un perfil UML, en lugar de utilizar un DSL.

### 4.3 CMT a Utilizar

Otro paso fundamental en el diseño de la solución es la elección de la herramienta de configuración a utilizar. Una posibilidad era utilizar un Configuration Manager Tool que se encargue de la configuración de los nodos y de la red, y éste fue el camino por el que se optó.

La principal ventaja de estas herramientas es que utilizan scripts con un formato específico para realizar la configuración, lo que es sumamente importante al momento de generar dichos

archivos, ya que se tendrá un formato definido y único. Otra ventaja de gran valor es que dichas herramientas centralizan la configuración, lo cuál era una de las características claves que se analizaron en capítulos anteriores.

Con esta decisión tomada, el siguiente paso es la elección de los CMTs a utilizar. En este paso se analizaron una variedad de CMTs, entre los cuales destacaron Puppet, Chef y Ansible. Finalmente se decidió decantarse por el lado de Puppet o Chef, ya que ambos pertenecen a lo que se podría denominar una misma familia de sistemas de manejo de configuración: ambos tienen un lenguaje declarativo que describe el estado de un sistema y apuntan a su uso durante todo el ciclo de vida del mismo. Ansible por el otro lado se enfoca más en el principio de este ciclo, en la etapa de aprovisionamiento de servidores, con lo cual mantener el estado más allá de esta etapa implicaría trabajo extra con respecto a Puppet/Chef, aunque de todos modos sería posible.

Finalmente se eligió Puppet como objetivo principal ya que si bien tanto Puppet como Chef son herramientas líderes en el mercado [Puppet, 2018], debido a la familiaridad del equipo con el primero se consideró que sería más sencillo utilizarlo. Adicionalmente ambos son tan similares (estructuras de directorios, archivos de configuración, sintaxis, arquitectura) que no se consideró que realizar transformaciones para ambos agregara valor.

## 4.4 Transformación M2T

Finalmente se debió diseñar la transformación M2T que tomará el modelo de la realidad, y tendrá como salida la configuración necesaria para que el Puppet Master pueda impactar el estado deseado en la red.

Para cada elemento del dominio específico se generará un archivo. Dichos archivos se organizarán en una estructura de directorios arborescente, donde se diferenciarán en tres clases, y dentro de cada clase se organizarán en base a cada elemento y sus relaciones. De este modo se tiene una gran modularidad y simplicidad en cada archivo, ya que en el mismo sólo se indican las propiedades del elemento correspondiente, y se apunta a otros archivos de similar complejidad.

En líneas generales, se plantean las siguientes reglas:

- En primer lugar se genera un archivo `site.pp` donde se listan todos los nodos físicos que permitan una configuración. Para cada nodo se genera una expresión regular y se le asigna una clase propia.
- Para cada nodo físico se genera un archivo cuyo contenido son *includes* de las clases generadas para los nodos lógicos pertenecientes a dicho nodo físico. Este archivo se ubica en un directorio llamado "device"; también se crea un directorio con el nombre del nodo físico, que contendrá los archivos generados por sus nodos lógicos asociados.



- De similar manera, para cada nodo lógico se genera un archivo cuyo contenido son los valores básicos de la instalación del mismo, así como *includes* para las configuraciones asociadas. Este archivo se ubica en el directorio previamente creado para el nodo físico correspondiente; al mismo tiempo se crea un directorio dentro del directorio "configuration" con el nombre del nodo lógico, que contendrá los archivos asociados a los elementos de configuración de este nodo.
- Luego, para cada configuración se genera un archivo con los valores de la misma. El archivo se ubica dentro del directorio del nodo lógico correspondiente, creado previamente en el directorio "configuration".
- Finalmente, para cualquier elemento que no corresponda crear un script de configuración, se crea un archivo de texto plano con su información en forma de clave valor, este archivo tendrá el nombre del elemento, y se ubicará en el directorio del nodo físico correspondiente (su propio directorio en caso que sea un nodo físico), dentro del directorio "information".

Un ejemplo de esta estructura puede ser el que se encuentra en la Figura 4.3.

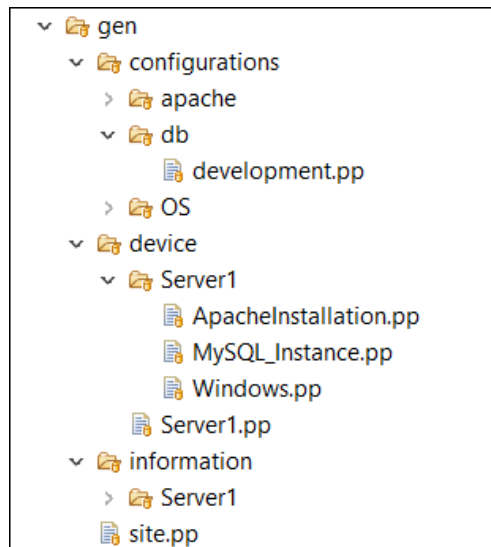


Figure 4.3: Ejemplo de estructura de archivos generados.



# 5

## Implementación de la Solución

---

En esta sección se detallan las implementaciones tecnológicas de los objetivos del proyecto, a partir de las soluciones mencionadas en el capítulo anterior. Estos objetivos son la definición de un lenguaje de modelado, que permitirá representar en forma de diagrama el estado de una topología de red. Esto incluye la creación del metamodelo en base a la Definición de Dominio Específico, y la implementación del perfil UML para agregar la extensiones de UML necesarias. La implementación de una transformación de modelo a texto, que permita la creación de scripts de configuración a partir de dichos modelos. La integración de los scripts de configuración generados con las herramientas de configuración (CMTs), detallando como estas herramientas utilizan los scripts para llevar a cabo la configuración.

### 5.1 Implementación del modelo UML

Partiendo de las decisiones que se tomaron en el capítulo anterior, en esta sección se especifica el modelo UML definido para representar el problema objetivo. De forma análoga a la sección Definición de Dominio Específico, se tendrá el metamodelo separado en dos grandes partes, por un lado se encuentra la topología, con nodos lógicos y físicos, y por otro lado se tiene la configuración de estos nodos.

A continuación se describe cada sección por separado, ilustrando las decisiones tomadas en cada una de ellas, para luego mostrar el resultado final del metamodelo, que será el punto de partida para la implementación del perfil UML, y por ende la transformación de modelo a texto.

5.1.1 Modelado de la Topología

El primer componente que se debe considerar dentro de la topología es el de **Nodo**, dicho componente encapsula todo tipo de nodo, ya sea físico, o lógico, por lo que puede representar desde un servidor hasta una instancia de base de datos. Este concepto es esencial ya que la configuración que se definirá debe ser capaz de ser aplicada tanto a dispositivos lógicos como físicos, y es necesario que haya un componente en común para representar ambos tipos de nodos independientemente.

De esto, y tal como se explicó anteriormente en la sección Definición de Dominio Específico, surge que se tendrán dos tipos de nodos: físicos y lógicos, lo que se representará con una relación de herencia sobre Nodo, quedando representado en la Figura 5.1.

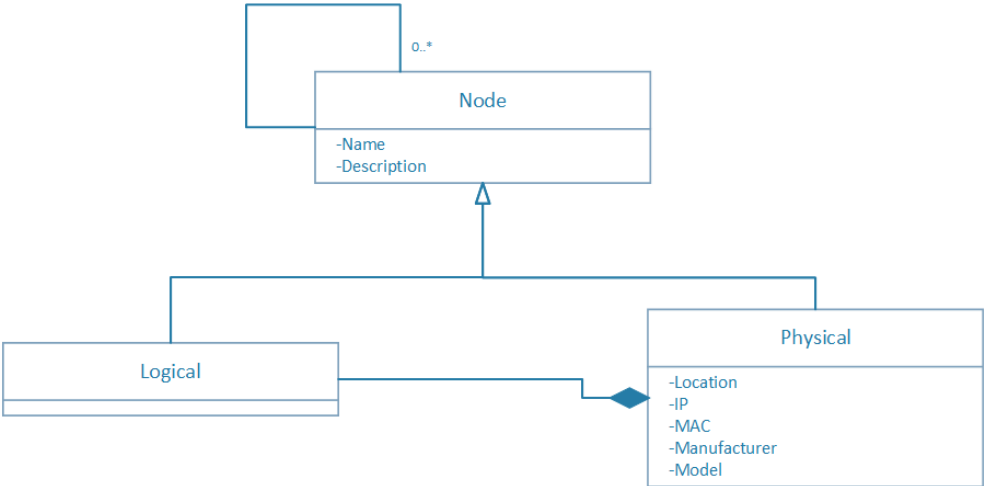


Figure 5.1: Representación UML de Nodo.

Además de la relación de herencia de Lógico y Físico sobre Nodo, se tiene una relación de Composición de un nodo Lógico a un nodo Físico, dado que un nodo Físico estará compuesto por varios nodos Lógicos.

También se agregan los atributos considerados principales para el nodo Físico, en el caso del nodo Lógico no se tienen atributos debido a que no existe alguno que se considere esencial y sea común a las especializaciones que se verán luego.

Por último, el componente de nodo tiene una relación con multiplicidad de cero a varios consigo mismo, con el fin de considerar la relación que existe entre dispositivos de red con servidores, o incluso otros dispositivos de red.

Una vez definido el componente de Nodo, se pasa a definir las especializaciones Físico y Lógico.

### Nodo Físico

Respecto a los nodos físicos, se mantiene la definición de dominio, es decir, se tienen las especializaciones Servidor (Server), PC, Dispositivo (Device), y se agrega una nueva relación de herencia para los Dispositivos de Red (Network), donde cada uno es una generalización de Router o Switch, y al mismo tiempo, estos son especializaciones de Dispositivo de Red. El metamodelo del componente Físico queda representado en la Figura 5.2.

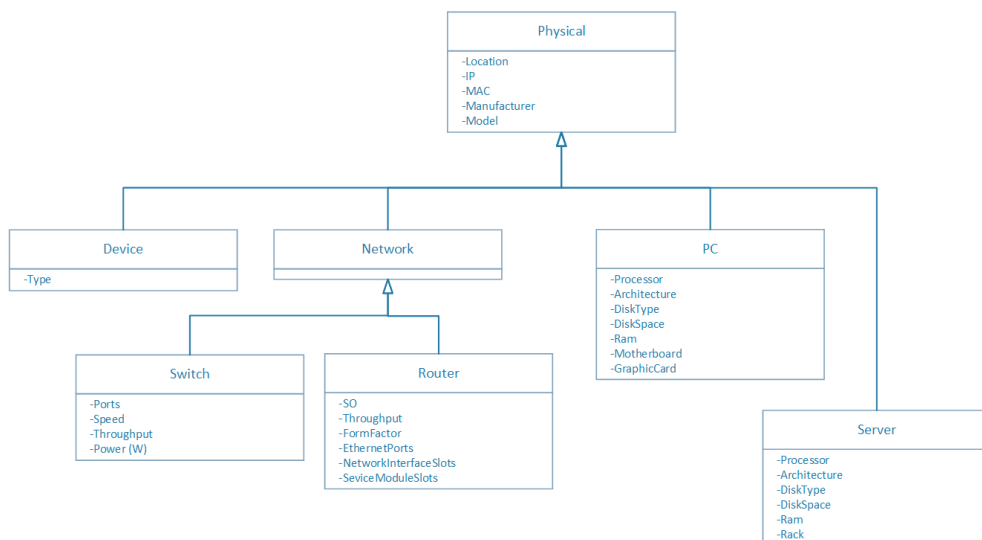


Figure 5.2: Representación UML de Nodo Físico.

Los atributos de cada componente fueron seleccionados en base a la información de interés para cada uno en el contexto de la topología y configuración, siempre teniendo en cuenta que los principales son heredados del nodo Físico (Physical).

### Nodo Lógico

De la misma forma, los nodos lógicos fueron seleccionados en base a las decisiones de definición de dominio previamente realizadas, por lo tanto, se considerarán como especialización de Lógico (Lógico) Firewall, Sistema Operativo (OS), Firmware, Ambiente de Ejecución (Runtime), Servidor de Aplicaciones (ApplicationServer), Base de Datos (DBEngine), y Servidor HTTP (HTTPServer).

A su vez, cada clase tiene una especialización en base los componentes de software que se consideraron para cada uno, por lo que se tienen las especializaciones Java para Runtime; Tomcat para ApplicationServer; Postgresql y MySQL para DBEngine; y Apache para HTTPServer.

En base a esto, el metamodelo del componente Lógico se puede ver en la Figura 5.3.

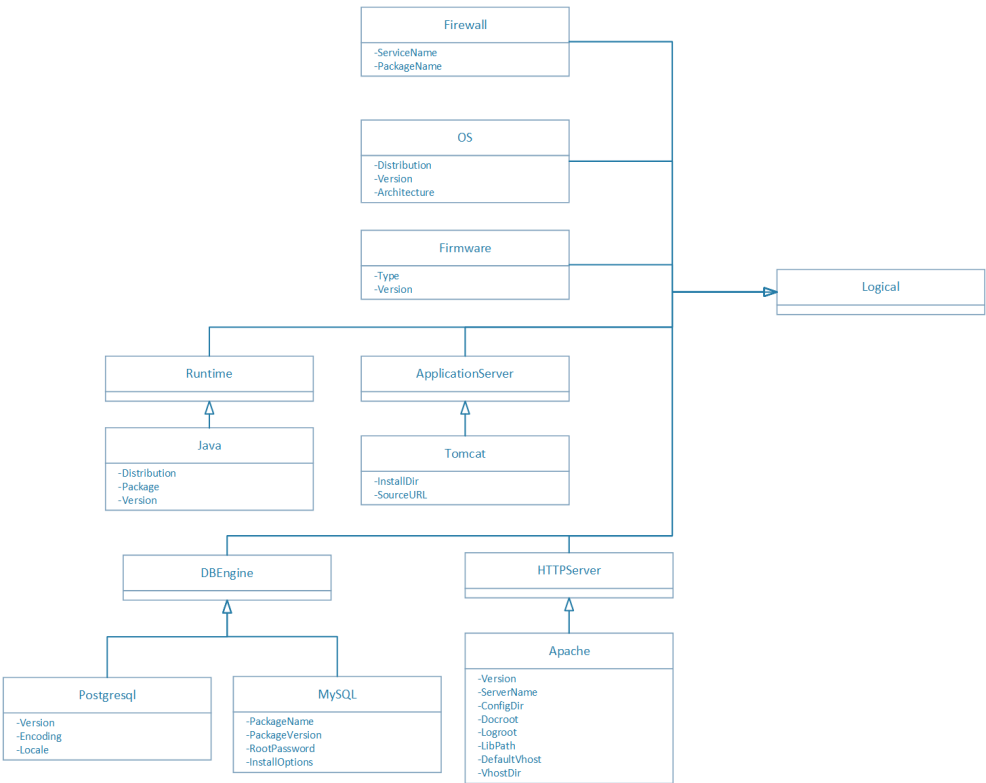


Figure 5.3: Representación UML de Nodo Lógico.

Los atributos de Sistema Operativo y Firmware fueron seleccionados en base a la información básica y necesaria en el contexto del problema, ya que serán utilizados de forma puramente informativa. Sin embargo, los atributos de los componentes de software debieron ser seleccionados cuidadosamente, tomando como referencia el soporte brindado por los CMTs a utilizar. Dichos componentes fueron seleccionados a partir del soporte brindado por los manejadores de configuración, por lo tanto, sus atributos se deben corresponder con los atributos que estos dispongan. En base al estudio de los diferentes componentes en cada CMT (Puppet, Chef) se obtuvieron los atributos que se muestran en la imagen.

**Topología**

Una vez definidos los metamodelos de Nodo, Físico y Lógico, sólo resta unificar los resultados para obtener el metamodelo de la topología, en la Figura 5.4.

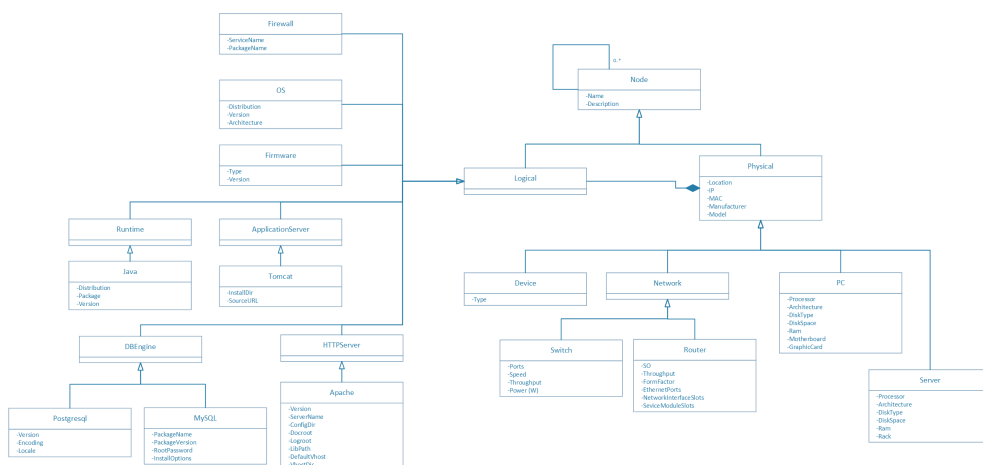


Figure 5.4: Representación UML de la Topología.

### 5.1.2 Modelado de la Configuración

En el caso de la configuración, como se discutió previamente, se tendrán distintos tipos. El tipo de configuración más básico es el de configuración Abierta (Open), que consiste simplemente de una descripción, ya sea a alto nivel o con opciones de configuración específicas de cualquier Nodo, se utiliza particularmente para Nodos que no poseen una configuración aplicable, y se desea tener un registro, por ejemplo los componentes Dispositivo.

Luego, se tiene la configuración de Registro (Registry), que aplica solamente a los nodos con Sistema Operativo Windows, y son configuraciones que se pueden aplicar a las entradas de dichos registros.

También se tiene configuración de tipo Archivo (File), que representa las diferentes acciones que se pueden realizar sobre los archivos del sistema.

Finalmente, se tiene una especialización para cada tipo de componente de software, los cuales se corresponderán con los componentes de Nodo Lógico que se mencionaron previamente. Cada instancia de configuración de estas herramientas representa una instancia del componente de software asociado a un componente del Nodo Lógico. De esta forma, por ejemplo, se puede tener una instancia de nodo Lógico de base de datos (DBEngine), el cual a su vez puede tener varias instancias de una base de datos, dicho de otra forma, un servicio de base de datos (MySQL) podrá tener varias instancias de base de datos (development, production). Lo mismo aplica para cada uno de los componentes de software, cada instancia de Nodo Lógico potencialmente puede tener asociadas varias instancias de Configuración.

Con el mismo análisis que se realizó para definir los atributos de los componentes Lógicos, se seleccionaron los atributos disponibles en los componentes de configuración. Los atributos

de cada componente representan las capacidades de configuración que brinda la herramienta de manejo de configuración (CMTs) para dicho componente.

Con base en lo anterior, el metamodelo final del componente de Configuración tendrá la forma de la Figura 5.5.

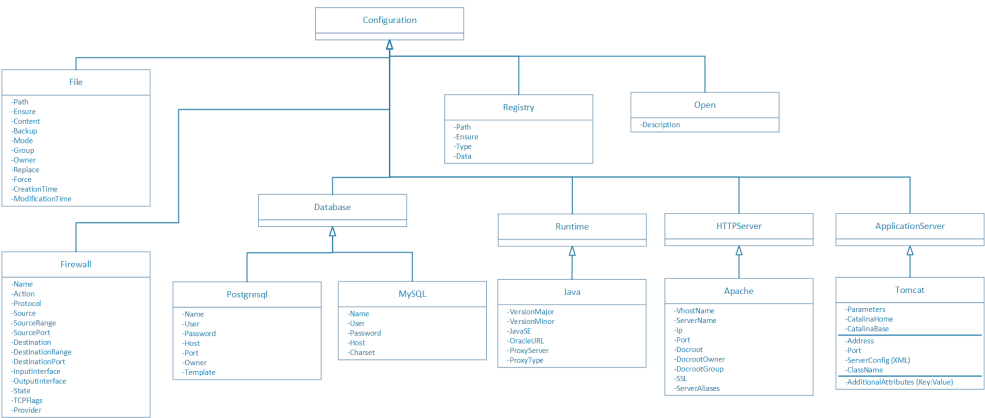


Figure 5.5: Representación UML de Configuración.

5.1.3 Modelado de la Solución

Una vez definidos los componentes de Topología y Configuración, que forman la solución objetivo, sólo resta unificar estos resultados en base a la relación de los mismos. En este caso, se tiene que los Nodos podrán tener varias instancias del componente de Configuración, por lo que dicha relación tendrá multiplicidad de cero a varios.

Por otro lado, también se debe considerar la posibilidad de tener un grupo de Nodos Físicos con su respectiva Configuración, esto es especialmente útil en los casos que se tienen conjuntos de dispositivos, y se desea que compartan una misma configuración. Para evitar que cada dispositivo necesite ser configurado individualmente, se agrega la multiplicidad de uno a varios con el componente de Configuración, de forma de que cada grupo de nodos pueda tener asociado una cantidad de instancias de Configuración, cumpliendo el propósito deseado.

El metamodelo que representará la solución propuesta se puede ver en la Figura 5.6. Se obviaron los componentes de bajo nivel mostrados previamente con el fin de representar de forma más clara la solución.



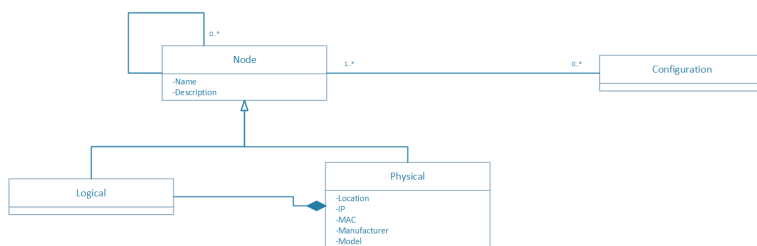


Figure 5.6: Representación UML de la solución propuesta.

## 5.2 Implementación del Perfil UML

Partiendo de la base del diagrama de la solución en la sección anterior, se puede pasar a la implementación del perfil UML. Para realizar dicha implementación se utilizará el entorno de modelado Papyrus, extensión de Eclipse.

Tal como se discutió en secciones previas, se utilizará el perfil para extender los componentes UML análogos a los presentados en la solución, por un lado se extenderá el componente **Device**, para poder representar los **Nodos Físicos**, y el componente **ExecutionEnvironment** para representar los **Nodos Lógicos**, y por otro lado se extenderá el componente **Artifact** representando los componentes de **Configuración**.

Las extensiones de los elementos UML se realizan utilizando estereotipos, que permiten agregar o crear nuevos elementos del modelo a partir de los existentes. Se necesitará crear un estereotipo para cada metaclase a extender, en este caso dichas metaclases son **Device**, **ExecutionEnvironment**, y **Artifact**, y los estereotipos a crear, para extender cada metaclase, serán **Physical**, **Logical**, y **Configuration**, respectivamente. Cabe notar que se obvió la implementación del estereotipo **Node**, que sí estaba presente en el diagrama de la solución analizada en la sección anterior. Esto se debe a que se consideró que dicha extensión no aportaba valor, dado que el elemento UML que se necesita extender, también llamado **Node**, posee la información necesaria, es decir, el nombre.

Para realizar la implementación de los estereotipos, basta con importar la metaclase a extender, y utilizar el componente de **Stereotype** que provee Papyrus para indicar el nombre y atributos de la extensión. Esto se realiza creando un proyecto Papyrus de tipo **Perfil UML**, que provee todas las herramientas necesarias para la implementación.

La extensión de los elementos mencionados previamente, utilizando estereotipos, se encuentra en la Figura 5.7.

Partiendo de esta base se continúa agregando el resto de los elementos de la solución, y dado que la implementación de cada estereotipo es independiente, se separa cada uno de ellos en una sección a continuación.

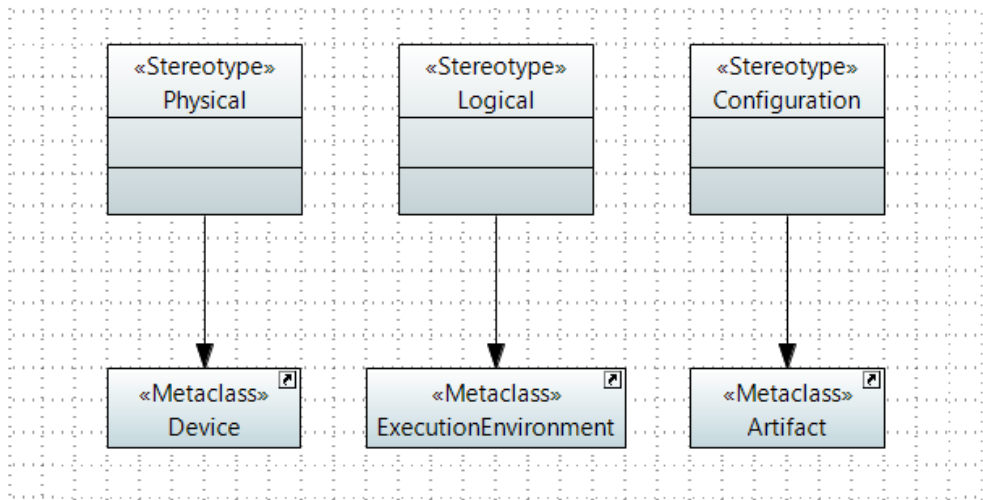


Figure 5.7: Estereotipos aplicados a Device, ExecutionEnvironment, y Artifact.

### 5.2.1 Nodos Físicos

En la implementación del estereotipo **Physical**, aplicado a la metaclase **Device**, se utilizan los elementos definidos previamente en el diagrama UML, estos son Servidor (**Server**), **PC**, Dispositivo (**Peripheral**), y los Dispositivos de Red (**NetworkDevice**), **Router** y **Switch**. Los elementos Peripheral, Server, Network Device, y PC son especializaciones del nodo Físico (**Physical**), y a su vez, Switch y Router son especializaciones de Network Device. Los atributos se agregan de acuerdo a como fueron seleccionados en el análisis de la solución, y se les asigna el tipo adecuado.

El diagrama de estereotipos definidos resultante se puede ver en la Figura 5.8.

### 5.2.2 Nodos Lógicos

En el caso de los nodos Lógicos, el estereotipo **Logical** es aplicado a la metaclase **ExecutionEnvironment**, esto permite que los nodos Lógicos puedan ser asignados a los nodos Físicos, al ser heredado de la relación entre Device y ExecutionEnvironment. De la misma forma que con los nodos Físicos, se utiliza como base el diagrama UML definido. Las especializaciones se realizan de la misma forma, en base a los elementos definidos con anterioridad. También se utilizan los atributos definidos, agregando para cada uno el tipo correspondiente, tal como se aprecia en la Figura 5.9.

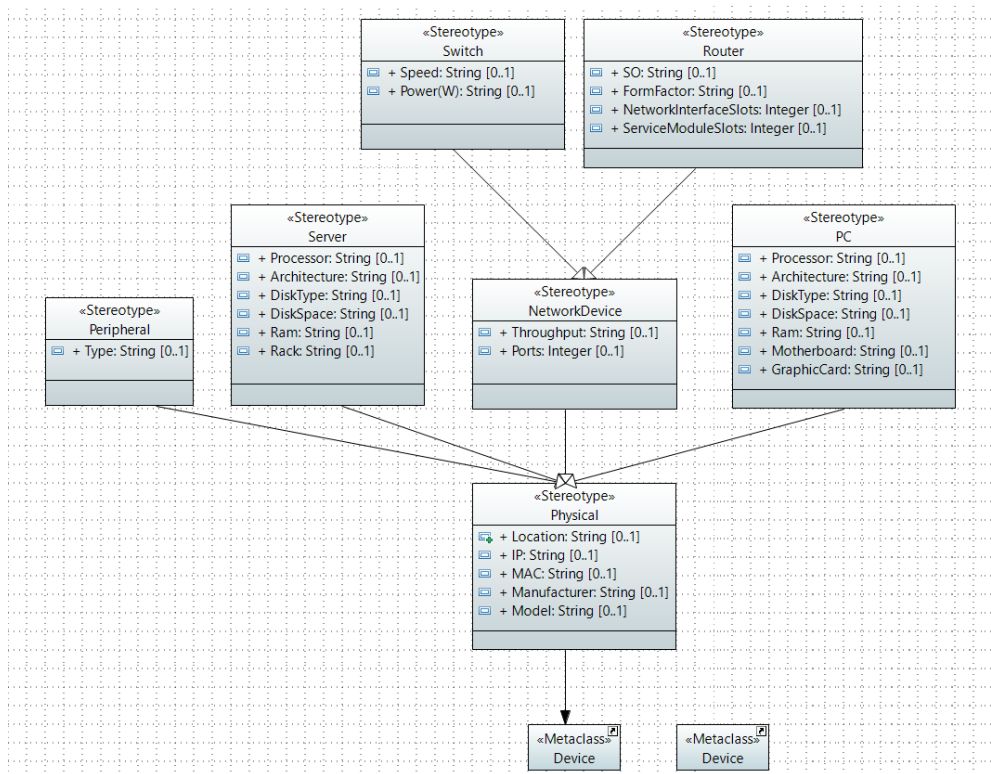


Figure 5.8: Diagrama de Estereotipos Physical.

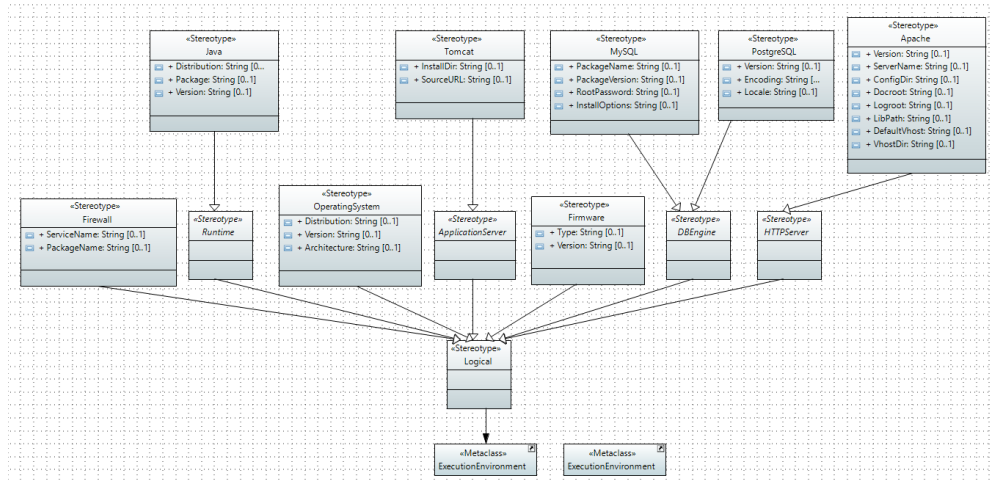


Figure 5.9: Diagrama de Estereotipos Logical.

### 5.2.3 Configuración

Finalmente se tiene la definición de Configuración, en este caso el estereotipo **Configuration** es aplicado a la metaclassa **Artifact**. De esta forma, una Configuración puede ser desplegada (*deployed*) en un nodo Lógico, ya que la relación de **deployment** es heredada por ser extensiones de las metaclases **Artifact**, y **ExecutionEnvironment** respectivamente.

La implementación del estereotipo se realiza a partir del análisis anterior, aunque en este caso es necesario realizar cambios con respecto al diagrama original. En dicho diagrama se puede ver que varios elementos bajo el nodo Logical comparten el mismo nombre con otros elementos bajo el metamodelo Configuration, por ejemplo, existen dos metamodelos con nombre Java. Con el fin de diferenciar dichos nombres en la definición de estereotipos, en este caso se cambian estos nombres:

- Apache se renombra a ApacheVhost
- Firewall se renombra a FirewallRule
- Java se renombra a JavaOracle
- Tomcat se renombra a TomcatApp
- Postgresql se renombra a PSQldb
- MySQL se renombra a MySQLDB

Esto expresa de mejor manera lo que representa cada elemento, y a su vez elimina cualquier lugar a confusión entre dichos elementos. Por esta razón también se modificó el nombre de configuración Libre, de **Open** a **FreeForm**, el cual resulta más expresivo.

Con este problema resuelto, se continúa con la implementación agregando los atributos correspondientes a cada estereotipo, con su tipo correspondiente. El resultado final del diagrama de estereotipos se encuentra en la Figura 5.10.

### 5.2.4 Restricciones

Como detalle final a la implementación del perfil, es necesario agregar las restricciones (**Constraints**) adecuadas. En este caso, las restricciones que se deben agregar son aquellas que relacionan la Configuración con los nodos Lógicos. Los elementos de Configuración que se corresponden a un componente de nodo Lógico, es decir, ApacheVhost, FirewallRule, JavaOracle, TomcatApp, PSQldb, y MySQLDB, deben ser desplegados sobre su correspondiente elemento de software, de forma de evitar que elementos incompatibles sean relacionados entre sí. Para ello se crea una restricción (Constrain) para cada uno de estos estereotipos, que explicita el tipo de estereotipo al cuál puede ser desplegado.

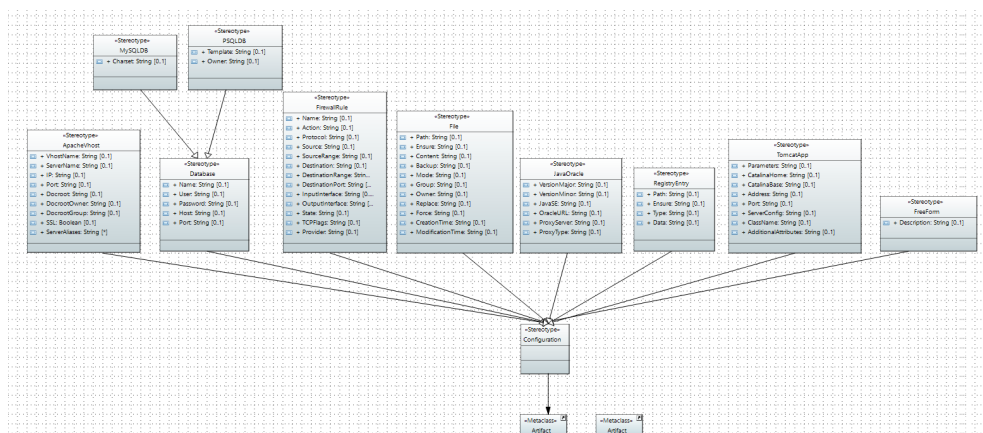


Figure 5.10: Diagrama de Estereotipos Configuration.

A continuación se presenta una descripción de las restricciones agregadas a cada componente de Configuración, asociado a su correspondiente componente Lógico:

- ApacheVhost: sólo puede ser desplegado a un componente lógico Apache
- MySQLDB: sólo puede ser desplegado a un componente lógico MySQL
- PSQDB: sólo puede ser desplegado a un componente lógico PostgreSQL
- FirewallRule: sólo puede ser desplegado a un componente lógico Firewall
- JavaOracle: sólo puede ser desplegado a un componente lógico Runtime
- TomcatApp: sólo puede ser desplegado a un componente lógico Tomcat

El resultado final del diagrama de estereotipos **Configuration**, incluyendo las restricciones, esta representado en la Figura 5.11.

### 5.2.5 Implementación del Perfil como plugin

Una vez realizado el perfil, es necesario instalarlo como plug-in, para ello se utiliza un generador de modelos EMF (Eclipse Modelling Framework). Primero se crea un proyecto EMF, utilizando como base el proyecto Papyrus y el perfil UML existentes. Esto generará los archivos descriptores Ecore y Genmodel basados en UML, que son las dos partes que componen un meta-modelo EMF. Luego, en base al Genmodel generado, se podrá generar el código del modelo para el perfil UML.

Finalmente, se agregan las extensiones necesarias para el plug-in, estas incluyen el paquete

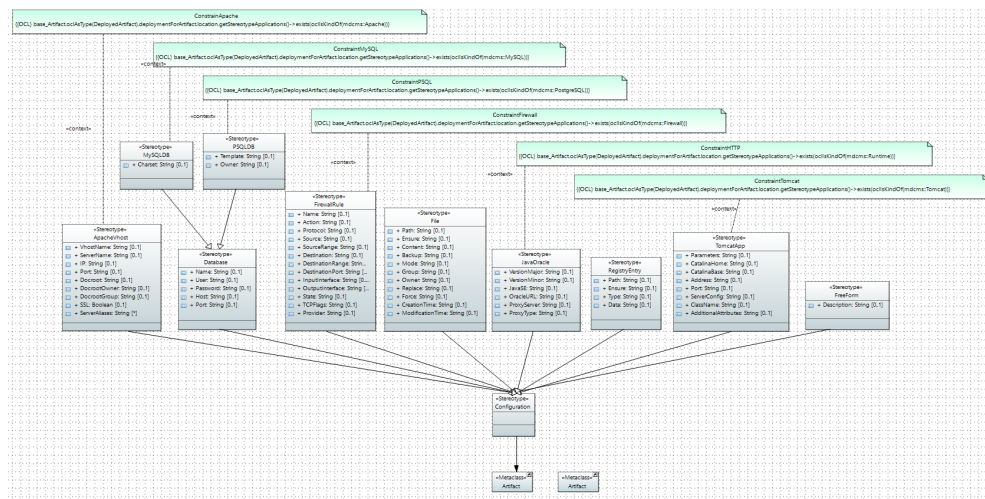


Figure 5.11: Diagrama de Estereotipos Configuration con restricciones.

Ecore generado, que se agrega automáticamente en el proceso de generación del modelo con EMF; un mapeo de URI para el modelo Ecore que indica el proyecto origen y el plug-in destino del proyecto; y los puntos de extensión (extension points) para el perfil UML, que posee el nombre que tendrá el perfil, así como la ubicación del perfil implementado (archivo `.profile.uml`). Esto se puede ver en la Figura 5.12.

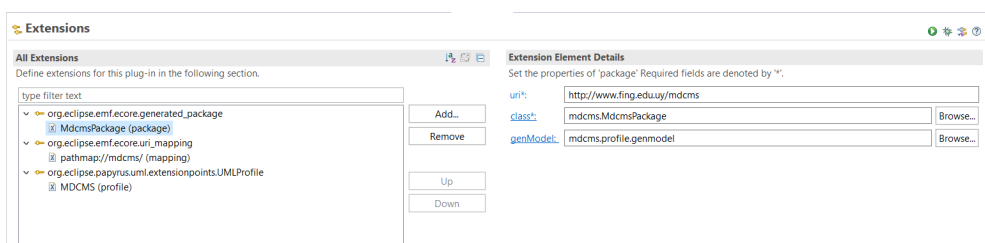


Figure 5.12: Extensiones necesarias para el perfil como plug-in.

Una vez definidos los pasos anteriores, simplemente se exporta el proyecto como plug-in, con la funcionalidad de Exportar que provee Eclipse para Desarrollo de Plug-ins, indicando la modalidad de exportación (directorio, archivo, o instalación local). (Figura 5.13)

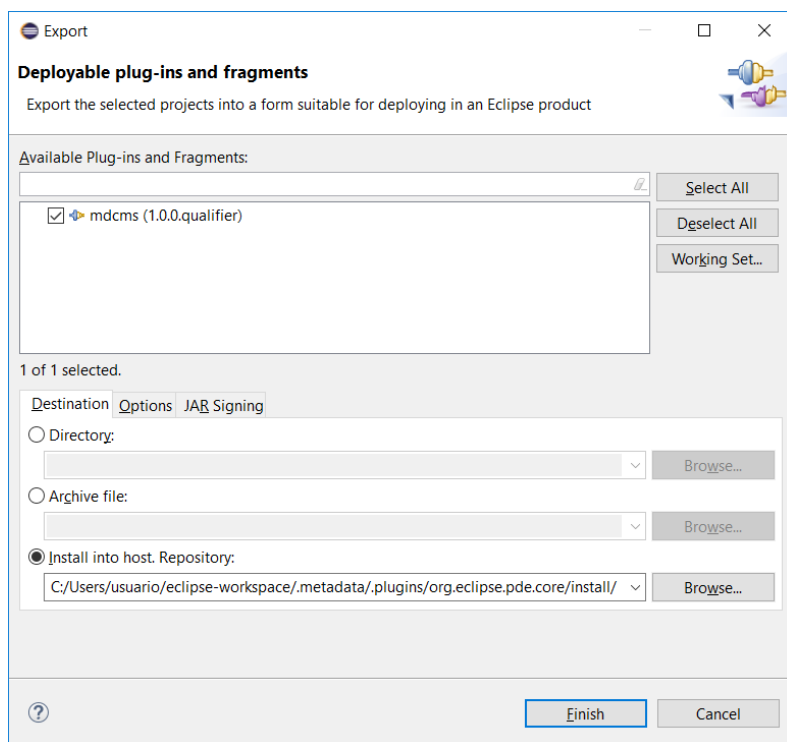


Figure 5.13: Exportación del proyecto como plug-in.

Luego de la instalación, al crear un nuevo modelo se tendrá la opción de aplicar el perfil UML creado, como se puede ver en la Figura 5.14.

## 5.3 Implementación de la Transformación

La transformación de modelo a texto se realizará utilizando la herramienta Acceleo, extensión de Eclipse, que permite generación de código en base a plantillas. Para los objetivos de este proyecto, la salida generada por la transformación fue dirigida a Puppet como CMT (Configuration Manager Tool) principal, esto quiere decir que los archivos de configuración generados tiene la sintaxis de dicha herramienta.

De igual forma que en capítulos anteriores, este se podría separar según nodo Físico, Lógico y Configuración, sin embargo, en este caso es conveniente hacer la distinción a partir del tipo de salida que se va a generar.

Por un lado se tienen los scripts de configuración que serán utilizados por un CMT, es decir,

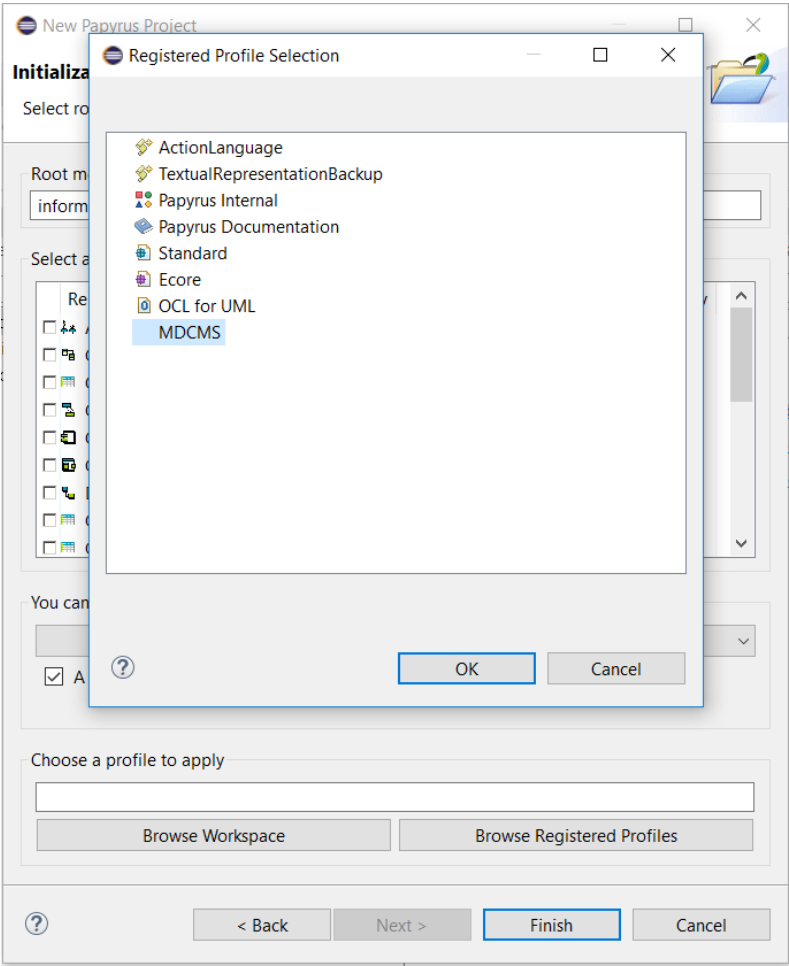


Figure 5.14: Aplicación del plug-in exportado.



los componentes de software que fueron analizados e incluidos en el diseño de la solución y definidos en el perfil UML; y por otro lado se tienen los textos planos que serán utilizados puramente como información, esto incluye, por ejemplo, los componentes físicos y la configuración de tipo Libre (FreeForm). Se realizará una transformación para cada una de estas clases, por un lado se tendrá la transformación **mdcms2puppet** que generará los archivos de configuración a ser utilizados por Puppet (con extensión **.pp**), mientras que los archivos informativos (con extensión **.txt**) serán generados por la transformación **mdcms2info**.

La salida generada por la transformación **mdcms2puppet** se organizó en base a lo anterior, y teniendo en cuenta la clase de elemento que genera dicha transformación. En primer lugar se genera un archivo Puppetfile que contiene los módulos que se necesitan instalar para la utilización en Puppet, y también un archivo **site.pp** (manifests/site.pp), que tendrá la información de los nodos existentes en la topología, esto será utilizado para declarar los diferentes nodos en Puppet, incluyendo la configuración correspondiente a cada uno.

En la Figura 5.15 se puede ver el modelo origen, el código de la transformación para esos elementos, y el resultado obtenido de dicha transformación.

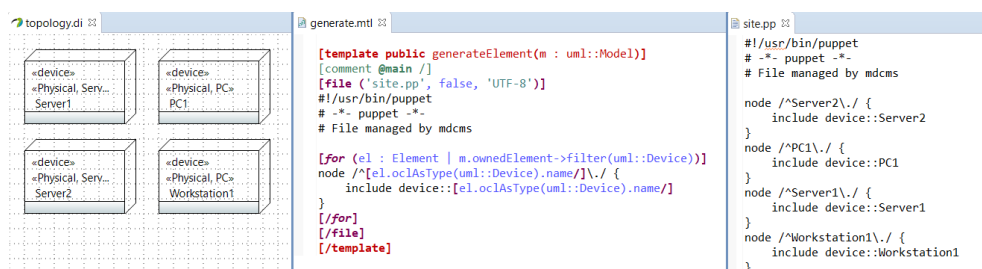


Figure 5.15: Ejemplo de un modelo origen, una transformación, y su salida correspondiente.

Tal como se puede apreciar, se genera un nodo de Puppet por cada componente en el modelo de origen, esto se debe a la funcionalidad en modo de plantilla de Acceleo, que permite obtener la salida de la forma que está plasmada en el archivo de generación.

Este archivo es el punto de partida de la configuración, la información de cada nodo, ya sea físico o lógico se encontrará bajo **modules/device/manifests**, mientras que los elementos de configuración serán creados en el directorio **modules/configurations/manifests**.

Por otro lado, para la transformación **mdcms2info**, bajo el directorio **Information** se encontrarán los archivos de texto plano, cuyo fin es puramente informativo, mientras que en el directorio **Configuration** se encontrarán los archivos generados a partir de los componentes FreeForm. Esta estructura facilita el manejo de dichos archivos, organizándolos en base a su comportamiento. En la Figura 5.16 se encuentra dicha estructura de archivos a alto nivel.

La transformación y estructura de estos elementos se analiza en las siguientes secciones.

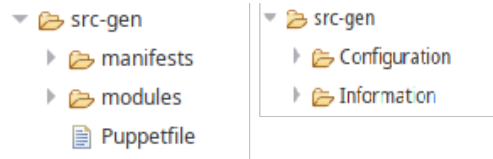


Figure 5.16: Estructura de archivos generada por cada transformación.

### 5.3.1 Generación de Información

Los elementos del modelo que pueden generar archivos de información son los siguientes:

- Cualquier nodo físico: Server, PC, Router, Switch, y Peripheral
- Dentro de los nodos lógicos solamente OperatingSystem y Firmware.
- En el caso de la configuración, FreeForm es el único que aplica.

Como se mencionó previamente, la generación de estos elementos se ubicará en el directorio **Information**, con la excepción de FreeForm que se encontrarán bajo **Configuration**. Para los primeros se creará un directorio por cada nodo Físico, y en dicho directorio se encontrará su información correspondiente, así como de todos los nodos lógicos que pertenezcan a este nodo. Esto se puede ver en la Figura 5.17.

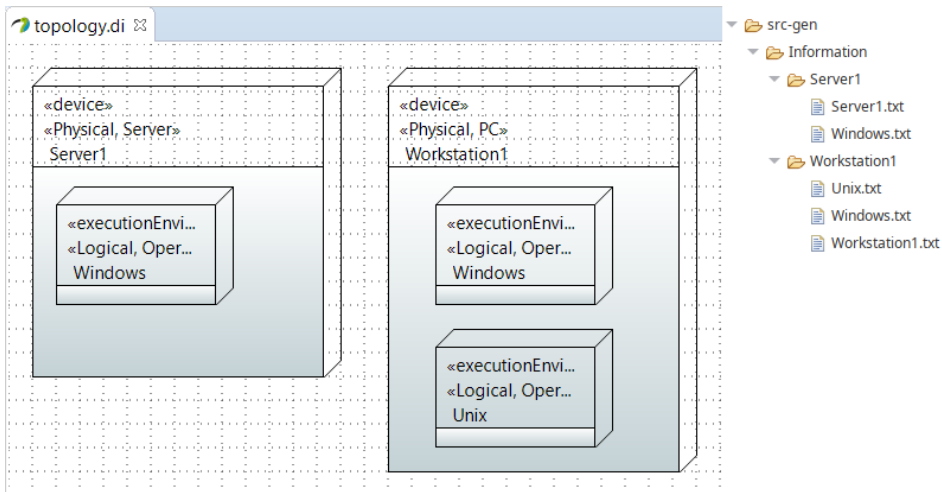
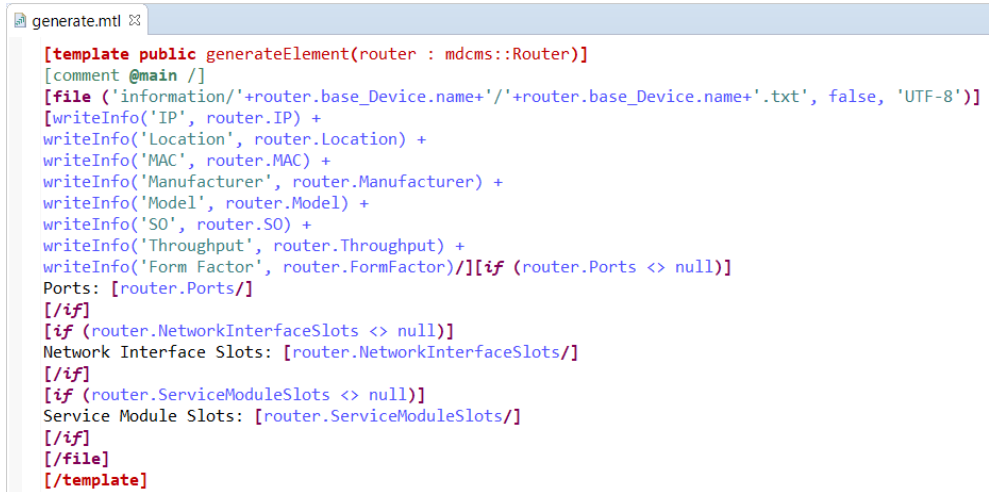


Figure 5.17: Ejemplo de generación de archivos de Información.

La generación de estos archivos de configuración se realiza en base a los atributos definidos en el modelo, gracias a la modalidad de transformación de Aceleio, fácilmente se pueden

definir los atributos de forma clave-valor en el archivo de texto, en este caso se utiliza una función auxiliar para dar este formato. El código utilizado para esta transformación se puede ver en la Figura 5.18.

The image shows a code editor window with a tab labeled 'generate.mtl'. The code is written in a syntax-highlighted language, likely Puppet DSL. It defines a template function 'generateElement' that takes a 'router' object as an argument. Inside the function, there is a comment '@main /', followed by a 'file' resource declaration to create an information file. The file's path is constructed using the router's base device name. The file's content is a series of 'writeInfo' calls for various attributes: IP, Location, MAC, Manufacturer, Model, SO, Throughput, and Form Factor. There is a conditional check for 'Ports' and another for 'NetworkInterfaceSlots'. The code ends with a closing tag for the template.

```
[template public generateElement(router : mdcms::Router)]
[comment @main /]
[file ('information/'+router.base_Device.name+'/'+router.base_Device.name+'.txt', false, 'UTF-8')]
[writeInfo('IP', router.IP) +
writeInfo('Location', router.Location) +
writeInfo('MAC', router.MAC) +
writeInfo('Manufacturer', router.Manufacturer) +
writeInfo('Model', router.Model) +
writeInfo('SO', router.SO) +
writeInfo('Throughput', router.Throughput) +
writeInfo('Form Factor', router.FormFactor)][if (router.Ports <> null)]
Ports: [router.Ports/]
[/if]
[if (router.NetworkInterfaceSlots <> null)]
Network Interface Slots: [router.NetworkInterfaceSlots/]
[/if]
[if (router.ServiceModuleSlots <> null)]
Service Module Slots: [router.ServiceModuleSlots/]
[/if]
[/file]
[/template]
```

Figure 5.18: Código de la generación del archivo de información para el elemento Router.

La Figura 5.19 muestra el resultado de la transformación del modelo presentado previamente (Figura 5.17).

Esta definición es análoga para todos los modelos que generan archivos de información, teniendo en mente que algunos elementos como PC y Server pueden generar tanto un archivo de configuración como de información, dependiendo de la transformación utilizada.

### 5.3.2 Generación de Scripts

La generación de scripts de configuración se realiza en forma análoga a los archivos de información, con la diferencia de que es necesario cumplir con la sintaxis de Puppet, de forma que la salida generada por la transformación pueda ser utilizada por dicho CMT (Configuration Manager Tool).

También es importante la estructura de directorios donde se ubicarán los archivos, esto se debe a que la inclusión de archivos debe ser consistente para que estos sean encontrados por el manejador.

Como se mencionó en capítulos anteriores, la distribución de los archivos de configuración consiste de un archivo principal llamado **site.pp** en la carpeta **manifests** (ubicada en la raíz), luego se tienen dos directorios, uno que incluye la información de los nodos físicos y lógicos

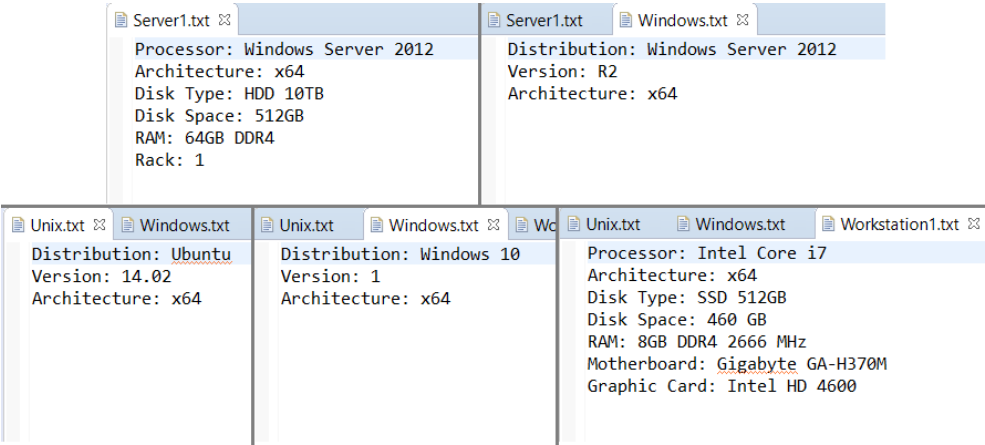


Figure 5.19: Resultado de la transformación del modelo en la Figura 5.17.

llamado **modules/device**, y otro que se corresponde con la configuración de dichos nodos, con el nombre **modules/configurations**. Estos directorios, a su vez, tendrán su propio directorio **manifests**.

En primer lugar se tiene el directorio **modules/device/manifests**, que contendrá un script con formato adecuado para Puppet por cada nodo Físico en el modelo, todos estos archivos tendrán el nombre del nodo y extensión **.pp**, cada archivo contendrá la declaración de la clase Puppet, e incluirá el resto de la configuración para el nodo.

Al mismo tiempo se creará un directorio por cada uno de estos nodos, con su nombre, incluyendo los archivos de configuración de Puppet correspondientes a los nodos Lógicos que posea dicho nodo Físico. Esto quiere decir que por cada nodo Físico, se tendrá un archivo de configuración para su propia instancia, y un directorio que contendrá un archivo de configuración por cada una de las instancias de nodo Lógico que tenga asociado.

En el ejemplo de la Figura 5.20 se puede ver la estructura de archivos generada a partir de un nodo Físico con sus respectivos nodos Lógicos:

En este ejemplo se puede ver la generación del archivo de configuración **Server1.pp**, que contiene la declaración del componente, e incluye al resto de los elementos que se encontrarán dentro del directorio **Server1**, un archivo de configuración por cada nodo Lógico. En la Figura 5.21 se puede analizar el contenido de dichos archivos generados.

Aquí se puede ver como el archivo de configuración del nodo Físico *Server1.pp* es el encargado de incluir el resto de los componentes Lógicos *MySQL\_Instance.pp*, *ApacheInstallation.pp*, y *Windows.pp*. Tanto el componente de *MySQL* como el de *Apache* genera su correspondiente archivo de configuración que será utilizado por el CMT, en el caso de *Windows* se encuentra vacío debido a que no le fueron asignadas configuraciones de tipo *File* o

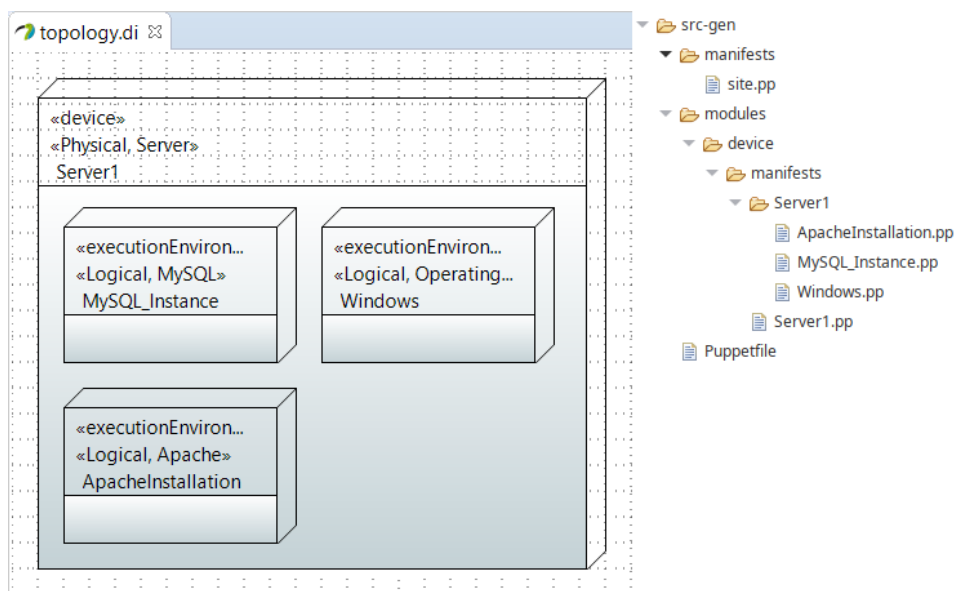


Figure 5.20: Ejemplo de generación de archivos de Configuración.

*Registry*, que puedan ser aplicadas.

Como se mencionaba previamente, algunos elementos generarán tanto un archivo de configuración, como uno de información en texto plano, en este caso los componentes de servidor (Server1) y sistema operativo (Windows) también generan estos archivos, en su correspondiente directorio (Figura 5.22).

La implementación de la transformación de los scripts se realizó de forma similar a los archivos de información, con el cuidado extra de mantener la sintaxis correcta de Puppet. Por ejemplo, el código de la transformación del archivo de configuración de Apache tiene la forma de la Figura 5.23.

De igual forma que con los archivos de información, se utiliza un método auxiliar para dar formato a los atributos, además de realizar la inclusión de los artefactos de configuración asociados a ese elemento.

Finalmente, se tiene la generación de los artefactos, o elementos de Configuración. En el ejemplo anterior se tenían nodos lógicos de MySQL, Apache, y Sistema Operativo, para ver esta última clase de elemento, se pueden agregar instancias de dichas configuraciones a cada uno de ellos. A modo de ejemplo, en la Figura 5.24 se agregó una instancia de base de datos MySQL, un host virtual de Apache, una configuración de archivo y una configuración de registro de Windows.

La transformación generada a partir de este modelo estará ubicada en el directorio **mod-**

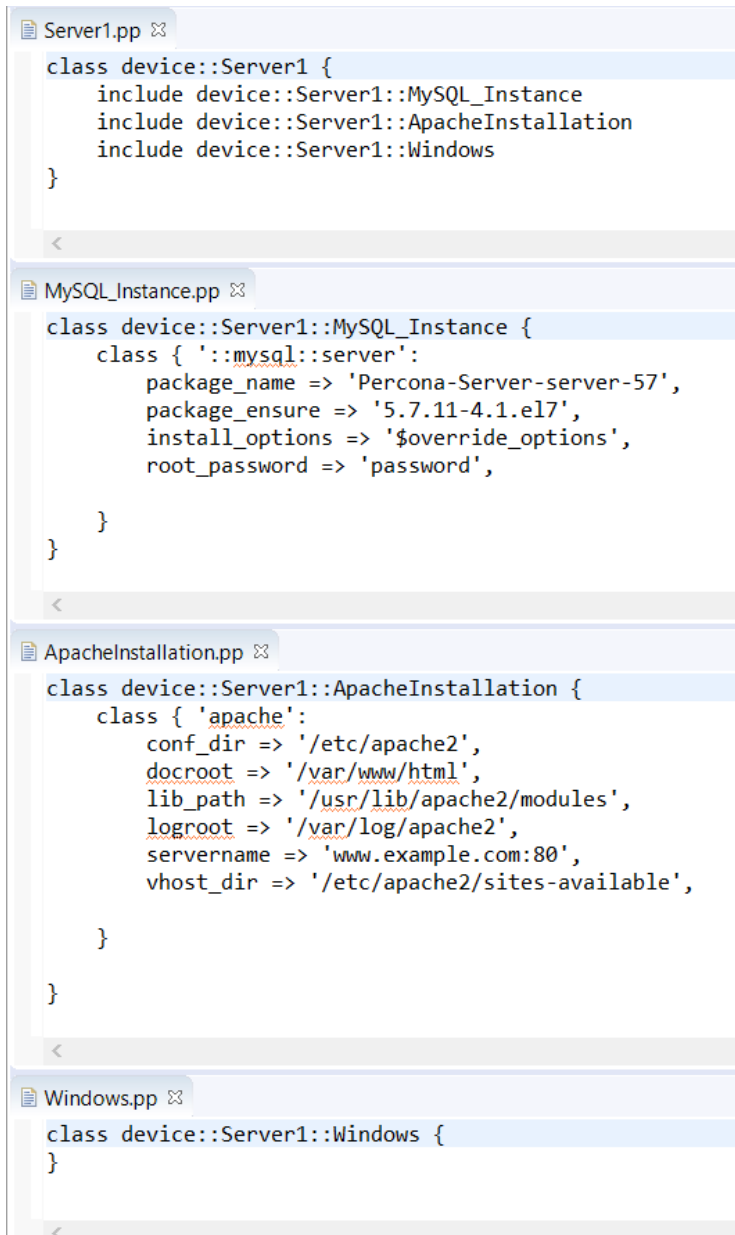


Figure 5.21: Resultado de la transformación del modelo en la Figura 5.20.

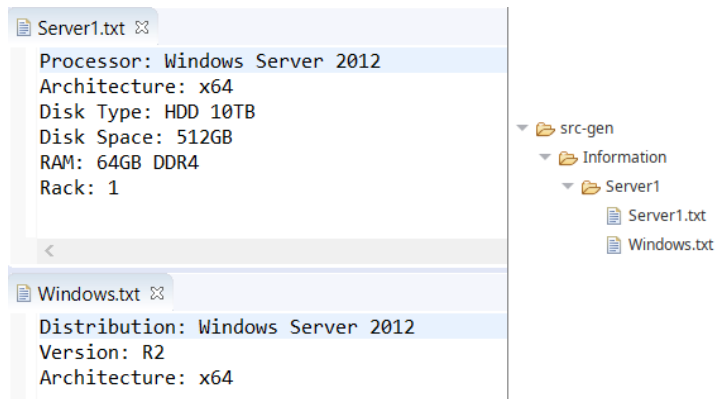


Figure 5.22: Archivos de información generados por los componentes servidor (Server1) y sistema operativo (Windows).

```
generate.mtl
[template public generateElement(apache : Apache)]
[comment @main/]
[file ('device/'+pcname(apache)+'/'+apache.base_ExecutionEnvironment.name+'.pp', false, 'UTF-8')]
class device::[pcname(apache)]::[apache.base_ExecutionEnvironment.name/] {
  class { 'apache':
    [writeAttribute('default_vhost', apache.DefaultVhost) +
      writeAttribute('version', apache.Version) +
      writeAttribute('conf_dir', apache.ConfigDir) +
      writeAttribute('docroot', apache.Docroot) +
      writeAttribute('lib_path', apache.LibPath) +
      writeAttribute('logroot', apache.Logroot) +
      writeAttribute('servername', apache.ServerName) +
      writeAttribute('vhost_dir', apache.VhostDir)/]
  }

  [for (apv : DeployedArtifact | apache.base_ExecutionEnvironment.deployment.deployedArtifact)]
  include configurations::apache::[apv.name/]
  [/for]
}
[/file]
[/template]
```

Figure 5.23: Código de la generación del archivo de configuración para el componente Lógico Apache.

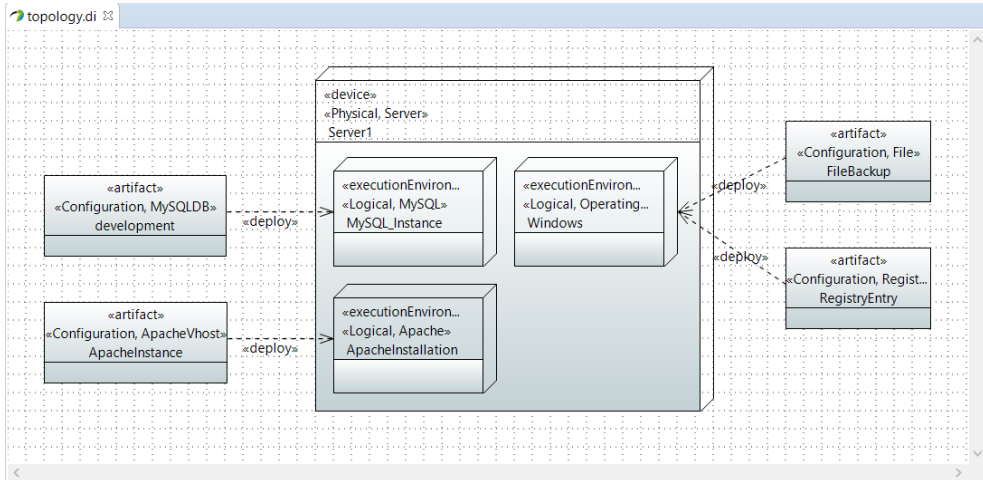


Figure 5.24: Ejemplo de modelado con elementos de Configuración.

**ules/configurations/manifests**, donde se creará un directorio por nodo lógico. Esta estructura se puede ver en la Figura 5.25.

Como se puede apreciar en la Figura 5.26, el directorio OS contiene un archivo por cada artefacto que tiene desplegado sobre sí mismo. Si se analizan los resultados obtenidos en los archivos de configuración se puede ver que el contenido es similar al de los nodos lógicos, esto se debe a que siguen la misma sintaxis para definir dichos componentes en el CMT.

Al mismo tiempo, si se analiza el código implementado para la transformación se puede ver que también es análogo al utilizado por el nodo lógico, por ejemplo, para el servidor virtual de Apache en la Figura 5.27.



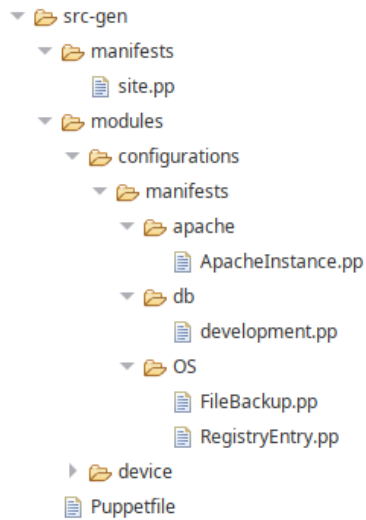


Figure 5.25: Ejemplo de generación de archivos para los componentes de la Figura 5.24.



Figure 5.26: Resultado de la transformación de los componentes de Configuración en la Figura 5.24.



```

generate.mtl
[template public generateElement(apv : mdcms::ApacheVhost)]
[comment @main /]
[file ('configurations/apache/'+apv.base_Artifact.name+'.pp', false, 'UTF-8')]
class configurations::apache::[apv.base_Artifact.name/] {
  apache::vhost { '[apv.base_Artifact.name/]':
    [writeAttribute('servername', apv.ServerName) +
      writeAttribute('vhost_name', apv.VhostName) +
      writeAttribute('ip', apv.IP) +
      writeAttribute('port', apv.Port) +
      writeAttribute('docroot', apv.Docroot) +
      writeAttribute('docroot_owner', apv.DocrootOwner) +
      writeAttribute('docroot_group', apv.DocrootGroup) +
      writeAttribute('ssl', apv.SSL.toString())/]
  }
}
[/file]
[/template]

```

Figure 5.27: Código de la generación del archivo de configuración para el componente de Configuración ApacheVhost.

Con la generación de archivos de configuración obtenida, lo único que resta para aplicar la configuración mediante Puppet es utilizar estos archivos como entrada de la herramienta. Para esto existen varias opciones, por ejemplo mover los archivos generados al directorio adecuado, configurar la herramienta de configuración para que utilice el directorio de salida por defecto, o simplemente seleccionar como directorio destino de la transformación el directorio adecuado para Puppet.

Para instalar dichas transformaciones en forma de plug-in es necesario generar un proyecto UI de Acceleo, lo cuál se puede realizar con la propia herramienta, seleccionando la opción Acceleo -> Create Acceleo UI Launcher Project, tal como se muestra en la Figura 5.28. Por último, simplemente se exporta cada proyecto, tanto la transformación como el proyecto UI, de la misma forma que se describe en la sección anterior para el perfil UML.

### 5.3.3 Resumen de transformaciones

A continuación se presentan a modo de resumen dos tablas que mapean los distintos elementos del modelo con su resultado luego de ejecutar cada transformación.

En la tabla 5.1 se pueden observar los elementos generados a partir de la transformación mdcms2puppet, mientras que en la tabla 5.2 se encuentran los archivos de información generados por la transformación mdcms2info.

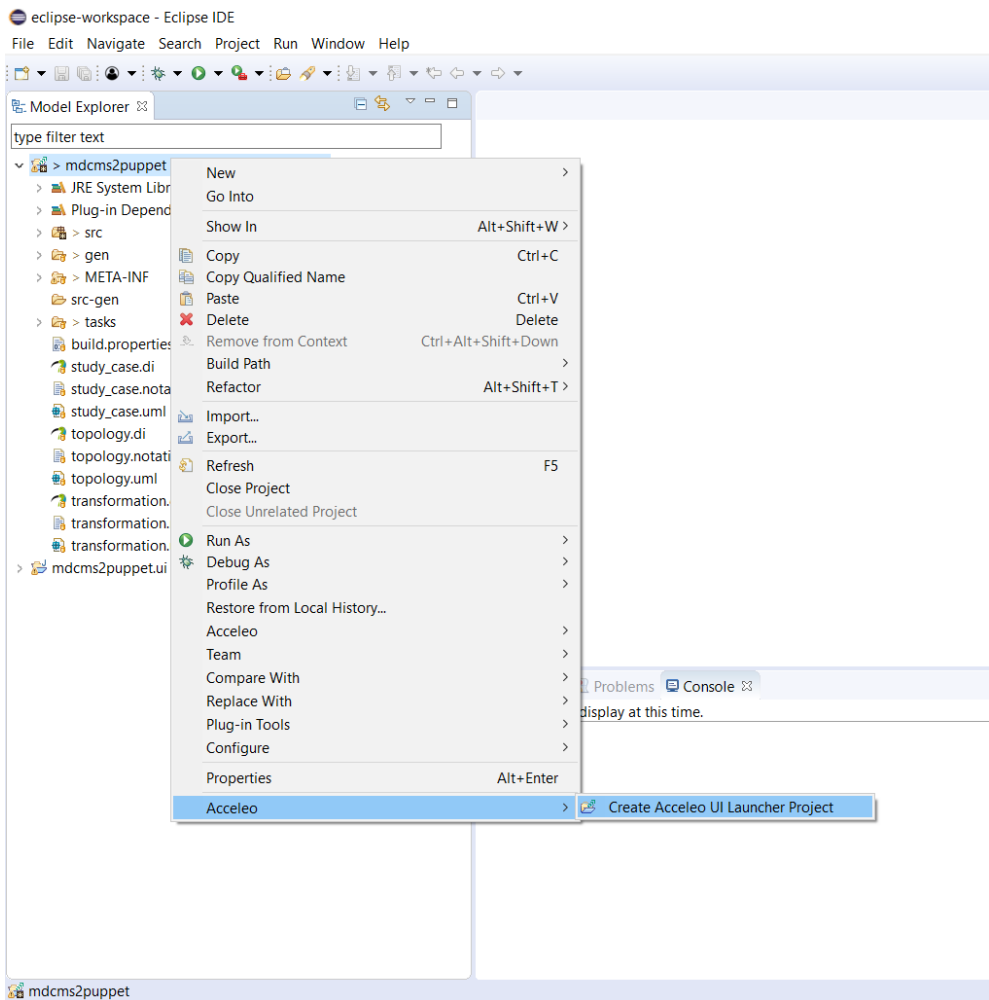


Figure 5.28: Creación de proyecto UI de Acceleo.

Elemento de modelo	Elemento generado en transformación Información
PC	Information/<pcname>/<pcname>.pp
Server	Information/<servername>/<servername>.pp
OperatingSystem	Information/<devicename>/<operatingsystemname>.pp
FreeForm	Configuration/<freeformname>.pp
Switch	Information/<switchname>/<switchname>.pp
Router	Information/<routername>/<routername>.pp
Peripheral	Information/<peripheralname>/<peripheralname>.pp
Firmware	Information/<firmwarename>/<firmwarename>.pp

Table 5.2: Elementos generados en transformación Información

Elemento de modelo	Elemento generado en transformación Puppet
PC	modules/device/manifests/<pcname>.pp
Server	modules/device/manifests/<servername>.pp
OperatingSystem	modules/device/manifests/<devicename>/<osname>.pp
Apache	modules/device/manifests/<devicename>/<apachename>.pp
Firewall	modules/device/manifests/<devicename>/<firewallname>.pp
Java	modules/device/manifests/<devicename>/<javaname>.pp
MySQL	modules/device/manifests/<devicename>/<mysqlname>.pp
PostgreSQL	modules/device/manifests/<devicename>/<postgresqlname>.pp
Tomcat	modules/device/manifests/<devicename>/<tomcatname>.pp
ApacheVhost	modules/configurations/manifests/apache/<vhostname>.pp
FirewallRule	modules/configurations/manifests/firewall/<firewallname>.pp
JavaOracle	modules/configurations/manifests/java/<javaoraclename>.pp
MySQLDB	modules/configurations/manifests/db/<dbname>.pp
PSQLDB	modules/configurations/manifests/db/<psqldbname>.pp
Tomcat	modules/configurations/manifests/tomcat/<tomcatname>.pp
RegistryEntry	modules/configurations/manifests/OS/<registryentryname>.pp
File	modules/configurations/manifests/OS/<filename>.pp

Table 5.1: Elementos generados en transformación Puppet

# 6

## Caso de Estudio

En esta sección se muestra un ejemplo de un caso de uso de la herramienta. Se mostrará tanto el modelo generado de una realidad planteada, como la salida generada de dicho modelo, explicando los detalles de los elementos y su resultado.

Para el caso de estudio se utilizó el diagrama de las Figuras 6.1 6.2, y 6.3. Este consiste de dos Routers conectados entre sí, por un lado se tiene la subred correspondiente a las PCs, y por otro lado se tienen los Servidores. En la subred de las PCs se tienen dos Switches que separan las computadoras con sistema operativo Unix (y una impresora), de las computadoras Windows. En el lado de los servidores se encuentra un Switch y dichos Servidores.

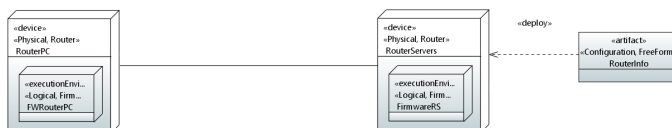


Figure 6.1: Ejemplo para el Caso de Estudio (alto nivel).

En la figura 6.4 se pueden observar las estructuras de archivos generadas por la ejecución de ambas transformaciones a partir de este diagrama. Como se puede ver, por una parte se tiene el archivo **manifests/site.pp**, el archivo **Puppetfile** y los directorios **modules/device** y **modules/configurations**, y por otra los directorios **Configurations** e **Information**. Dentro del directorio **modules/device** se encuentra un archivo de configuración por cada componente del tipo **Device** que sea configurable, y además su correspondiente directorio. Dentro del directorio **modules/configurations**, se encuentra un directorio por cada clase de componente que se encuentra en el diagrama, esta distribución permite identificar fácilmente las diferentes clases de componentes lógico. También se encuentra el directorio **Configurations** que contiene configuración del tipo **Libre** para cada componente. Por último se tiene el directorio **Information**, que contiene un directorio por cada componente de tipo **Device**, es decir cada componente físico, presente en el diagrama. Como es de esperar, estos directorios estarán

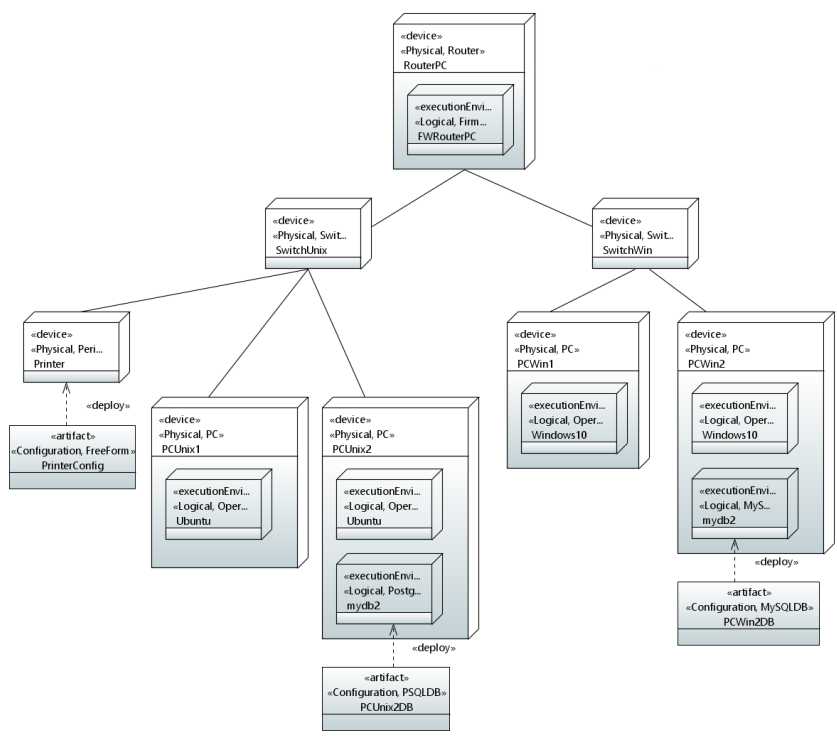


Figure 6.2: Ejemplo para el Caso de Estudio - RouterPC.

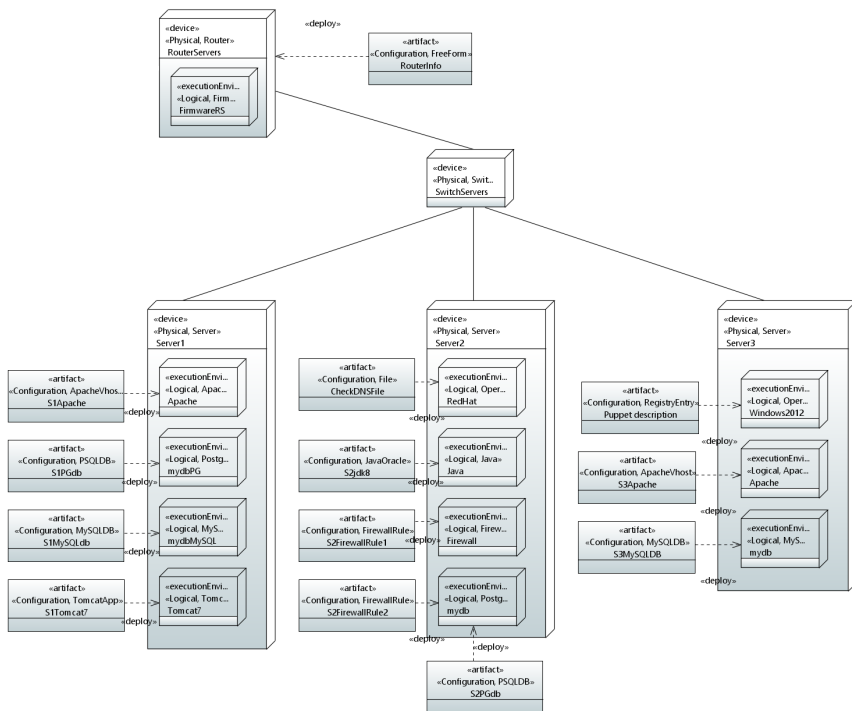


Figure 6.3: Ejemplo para el Caso de Estudio - RouterServers.

presentes con la condición de que exista su respectivo componente en el diagrama.

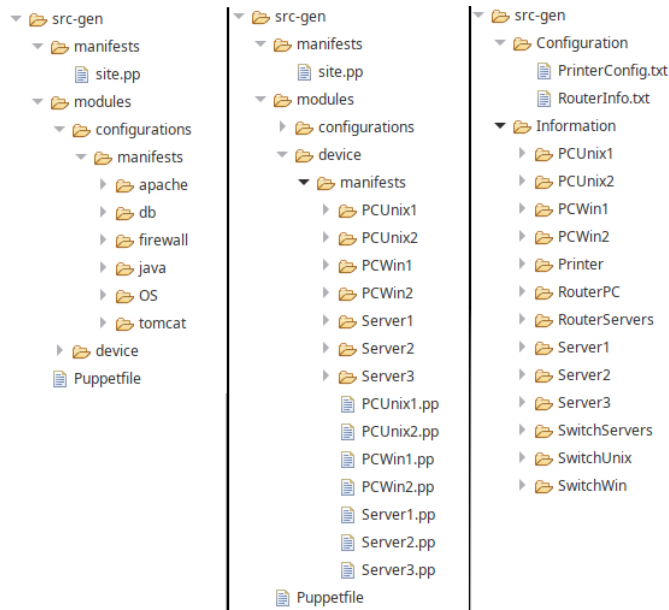


Figure 6.4: Estructura de archivos generada para el Caso de Estudio.

Teniendo presente la estructura de directorios generada, se puede pasar a analizar elementos en particular, y el resultado de la transformación generada. En la Figura 6.5 se tiene el componente de clase **device PCUnix2**, dicho componente tiene a su vez dos nodos lógicos, y existe un artefacto **configuration** aplicado a uno de estos nodos. Al ejecutar la transformación `mdcms2puppet` esto generará el archivo **PCUnix2.pp**, y un directorio con el mismo nombre, ubicado bajo el directorio **modules/device/manifests**. Al mismo tiempo, en el directorio **PCUnix2** se generará un archivo por cada nodo lógico presente, en este caso dos: **mydb2.pp** y **Ubuntu.pp** (dado que los componentes se crearon con estos nombres). Esta estructura se puede ver en la Figura 6.6. Luego, dado que se tiene un artefacto **configuration** de tipo **PSQLDB**, se creará bajo el directorio **/modules/configurations/manifests/db** un archivo con el nombre de dicho artefacto, en este caso **PCUnix2DB**, conteniendo la configuración de la base de datos. Al ejecutar la transformación `mdcms2info`, dentro del directorio **information** y bajo el directorio correspondiente a este componente (**PCUnix2**), se creará un archivo correspondiente al device (en este caso **PC**), y otro correspondiente al sistema operativo, con los nombres de dichos componentes. Los archivos de información se crearán dependiendo del componente, dado que sólo los que contienen información relevante serán creados (como se explicó en secciones previas). En la Figura 6.6 se pueden ver resaltados los archivos generados a partir del componente en la Figura 6.5.

Conociendo la estructura de directorios y los archivos generados, se puede pasar a analizar el contenido de estos archivos. En primer lugar se tiene el archivo principal **manifests/site.pp**,



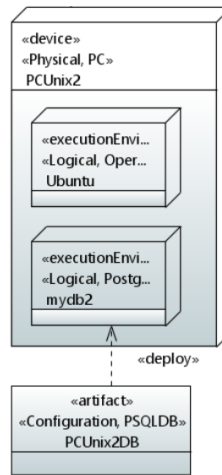
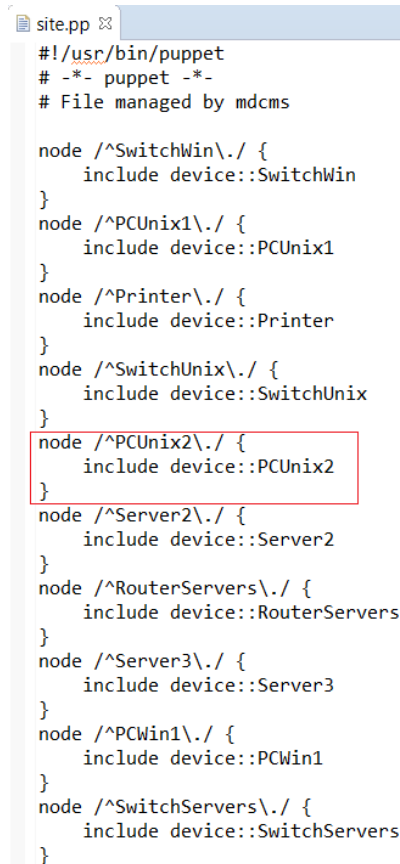


Figure 6.5: Componente de ejemplo PCUnix2.



Figure 6.6: Archivos generados a partir de la transformación del componente PCUnix2.

el cual se encarga de declarar los componentes que se definieron, incluyendo el nodo de ejemplo **PCUnix2**, como se puede ver en la Figura 6.7.



```
site.pp
#!/usr/bin/puppet
# -*- puppet -*-
# File managed by mdcms

node /^SwitchWin\. / {
    include device::SwitchWin
}
node /^PCUnix1\. / {
    include device::PCUnix1
}
node /^Printer\. / {
    include device::Printer
}
node /^SwitchUnix\. / {
    include device::SwitchUnix
}
node /^PCUnix2\. / {
    include device::PCUnix2
}
node /^Server2\. / {
    include device::Server2
}
node /^RouterServers\. / {
    include device::RouterServers
}
node /^Server3\. / {
    include device::Server3
}
node /^PCWin1\. / {
    include device::PCWin1
}
node /^SwitchServers\. / {
    include device::SwitchServers
}
```

Figure 6.7: Declaración de nodos en el archivo Site.pp.

Luego, se tienen los archivos de definición de cada nodo en particular bajo el directorio **modules/device/manifests**, en este caso tenemos el archivo base **PCUnix2.pp** que se encarga de incluir los componentes lógicos definidos, y sus dos archivos correspondientes, **mydb2.pp**, y **Ubuntu.pp**. Dichos archivos definen los componentes con los parámetros indicados, en caso de que el componente sea configurable. Los atributos utilizados en los componentes de **PCUnix2** se pueden ver en las Figuras 6.8, 6.9, 6.10, mientras que los archivos generados tendrán la forma que se muestra en la Figura 6.11.

En el lado de la configuración, tenemos el artefacto **PCUnix2DB** aplicado al nodo lógico **mydb2**, que a su vez está instalado en el nodo físico **PCUnix2**. Las propiedades de dicho artefacto se pueden ver en la Figura 6.12, mientras que el archivo de configuración generado se encuentra en la Figura 6.13. Como referencia, en la Figura 6.14 se incluye un ejemplo

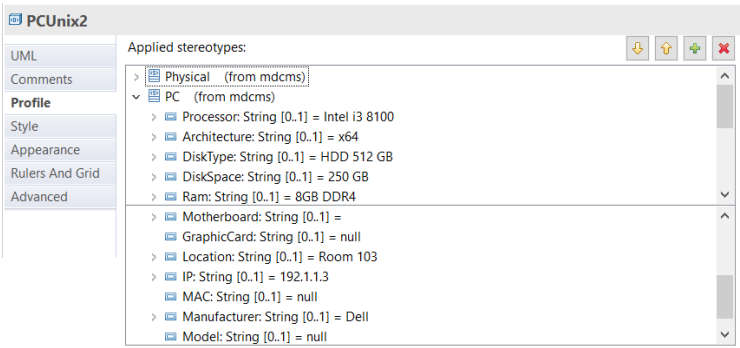


Figure 6.8: Propiedades del componente PCUnix2.

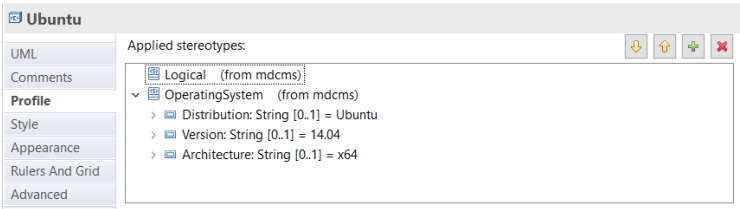


Figure 6.9: Propiedades del nodo lógico Ubuntu de PCUnix2.

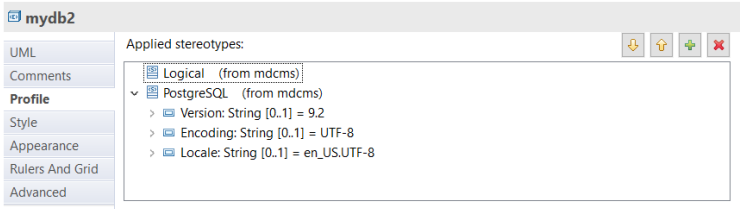


Figure 6.10: Propiedades del nodo lógico mydb2 de PCUnix2.

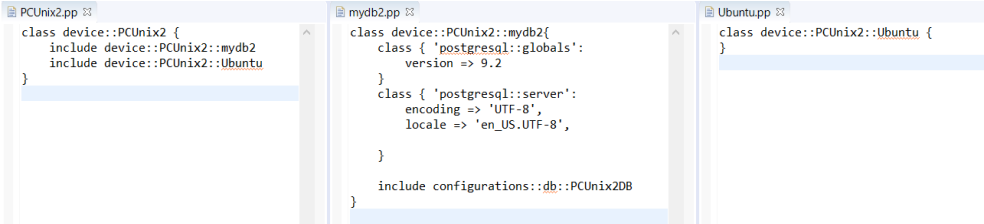


Figure 6.11: Archivos generados a partir de los componentes lógicos de PCUnix2.

simple de una configuración de este componente (Postgresql) utilizando Puppet, tomado de la definición del componente en la página web de Puppet. [Puppet, 2019]

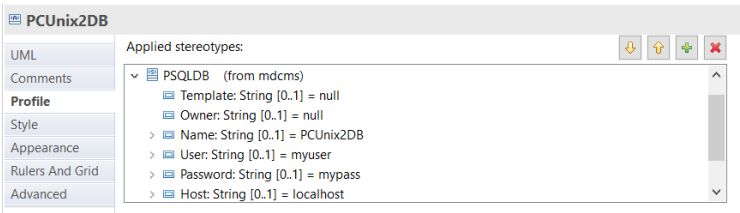


Figure 6.12: Propiedades del artefacto PCUnix2DB aplicado al nodo lógico mydb2 de PCUnix2.

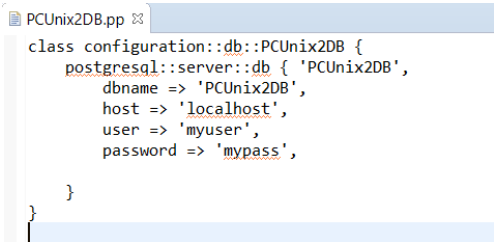


Figure 6.13: Archivo de configuración generado para el artefacto PCUnix2DB.



Figure 6.14: Archivo de configuración para PSQL en Puppet.

Por último tenemos los archivos de información generados para el componente **PCUnix2**, y su sistema operativo **Ubuntu**, las propiedades de dichos componentes se pueden ver en las Figuras 6.8 y 6.9 respectivamente. Los archivos de información que se generen dependerán de la clase de componentes que se definan en el modelo, dado que se generan únicamente para aquellos que sean relevantes, como se explicó previamente. Los archivos generados para los componentes mencionados anteriormente se pueden apreciar en la Figura 6.15.

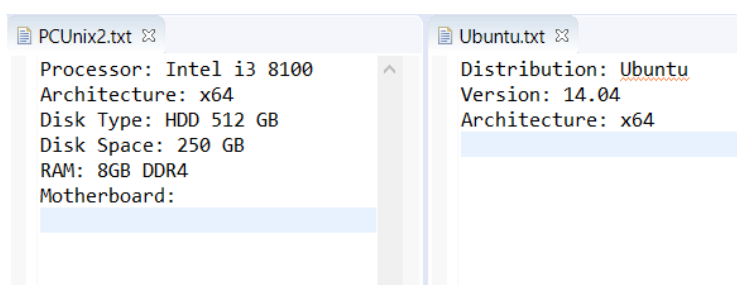


Figure 6.15: Archivos de información generados para los componentes PCUnix2 y Ubuntu.



# 7

## Conclusiones

---

En este capítulo se presentan conclusiones obtenidas del proyecto, analizando tanto las fortalezas como las debilidades que surgieron del mismo, y tomando como referencia los objetivos planteados al inicio. También se incluye una sección de trabajo futuro, que detalla las posibilidades de extensión que pueden colaborar en darle más forma al producto.

### 7.1 Fortalezas

A lo largo del trabajo se aprovecharon herramientas existentes para facilitar no sólo la construcción del prototipo sino para facilitar en un futuro la extensión del mismo en un producto más completo. El uso de UML trajo consigo una plétora de proyectos de software que permiten la manipulación y transformación de modelos, lo que permite un proceso más sencillo y menos propenso a errores comparado con el desarrollo de una herramienta de porte similar desde cero.

El uso de lenguajes y herramientas estándar en la industria también permite la integración con otras herramientas. En particular el uso de UML permitió hacer uso de Papyrus y no tener que crear un editor gráfico, el uso de Aceleo permitió que la generación de texto fuera más sencilla de lo que habría sido si se hubiera tenido que parsear XML y generar el mismo a partir de esta representación. En la sección 7.3 se explorarán más a fondo las posibilidades que el uso de herramientas estándar trae.

El uso de la herramienta facilita la administración de la configuración de la red, uniendo en un solo modelo la estructura física y la configuración de los nodos. La existencia de un lenguaje común también simplifica la comunicación entre miembros de un equipo, lo cual disminuye los problemas asociados a la misma.

Asimismo, una herramienta con un editor visual y una semántica sencilla baja considerable-

mente la barrera de acceso al uso de sistemas de manejo de la configuración. Arrastrar nodos, conectarlos, asignarles estereotipos y completar campos, todo desde una interfaz gráfica, es mucho menos amenazador que enfrentarse a un editor de texto en blanco y comenzar a escribir código puppet. Esperamos que esto permita el acercamiento de nuevos usuarios que se habrían visto intimidados por la aparente dificultad de utilizar un CMT.

Y es en esta posibilidad que creemos se encuentra el mayor valor del trabajo. Con el aumento de la complejidad de los entornos de trabajo, de la cantidad de servidores y workstations presentes y la creciente esencialidad del uso de los mismos en cualquier tipo de negocio, la posibilidad de perder un servidor o puesto de trabajo y poder reemplazarlo, con la seguridad de que el estado del mismo va a ser el adecuado es imperante. Es por esto que el uso de CMTs es cada vez más importante y la existencia de una herramienta que disminuye la dificultad de su utilización por parte de todos los miembros de un equipo de trabajo podría facilitar la implementación de los mismos.

## 7.2 Debilidades

La primera debilidad que se le puede observar al prototipo es la magra selección de elementos de configuración disponibles: una sola opción de servidor http, dos para manejadores de bases de datos, un solo servidor de aplicaciones, etc. Esto se podría subsanar con un sistema robusto de extensión, lo cual nos lleva a la siguiente debilidad.

La mayor debilidad del prototipo es la dificultad de extender el mismo, ya que para agregar algún nuevo elemento se debe definir el mismo a nivel de metamodelo. Esta tarea implica recrear el ambiente de desarrollo, tomar el código fuente y estudiarlo para comprender qué modificar, agregar el elemento y luego pasar a modificar la transformación con los cambios correspondientes, lo cual es muy trabajoso.

Finalmente, aún se necesita un conocimiento al menos básico del CMT a utilizar (Puppet) para poder hacer uso del código generado. Esto se podría subsanar brindando la instalación de los componentes relacionados a Puppet, esto se lograría creando un script que instale y configure un puppet master, insertando el código generado en el directorio correspondiente, y otro que se ejecute en los clientes, el cual instale y configure el puppet agent, apuntando al puppet master creado.

## 7.3 Trabajo Futuro

El primer trabajo a futuro (y quizás el más sencillo) podría ser agregar nuevos targets a la transformación. Esto es, permitir generar código no sólo para Puppet, sino para Chef, Ansible, o cualquiera de los nuevos sistemas de manejo de configuración que surgen día a día. Para esto sólo habría que escribir una nueva transformación para cada uno y agregarlas al plu-



gin generado, por lo que la mayor carga del trabajo se encontraría en encontrar una relación entre los diferentes elementos del modelo y los elementos correspondientes en el lenguaje utilizado por el sistema de configuración deseado.

De la mano de este trabajo se encuentra también la posibilidad de agregar otros tipos de salida, como pueden ser reportes de distintos aspectos de la red: un reporte de cuántas impresoras hay y las características de cada una, un reporte de a cuántos hosts sirve un router en particular, un reporte de cuántos switches en cascada se encuentran en la red, etc.

Otro trabajo podría ser continuar extendiendo el lenguaje de modelado y las transformaciones, agregando más elementos de configuración. Esto permitiría que un mayor número de usuarios encuentre en la herramienta una alternativa viable a un CMT tradicional. De todos modos, esto puede ser laborioso e ineficaz si el equipo de desarrollo es reducido o si el usuario final tiene requisitos muy específicos a su entorno; es por esto que el siguiente ítem se encuentra en este listado.

De forma similar al punto anterior, también se podría agregar la capacidad de definir los atributos de los componentes de forma dinámica, dicho de otra forma, que el usuario sea capaz de ingresar cualquier atributo disponible en Puppet para dicho componente, en lugar de depender de los que se hayan definido a nivel de metamodelo. Esto también evitaría la necesidad de actualizar el metamodelo cada vez que un componente de un CMT introduzca un nuevo atributo, o modifique uno existente, ya que el usuario podría, a nivel de modelo, modificar el atributo ingresado. Hay diferentes formas de lograr esto, la más sencilla sería permitir atributos de entrada libre por el usuario, y manejar dicho atributo a nivel de transformación. Sin embargo, hay que considerar que algunos Configuration Managers (Puppet por ejemplo) no admiten errores de sintaxis, por lo que en principio el usuario no podría ingresar un atributo que no este definido en el CMT, y se tendría que agregar algún tipo de validación, o permitir la eventual generación de scripts inválidos, lo que no es deseable.

Finalmente, el proyecto más ambicioso sería crear un mecanismo sencillo de extensión mediante el cual se puedan agregar elementos de configuración nuevos al producto de una manera sencilla. Una opción sería tomar un tipo de entrada textual, donde se indiquen las características del nuevo elemento de configuración como pueden ser el nombre, los atributos asociados, qué código nuevo se generaría a partir de la presencia del mismo en un modelo, y mediante una transformación, inyectarlo en el metamodelo y transformación, generando una nueva versión de producto.



## Bibliography

---

- [Ahmed, 2018] Ahmed, R. (2018). Overview - how ansible works.
- [Ambler, 2018] Ambler, S. W. (2018). Uml 2 deployment diagrams: An agile introduction.
- [Ansible, 2019] Ansible (2019). Overview - how ansible works.
- [Bézivin, 2005] Bézivin, J. (2005). On the unification power of models. *Software & Systems Modeling*, 4(2):171–188.
- [Biehl, 2010] Biehl, M. (2010). Literature study on model transformations. *Royal Institute of Technology, Tech. Rep. ISRN/KTH/MMK*, 291.
- [Binkert et al., 2006] Binkert, N. L., Dreslinski, R. G., Hsu, L. R., Lim, K. T., Saidi, A. G., and Reinhardt, S. K. (2006). The m5 simulator: Modeling networked systems. *Ieee micro*, (4):52–60.
- [Bjorklund, 2010] Bjorklund, M. (2010). Yang-a data modeling language for the network configuration protocol (netconf). Technical report.
- [Brambilla et al., 2012] Brambilla, M., Cabot, J., and Wimmer, M. (2012). Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182.
- [Chef, 2019] Chef, L. (2019). An overview of chef.
- [Clark et al., 2015] Clark, T., Sammut, P., and Willans, J. (2015). Applied metamodeling: a foundation for language driven development. *arXiv preprint arXiv:1505.00149*.
- [Cotton, 2016] Cotton, B. (2016). What are configuration management tools?
- [Da Silva, 2015] Da Silva, A. R. (2015). Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139–155.
- [Diagrams, 2018a] Diagrams, U. (2018a). Deployment diagrams overview.
- [Diagrams, 2018b] Diagrams, U. (2018b). Uml profile.

- [Española, 2017] Española, R. A. (2017). Modelo. en diccionario de la lengua española.
- [Essaadi et al., 2017] Essaadi, F., Maissa, Y. B., and Dahchour, M. (2017). Mde-based languages for wireless sensor networks modeling: A systematic mapping study. In *Advances in Ubiquitous Networking 2*, pages 331–346. Springer.
- [Foundation, 2019a] Foundation, E. (2019a). Acceleo.
- [Foundation, 2019b] Foundation, E. (2019b). Atl - a model transformation technology.
- [Foundation, 2019c] Foundation, E. (2019c). Atl/user guide - overview of the atl language.
- [Foundation, 2019d] Foundation, E. (2019d). Papyrus modelling environment.
- [Group, 2015] Group, O. M. (2015). Uml.
- [Group, 2018] Group, O. M. (2018). Mof model to text transformation language, v1.0.
- [Higuera Castro et al., 2016] Higuera Castro, G. A. et al. (2016). Metamodelo para configuraciones en dispositivos de redes como estándar soportado en la ingeniería dirigida por modelos.
- [IBM, 2019] IBM (2019). Deployment diagrams.
- [Kent, 2002] Kent, S. (2002). Model driven engineering. In *International Conference on Integrated Formal Methods*. Springer.
- [Kleppe et al., 2003] Kleppe, A. G., Warmer, J., Warmer, J. B., and Bast, W. (2003). *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional.
- [Mens and Van Gorp, 2006] Mens, T. and Van Gorp, P. (2006). A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142.
- [Paradigm, 2018] Paradigm, V. (2018). What is profile diagram in uml?
- [Puppet, 2018] Puppet (2018). The forrester wave<sup>TM</sup>: Configuration management software for infrastructure automation, q4 2018.
- [Puppet, 2019] Puppet (2019). Puppet - postgresql.
- [SolarWinds, 2019] SolarWinds (2019). Solarwinds: Network topology mapper.
- [Steinberg et al., 2008] Steinberg, D., Budinsky, F., Merks, E., and Paternostro, M. (2008). *EMF: eclipse modeling framework*. Pearson Education.
- [Tangmunarunkit et al., 2002] Tangmunarunkit, H., Govindan, R., Jamin, S., Shenker, S., and Willinger, W. (2002). Network topology generators: Degree-based vs. structural. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 147–159. ACM.

---

[Yigal, 2017] Yigal, A. (2017). Chef vs. puppet: Methodologies, concepts, and support.