Universidad de Murcia

Departamento de Ingeniería de la Información y las Comunicaciones

Modelado y Definición de un Lenguaje de Políticas Enriquecido Semánticamente para la Gestión de la Seguridad

Tesis Doctoral

Presentada por:
Félix Jesús García Clemente

Supervisada por:
Dr. Antonio Fernando Gómez Skarmeta
Dr. Gregorio Martínez Pérez

Murcia, Septiembre de 2006

D. Antonio Fernando Gómez Skarmeta, Profesor Titular de Universidad del Área de Ingeniería Telemática en el Departamento de Ingeniería de la Información y las Comunicaciones de la Universidad de Murcia, y D. Gregorio Martínez Pérez, profesor Ayudante de Facultad de Segundo Periodo del Área de Ciencia de la Computación e Inteligencia Artificial en el Departamento de Ingeniería de la Información y las Comunicaciones de la Universidad de Murcia, AUTORIZAN:

La presentación de la Tesis Doctoral titulada "Modelado y Definición de un Lenguaje de Políticas Enriquecido Semánticamente para la Gestión de la Seguridad", realizada por D. Félix Jesús García Clemente, bajo nuestra inmediata dirección y supervisión, en el Departamento de Ingeniería de la Información y las Comunicaciones, y que presenta para la obtención del grado de Doctor por la Universidad de Murcia.

En Murcia, a 5 de Septiembre de 2006

D. Antonio Fernando Gómez Skarmeta D. Gregorio Martínez Pérez

D. Antonio Fernando Gómez Skarmeta, Profesor Titular de Universidad del Área de Ingeniería Telemática y Director del Departamento de Ingeniería de la Información y las Comunicaciones de la Universidad de Murcia, INFORMA:

Que la Tesis Doctoral titulada "Modelado y Definición de un Lenguaje de Políticas Enriquecido Semánticamente para la Gestión de la Seguridad", ha sido realizada por D. Félix Jesús García Clemente, bajo la inmediata dirección y supervisión de D. Antonio Fernando Gómez Skarmeta y D. Gregorio Martínez Pérez, y que el Departamento ha dado su conformidad para que sea presentada ante la Comisión de Doctorado.

En Murcia, a 7 de Septiembre de 2006

D. Antonio Fernando Gómez Skarmeta

Resumen

En la administración de sistemas distribuidos es esencial una gestión automática que asegure un funcionamiento correcto del sistema y además permita una administración descentralizada que no limite la dinamicidad y la extensión de las organizaciones que dependen de estos sistemas distribuidos.

En el camino de conseguir una solución para la gestión automática de sistemas, un nuevo paradigma denominado Gestión Basada en Políticas ha sido utilizado de manera extendida y actualmente es la base de las principales propuestas en el ámbito de la gestión, además de ser la base del trabajo de investigación que se presenta en esta tesis. En este nuevo paradigma, una Política se define como un conjunto de reglas o prácticas que regulan dinámicamente el comportamiento del sistema de manera autónoma frente a los elementos gestionados. Así una de las principales metas de la Gestión basada en Políticas es alcanzar el control y la gestión de la red, sus aplicaciones y servicios en una capa de abstracción de alto nivel. En este sentido el administrador determina las reglas que describen las políticas usando un lenguaje específico para ello y es entonces la arquitectura de gestión la que proporciona soporte para transformar y distribuir las políticas a cada dispositivo y teniendo como último objetivo aplicar una configuración consistente en cada uno de ellos.

Por su parte la gestión de la seguridad basada en políticas es un campo de investigación que aún necesita de grandes esfuerzos para conseguir una solución final al problema de la provisión automática de seguridad. La base de esta solución se encuentra en resolver los problemas que en la actualidad tienen los lenguajes de especificación de políticas entre los que se encuentran la falta de un modelo común que garantice la extensibilidad y la interoperabilidad entre diferentes sistemas de gestión, y las carencias en su diseño para facilitar el análisis de conflictos entre políticas y el acceso a la propia información de las políticas.

En este sentido esta tesis intenta avanzar en la solución de estos problemas proponiendo un nuevo lenguaje de políticas para la gestión de la seguridad. Este lenguaje de políticas está basado en el modelo CIM que proporciona

un modelo de información estandarizado que define todo un conjunto de conceptos relativos a la gestión de redes, usuarios y servicios. En nuestra propuesta definimos una representación formal de los conceptos e instancias del modelo CIM y una representación lógica de las reglas de políticas. El uso de estas representaciones permiten el uso de técnicas de razonamiento sobre las políticas que facilitan el acceso a la información de la política y el análisis de conflictos entre las políticas.

La flexibilidad que proporciona el uso del modelo CIM (Common Information Model) permite que nuestra propuesta pueda definir diferentes tipos de políticas de seguridad. En nuestra propuesta presentamos a modo de ejemplo: políticas de autenticación, políticas de autorización, políticas de delegación, políticas de obligación, políticas de filtrado y políticas de seguridad IP.

Además se presenta una arquitectura de gestión donde nuestro lenguaje de políticas se implementa y se integra para la definición de políticas de seguridad en diferentes entornos de aplicación. Así se muestra como nuestro lenguaje de políticas aporta una solución válida y adecuada a los problemas antes planteados.

A Manoli ...
A mis padres ...

Agradecimientos

Mi más sincero agradecimiento a Antonio, Gregorio, Óscar y Gabi que son la principal referencia en mi vida profesional en la Universidad de Murcia. Vuestra confianza y apoyo durante estos años es un orgullo y una garantía de seguridad.

También mi agradecimiento a todos los que me habéis acompañado estos años, en especial a la gente del grupo ANTS y a los profesores del departamento DITEC.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo, que sirve de introducción y motivación para el resto de la tesis doctoral, se pretende mostrar la evolución de los sistemas de gestión de los sistemas distribuidos de comunicaciones y, en particular, como la gestión de la Seguridad ha sido siempre tratada como un área fundamental en la definición de las principales propuestas. También es intención de este capítulo el definir de manera clara los problemas que intenta resolver esta tesis doctoral junto con las aportaciones de mayor importancia que se han realizado.

Este capítulo se aborda dando una visión general de las diferentes propuestas para la gestión de los sistemas distribuidos, pero prestando un especial interés a las tecnologías y los paradigmas utilizados en la gestión de la Seguridad, de lo cual se extraerán una relación de deficiencias a las cuales se les va a tratar de dar solución en los siguientes capítulos. Se concluirá el capítulo con el resumen de las aportaciones de mayor importancia asociadas a esta labor de investigación, y finalmente la especificación de la estructura del documento que aquí se presenta.

1.1. Gestión de Sistemas Distribuidos de Comunicaciones

La gestión de sistemas distribuidos puede ser definida como la tarea de mantener el correcto funcionamiento de un sistema acorde con una serie de reglas de comportamiento determinadas por el administrador o gestor del sistema. Para ello es esencial una gestión automática fundamentalmente por dos razones. La primera es que en muchos casos el sistema no está controlado desde un punto central, sino que tiene que ser mantenido por el esfuerzo cooperativo de varios administradores independientes. La segunda

consiste en que los sistemas distribuidos son altamente complejos ya que tienen diversos componentes suministrados por diferentes fabricantes; además pueden estar distribuidos a través de áreas geográficas lejanas con diferentes franjas horarias y autoridades reguladoras, además de poder contener cientos de miles de recursos y ser usados por miles de usuarios. En definitiva, las organizaciones dependen de los sistemas distribuidos para funcionar y una gestión automática pretende asegurar que funcionarán adecuadamente.

El primer esfuerzo para hacer posible la gestión automática de sistemas heterogéneos, y sobrellevar la complejidad asociada a este proceso, fue realizado por ISO a través de un conjunto de estándares. En este sentido destaca OSI Systems Management (OSI-SM) [78] que hace la primera división del conjunto de la gestión en áreas de responsabilidad funcional conocidas con el acrónimo inglés FCAPS (Fault, Configuration, Accounting, Performance, Security). Estas áreas funcionales son:

- *Gestión de la Configuración* para controlar la instalación de los componentes tanto hardware como software dentro de un sistema distribuido o de una aplicación.
- Gestión del Rendimiento, centrado en la optimización del rendimiento para mejorar el servicio proporcionado a los usuarios en términos de mejor tasa de transferencia, tiempos de respuesta o fiabilidad, o para reducir costes operativos.
- *Gestión de Fallos*, incluyendo la detección, localización y recuperación de fallos.
- *Gestión de Registro*, que registra la información de uso de los recursos y permite proveer servicios de cobro por el uso de estos servicios.
- *Gestión de la Seguridad* para mantener los mecanismos de seguridad del sistema, por ejemplo el control de acceso, los sistemas de cifrado y la seguridad física.

Posteriormente al trabajo de OSI y avanzando en la organización de la gestión automática, el organismo normativo ITU-T desarrolló el modelo TMN (Telecommunications Management Network) [55] como el framework para soportar la gestión abierta de redes y servicios de telecomunicaciones. Este modelo define una organización para la gestión de sistemas basada en capas. Para ello usa un paradigma distribuido jerárquico y toma como base tecnológica OSI-SM [54] como puede verse en la figura 1.1. El modelo TMN está compuesto por cuatro capas, estas son:

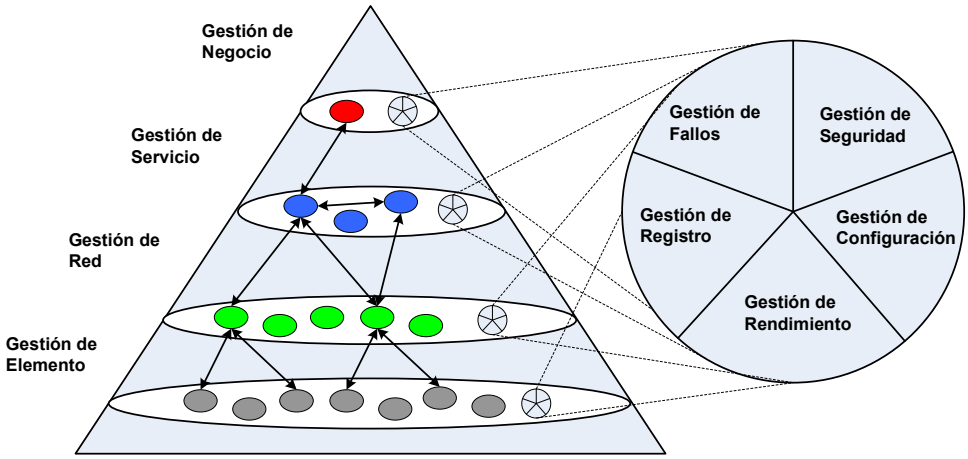

Figura 1.1: *Modelo TMN/OSI-SM*

- La capa de *gestión de negocio* que realiza las funciones relacionadas con los aspectos de negocio, analiza tendencias y cuestiones de calidad, por ejemplo, para proporcionar una base para la facturación y otros informes financieros.

- La capa de *gestión de servicio* que realiza funciones para el manejo de servicios en la red: definición, administración y tarificación de los servicios.

- La capa de *gestión de red* realiza las funciones para la distribución de los recursos de red: configuración, control y supervisión de la red.

- La capa de *gestión de elemento* contiene funciones para el manejo de elementos individuales de la red. Esto incluye gestión de alarmas, manejo de información, backup, registro, y mantenimiento de hardware y software.

El resultado del modelo TMN/OSI-SM fue una solución centrada en la especificación de arquitecturas de gestión de manera general; sofisticada y potente, pero también complicada y costosa de desarrollar; de tal manera que sólo encontró uso en entornos de aplicación de telecomunicaciones.

En el camino de conseguir una solución menos compleja para la gestión automática de sistemas, OMG CORBA [76] busca una solución pragmática basada en objetos distribuidos pero dirigiendo sus esfuerzos de estandarización al nivel de aplicación. De manera que OMG CORBA se preocupa fundamentalmente de los sistemas de software distribuido definiendo todo un conjunto de APIs entre componentes de aplicación. De

manera similar, el organismo IETF define un modelo de gestión basado en el uso del protocolo SNMP [44] como solución a la gestión de redes IP, si bien es cierto que SNMP ha sido usado fundamentalmente para tareas de monitorización.

Aunque la aportación más importante realizada por IETF para la gestión automática fue la definición de un nuevo paradigma denominado Gestión Basada en Políticas [104]. Este nuevo paradigma ha sido utilizado de manera extendida y actualmente es la base de las principales propuestas en el ámbito de la gestión, además de ser la base del trabajo de investigación que se presenta en esta tesis. Una visión más detalla de la Gestión basada en Políticas es presentada en la siguiente sección.

Por último destacar que en los últimos años han aparecido nuevas soluciones basadas en XML [66] que codifican la información de gestión en XML debido a las ventajas que ello conlleva: su gran aceptación con multitud de herramientas y la facilidad de integración con distintas aplicaciones de gestión. Uno de los pioneros en esta área es el DMTF que desarrolló la arquitectura WBEM [19] que toma como base el modelo de información CIM [18]. Este modelo de información que define todo un conjunto de conceptos relativos a la gestión de redes, usuarios y servicios también será una de las bases del trabajo de investigación de esta tesis y será presentado en el siguiente capítulo. Junto a la iniciativa WBEM existen otros trabajos basados en XML, los más destacados son: la propuesta de NMRG para usar XML en SNMP [59], la iniciativa del IETF con la definición del protocolo de configuración XMLCONF [47] y la propuesta de OASIS para la especificación de políticas de control de acceso mediante el lenguaje XACML [72]. Actualmente los mayores esfuerzos son acaparados por los Servicios Web que han emergido como la tecnología de gestión más prometedora y, en particular, en su modelo de seguridad [46] donde destaca la propuesta WS-Policy [5] para la gestión de políticas.

1.2. Sistemas de Gestión basados en Políticas

En los esfuerzos de estandarización del IETF es fundamental su aportación con la definición de un nuevo paradigma denominado Gestión basada en Políticas [104, 103], donde una Política se define como un conjunto de reglas o prácticas que regulan dinámicamente el comportamiento del sistema de manera autónoma frente a los elementos gestionados.

Así una de las principales metas de la Gestión basada en Políticas es alcanzar el control y la gestión de la red, sus aplicaciones y servicios en una capa de abstracción de alto nivel [93, 94, 88]. En este sentido el administrador

determina las reglas que describen las políticas usando un lenguaje específico para ello y es entonces la arquitectura de gestión la que proporciona soporte para transformar y distribuir las políticas a cada dispositivo y teniendo como último objetivo aplicar una configuración consistente en cada uno de ellos. Así las principales funciones de una arquitectura de gestión de políticas son:

- *Aplicación*: implementar la política deseada mediante los comandos de gestión disponibles.
- *Monitorización*: examen continuo activo y pasivo del sistema de información, sus servicios y aplicaciones para comprobar su estado y si las políticas son satisfechas o no.
- *Toma de decisiones*: comparar el estado actual del sistema de comunicación con el estado deseado descrito por las políticas y decidir como este estado puede ser alcanzado o mantenido.

El uso de políticas en la arquitectura de gestión permite varios niveles de abstracción. Habría, por ejemplo, políticas generales expresando un objetivo abstracto de negocio, y para otro fin, habría políticas que expresarían un conjunto de reglas más dependiente del servicio, la red o el dispositivo. La comunidad investigadora, la industria, y los organismos de estandarización han propuesto diferentes lenguajes de especificación de políticas para intentar cubrir estos niveles o completamente (por ejemplo, políticas de seguridad en la capa de negocio) o sólo en parte (por ejemplo, políticas de control de acceso), pero parece ser necesario cierto grado adicional de investigación para encontrar un estándar común de-facto para ser usado homogéneamente, o al menos un camino para combinar y validar varias aplicaciones y/o lenguajes de especificación para cubrir todos los aspectos de cualquier sistema de información.

Un avance más homogéneo ha sido identificado respecto al framework necesario para gestionar toda esta información de políticas. En este sentido, el esfuerzo de estandarización del IETF/DMTF es considerado como un buen punto de partida, aunque varios trabajos de investigación e implementaciones han propuesto vistas alternativas o un mayor detalle en los diferentes bloques que forman los componentes de la arquitectura de gestión de políticas.

La figura 1.2 muestra los cuatro principales elementos funcionales del framework de políticas del IETF. Estos son también utilizados como parte de la mayoría de las arquitecturas de gestión basadas en políticas, como es el caso de XACML [72].

- El *Policy Management Tool* (PMT) permite al administrador crear, editar y borrar políticas y monitorizar el estado del entorno gestionado.

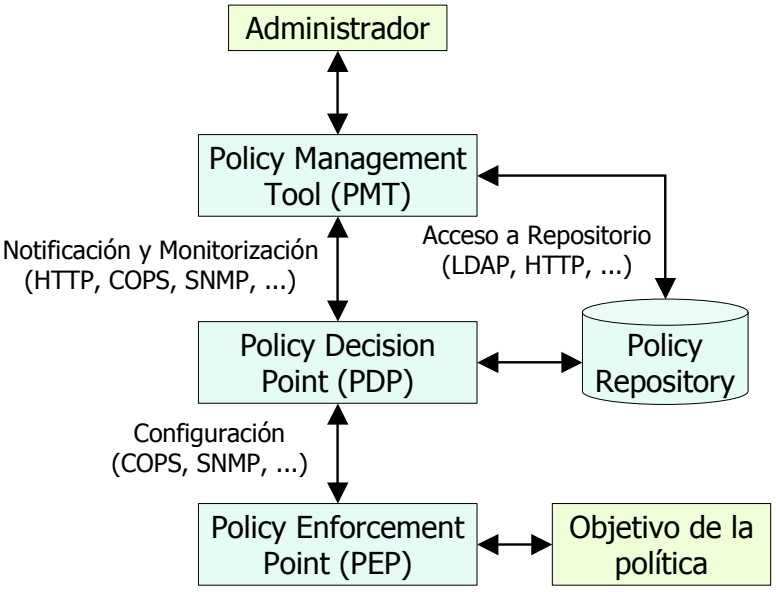

Figura 1.2: *Arquitectura IETF*

- El *Policy Repository* es un repositorio que es usado por el PMT para almacenar la políticas, y por los PDPs para recuperarlas. El IETF sugiere el uso de LDAP, pero otras soluciones son posibles tales como directorios o bases de datos.

- El *Policy Decision Point* (PDP) tiene una visión sobre todo el área de red bajo su control y es responsable de interpretar las políticas almacenadas en el repositorio, recuperar el conjunto de reglas necesarias por cada PEP, transformarlas en un formato que pueda ser entendido por el PEP y distribuirlas al PEP. Normalmente el PDP es responsable de un conjunto específico de políticas, por ejemplo políticas de control de acceso.

- El *Policy Enforcement Point* (PEP) es el componente ejecutándose en un nodo de red que puede aplicar diferentes políticas. Los PEPs pueden también informar al PDP cuando alguna situación desconocida se produce.

El framework del IETF establece COPS y COPS-PR [23] para transferir decisiones de políticas a los PEPs y para transferir solicitudes y reportes desde el PEP al PDP. El modelo queda abierto a otros mecanismos tales como HTTP/HTTPS, FTP o SNMP. Además, los datos en el Policy Repository cambiarán de vez en cuando y los PDPs serán informados de estos cambios cuando ocurran. Aunque el framework no especifica el medio de notificar

estos cambios, hay varias formas posibles para hacer esta notificación (por ej. mediante polling, notificaciones de cambio LDAP, SNMP traps, etc).

1.3. Gestión de Seguridad mediante Políticas

Como hemos visto anteriormente, la Gestión de la Seguridad era definida en el modelo TMN/OSI como una de las principales áreas funcionales de la gestión, a la vez que se situaba en cada una de la capas de gestión. De esta manera podemos hablar de una gestión de la seguridad a nivel de negocio, a nivel de servicio, a nivel de red y a nivel de elemento. De acuerdo con esta visión de la gestión, el paradigma de la Gestión basada en Políticas encaja perfectamente, donde una política de seguridad – que puede ser definida como un conjunto de reglas o prácticas que describen como una organización establece sus servicios de seguridad – queda determinada según su nivel de abstracción.

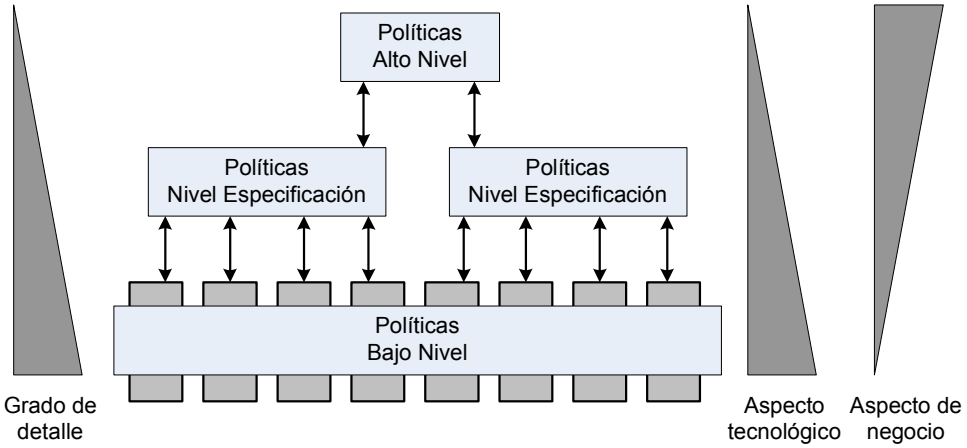

Figura 1.3: *Niveles de abstracción de una política*

Estos niveles de abstracción obligan al sistema a que deba ser capaz de refinar políticas de negocio (es decir, políticas de alto nivel) a políticas de servicios o/y red (es decir, políticas de nivel de especificación) y finalmente transformar éstas a configuraciones de los elementos (es decir, políticas de bajo nivel). Es claro, como muestra la figura 1.3, que cuanto menor nivel de abstracción tienen las políticas mayor es el grado de detalles tecnológicos expresados, y en cambio que cuanto mayor es el nivel de abstracción menores son los aspectos tecnológicos de la política. Por ejemplo, una política de seguridad de alto nivel puede quedar expresada en lenguaje natural como 'No está permitido el acceso a las oficinas fuera

de horario de trabajo'. Esta política debe ser refinada a políticas de nivel de especificación por el administrador de políticas de seguridad del sistema utilizando un lenguaje de especificación que, por ejemplo, podrían expresarse utilizando un lenguaje IF-THEN 'IF (hora>20:00) AND (hora<8:00) THEN forbidden(oficinas.abrirPuerta)'. La arquitectura del sistema de gestión de políticas se encarga de distribuir esta política a los elementos y transformarla en configuraciones finales.

Por otro lado, existe el riesgo potencial de que aparezcan conflictos entre políticas [63] que pueden darse por varios motivos, entre ellos, que se definan políticas que deriven en acciones contradictorias o en acciones no permitidas. Estos conflictos pueden ser causa de un administrador poco experimentado o por la existencia de varios administradores en el sistema. Por ejemplo, la política de alto nivel 'Permitir el acceso a las oficinas en caso de incendio', que puede ser expresada en IF-THEN como 'IF (detector.hayFuego()) THEN permitted(oficinas.abrirPuerta)', generaría un conflicto con la regla anterior ya que si se produjera un incendio fuera del horario de trabajo el sistema tendría que realizar simultáneamente dos acciones contradictorias, permitir y negar la apertura de puertas.

Estos conflictos podrían resolverse informalmente por los administradores, pero si deseamos un sistema de gestión automático, el reto está en conseguir que el sistema esté preparado para detectarlos y resolverlos apropiadamente. De esta manera resulta necesario incluir técnicas avanzadas que sean fáciles de integrar en la arquitectura y soportadas por los lenguajes de especificación de políticas.

1.4. Contribución de la Tesis

La gestión de la seguridad basada en políticas es un campo de investigación que aún necesita de grandes esfuerzos para conseguir una solución final al problema de la provisión automática de seguridad. La base de esta solución se encuentra en resolver los problemas que en la actualidad tienen los lenguajes de especificación de políticas. Fundamentalmente identificamos los siguientes problemas:

- *Falta de un modelo común.* No existe un modelo común que garantice la extensibilidad y la interoperabilidad entre diferentes sistemas de gestión. Cada lenguaje define su propio modelo de política y utiliza sus propios conceptos para definir los elementos de la política. Esto hace que la interoperabilidad entre sistemas sea muy compleja.

- *Facilidades para el análisis de conflictos.* La mayoría de los lenguajes

de políticas no están diseñados para facilitar la detección de conflictos y, muchos menos, pensados para realizar una resolución de conflictos.

- *Complejidad en el acceso a la información de la política.* La representación de las políticas no facilita las consultas sobre la propia información de las políticas y la mayoría de los lenguajes incluyen utilidades propietarias adicionales para este fin.

- *Facilidades para la distribución y aplicación de las políticas.* Los lenguajes no son diseñados para permitir su fácil integración en arquitecturas externas de gestión de políticas, y la mayoría ofrecen una solución que se integra dentro de una arquitectura concreta.

Esta tesis intenta avanzar en la solución de estos problemas anteriormente enumerados proponiendo un nuevo lenguaje de políticas para la gestión de la seguridad. Este nuevo lenguaje incluye características novedosas que lo diferencian del resto y a la vez mejora los lenguajes actuales como veremos con su validación en un framework de gestión concreto.

Este lenguaje de políticas incluye las siguientes características:

- Basado en el *modelo de información CIM*. Esto hace que el lenguaje esté basado en un modelo de información estandarizado con una semántica clara y bien definida que es independiente de cualquier implementación o representación específica. Además CIM permite definiciones de conceptos en diferentes grado de detalle consiguiendo con ello un lenguaje con múltiples niveles de abstracción.

- Una representación OWL de los conceptos e instancias del modelo CIM. Esta representación semántica, que denominamos *ontología CIM-OWL*, permite disponer de un lenguaje extensible ya que la ontología puede ser extendida en cualquier momento. Además el uso de OWL facilita la interoperabilidad con el resto de lenguajes semánticos de políticas.

- Una representación SWRL de las reglas de políticas. Esta representación lógica, que denominamos *lenguaje SWRL-CIM*, es propiamente el lenguaje de polítcas ya que define las políticas como reglas lógicas de primer orden (reglas Horn) donde los elementos de la regla son definidos mediante la ontología CIM-OWL.

- Mecanismos de razonamiento que facilitan las funciones de la arquitectura de gestión. El uso de representaciones semánticas permiten el uso de técnicas de razonamiento sobre las políticas que facilitan el acceso a la información de la política y el análisis de conflictos entre las políticas.

- Soporte para diferentes tipos de políticas de seguridad. La flexibilidad que proporciona el uso del modelo CIM permite que SWRL-CIM pueda definir diferentes tipos de políticas de seguridad. En nuestra propuesta presentamos a modo de ejemplo: políticas de autenticación basadas en credenciales; políticas de control de acceso que incluyen políticas de autorización y políticas de delegación; políticas de obligación basadas en eventos implícitos; políticas de red que incluyen políticas de filtrado y políticas de seguridad IP.

Otras contribuciones son:

- Se describen los conceptos fundamentales del proceso de definición de políticas y el papel que juega el lenguaje de políticas en este proceso para la definición de políticas formales.
- Se definen los requerimientos que un lenguaje de políticas debe cumplir para que facilite en la mayor medida posible su integración en una arquitectura de gestión de políticas.
- Se realiza un revisión crítica del estado del arte de los lenguajes de especificación de políticas de seguridad realizando una comparativa en base a los requerimientos establecidos.
- Se presenta una arquitectura de gestión donde nuestro lenguaje de políticas se implementa y se integra para la definición de políticas de seguridad en diferentes entornos de aplicación. Así se demuestra como nuestro lenguaje de políticas da una solución válida y adecuada.

1.5. Estructura de la Tesis

La memoria de esta tesis está estructurada del siguiente modo.

- En el Capítulo 2 se presenta el estado del arte de los lenguajes de políticas para la gestión de la seguridad. Se analizan y comparan desde lenguajes orientados a objetos a lenguajes semánticos basados en lógica y ontología. Mediante este análisis se intentan resaltar las principales características de unos y otros, al tiempo que se describen sus principales limitaciones y se constatan los problemas que se han resaltado anteriormente en este primer capítulo. Además se describe el proceso de definición de políticas, los elementos y los actores involucrados, los principales requerimientos de un lenguaje de políticas y las clases de políticas de seguridad que son consideradas cuando se analizan los lenguajes de políticas.

- El Capítulo 3 presenta nuestra propuesta para un lenguaje semántico de políticas que mantiene las ventajas de los lenguajes semánticos presentados en el capítulo anterior y a la vez aporta respuesta a sus principales desventajas y limitaciones. Para ello haremos una descripción detallada de nuestra propuesta: modelo de información CIM, ontología CIM-OWL y propiamente el lenguaje SWRL-CIM. Además veremos como SWRL-CIM facilita la aplicación de técnicas de razonamiento basado en reglas que nos permite definir un procedimiento para la detección de conflictos sobre las políticas definidas.

- El Capítulo 4 presenta como nuestra propuesta para la especificación de políticas de seguridad, es decir SWRL-CIM, se integra en un arquitectura de gestión de políticas y como esta arquitectura se utiliza en diferentes entornos de aplicación. Para ello, en primer lugar se presentan los detalles de implementación del lenguaje y después como éste se integra dentro de la arquitectura de gestión definida en el proyecto europeo POSITIF (del inglés, Policy-based Security Tools and Framework). Los entornos de aplicación seleccionados para mostrar la aplicabilidad del lenguaje son la gestión de las políticas de seguridad en servicios Grid y la gestión de las políticas de seguridad en firewalls distribuidos.

- El Capítulo 5 finaliza este trabajo de investigación poniendo de manifiesto los resultados más relevantes que se han conseguido. También es objetivo de este capítulo el definir algunas de las vías futuras que se vislumbran como de mayor interés a partir de la labor de investigación que se ha llevado a cabo.

1.6. Publicaciones relacionadas

El trabajo de investigación llevado a cabo a lo largo del desarrollo de esta tesis ha propiciado la publicación de diferentes trabajos en varios congresos, conferencias, capítulos de libro y revistas internacionales. A continuación se describen los trabajos más significativos:

1.6.1. Congresos y Conferencias

- Félix J. García, Gregorio Martínez, Andrés Muñoz, Juan A. Botía y Antonio F. Gómez-Skarmeta. **Distributed Provision and Management of Security Services in Globus Toolkit 4**. En

GADA Workshop, OnTheMove Workshops 2006, Montpellier, Francia, Noviembre 2006.

Este trabajo [37] muestra cómo el framework de gestión de POSITIF y el lenguaje SWRL-CIM son utilizados para gestión de políticas de seguridad en Globus Toolkit 4.

- Gregorio Martinez, Félix J. García, Andrés Muñoz, Juan A. Botía y Antonio F. Gómez-Skarmeta. **Enabling Conflict Detection using Ontology and Rule-Based Reasoning in the Specification of Security Policies**. En *HPOVUA'2006*, Niza, Francia, Mayo 2006.

  Este artículo [68] está centrado en la descripción del procedimiento definido para la detección de conflictos en políticas de seguridad definidas mediante el lenguaje SWRL-CIM.

- Félix J. García, Gregorio Martinez, Juan A. Botía y Antonio F. Gómez-Skarmeta. **On the Application of the Semantic Web Rule Language in the Definition of Policies for System Security Management**. En *AWeSOMe Workshop, OnTheMove Workshops 2005*, Larnaca, Chipre, Octubre 2005.

  Este trabajo [31] presenta el uso de la ontología CIM-OWL y el lenguaje SWRL-CIM para la definición de políticas de seguridad.

- Félix J. García, Jesús D. Jiménez, Gregorio Martínez y Antonio F. Gómez-Skarmeta. **Policy-Driven Routing Management Using CIM**. En *MMM-ACNS 2005*, San Petersburgo, Rusia, Septiembre 2005.

  Este trabajo [29] muestra como el modelo de información CIM puede ser utilizado para la definición de políticas de enrutamiento.

- Félix J. García, Gregorio Martinez y Antonio F. Gómez-Skarmeta. **An XML-Seamless Policy Based Management Framework**. En *MMM-ACNS 2005*, San Petersburgo, Rusia, Septiembre 2005.

  Este trabajo [35] presenta la definición de una arquitectura de gestión basada en políticas que hace uso de las tecnologías XML en todas sus áreas, desde la especificación a la aplicación de las políticas.

- Gregorio Martinez, Félix J. García, Juan A. Botía y Antonio F. Gómez-Skarmeta. **Extending the Common Information Model for Incorporating Semantics and Ontology-based Reasoning in the Specification of Security Policies**. En *HPOVUA'2005*, Oporto, Portugal, Julio 2005.

Este artículo [67] está centrado en la ontología CIM-OWL y en las técnicas de razonamiento que pueden aplicarse sobre ella.

- Félix J. García, Gregorio Martinez y Antonio F. Gómez-Skarmeta. **A Semantically-Rich Management System based on CIM for the OGSA Security Services**. En *KDMG Workshop, AWIC 2005*, Lodz, Polonia, Junio 2005.

  Este trabajo [34] muestra cómo la ontología CIM-OWL puede ser utilizada para la gestión de políticas de seguridad en OGSA (del inglés, Open Grid Services Architecture).

- Félix J. García, Gregorio Martinez, Juan A. Botía y Antonio F. Gómez-Skarmeta. **Representing Security Policies in Web Information Systems**. En *PM4W Workshop, WWW 2005*, Chiba, Japón, Mayo 2005.

  Este trabajo [32] presenta una comparativa entre KAoS, Rei y SWRL. Además muestra los beneficios de usar SWLR-CIM frente al resto de propuestas.

- Gabriel López, Félix J. García, Manuel Gil, Gregorio Martínez y Antonio F. Gómez-Skarmeta. **Deploying Secure Cryptographic Services in Multi-domain IPv6 Networks**. En *IEEE AINA 2005*, Taiwan, Marzo 2005.

  Este artículo [61] muestra los beneficios de utilizar una gestión basada en políticas en la gestión de redes IPv6 multidominio.

- Félix J. García, Gabriel López, Jesús D. Jiménez, Gregorio Martínez y Antonio F. Gómez-Skarmeta. **Deployment of a Policy-based Management System for the Dynamic Provision of IPsec-based VPNs in IPv6 Networks**. En *IEEE SAINT 2005*, Trento, Italia, Febrero 2005.

  Este trabajo [30] muestra los beneficios de utilizar una gestión basada en políticas en la gestión de redes privadas virtuales en redes IPv6.

- Félix J. García, Gregorio Martínez, Óscar Cánovas y Antonio F. Gómez-Skarmeta. **A Proposal of a CIM-based Policy Management Model for the OGSA Security Architecture**. En *GADA Workshop, OnTheMove Workshops 2004*, Larnaca, Chipre, Octubre 2004.

Este trabajo [36] muestra cómo la representación xCIM puede ser utilizada para la gestión de políticas de seguridad en OGSA (del inglés, Open Grid Services Architecture).

- Félix J. García, Óscar Cánovas, Gregorio Martínez y Antonio F. Gómez-Skarmeta. **Self-configuration of grid nodes using a policy-based management architecture**. En *APGAC Workshop, ICCS 2004*, Cracovia, Polonia, Junio 2004.

  Este trabajo [28] muestra cómo realizar una gestión basada en políticas en la gestión de Globus Toolkit 2.4.

1.6.2. Capítulos de Libro

- Gregorio Martínez, Félix J. García y Antonio F. Gómez-Skarmeta. *Web and Information Security*, capítulo **Policy-based Management of Web and Information Systems Security: an Emerging Technology**. E. Ferrari and B. Thuraisingham (eds), Idea Group Inc., 2005.

  Este capítulo [69] presenta los principales conceptos relativos a la gestión basada en políticas y cómo ésta puede utilizarse en la gestión de la seguridad en sistemas web.

- Félix J. García, Gregorio Martinez, Juan A. Botía y Antonio F. Gómez-Skarmeta. *Web Semantics and Ontology*, capítulo **Description of Policies Enriched by Semantics for Security Management**. Idea Group Inc., 2006.

  Este capítulo [33] presenta un análisis comparativo de los principales lenguajes usados para la especificación de políticas de seguridad.

1.6.3. Revistas internacionales

- Gregorio Martínez, Gabriel López, Félix J. García y Antonio F. Gómez-Skarmeta. **Dynamic and Secure Management of VPNs in IPv6 Multi-Domain Scenarios**. *Elsevier Computer Communications*, In Press., 2006.

  Este trabajo [70] muestra los beneficios de utilizar una gestión basada en políticas en la gestión de redes privadas virtuales en escenarios multidominio en redes IPv6.

- Gabriel López, Félix J. García, Manuel Gil, Gregorio Martínez y Antonio F. Gómez-Skarmeta. **Providing advanced authentication services in IPv6 multi-domain scenarios**. *International Journal Internet Protocol Technology*, 1(2), 2005.

  Este trabajo [62] muestra el uso de técnicas avanzadas para la gestión de los servicios de autenticación en redes IPv6 multidominio.

- Félix J. García, Antonio F. Gómez-Skameta, Gabriel López y Gregorio Martínez. **Secure VPNs over IPv6 Networks: An Evaluation and its Integration in a Policy Management Framework**. *Journal of Internet Technology*, 5(2), 2004.

  Este trabajo [38] muestra los beneficios de utilizar una gestión basada en políticas en la gestión de redes privadas virtuales en redes IPv6.

Capítulo 2

Lenguajes de Políticas para la Gestión de la Seguridad

Este capítulo pretende presentar el estado del arte en los lenguajes de políticas para la gestión de la seguridad. Se analizan y comparan desde lenguajes orientados a objetos a lenguajes semánticos basados en lógica y ontología. Mediante este análisis se intentan resaltar las principales características de unos y otros, al tiempo que se describen sus principales limitaciones. Comenzamos describiendo el proceso de definición de políticas, los elementos y los actores involucrados, los principales requerimientos de un lenguaje de políticas y las clases de políticas de seguridad que serán consideradas cuando analicemos los lenguajes de políticas.

2.1. Introducción

Hay dos perspectivas principales para definir una política [69] y cada una representa un nivel de abstracción diferente. Si una política expresa un objetivo, una dirección o una forma de proceder, esta clase de política será referenciada como una política de negocio (business policy), dado que su contexto es el nivel de negocio (business level). Por otra parte, si una política es definida como un conjunto específico de reglas para gestionar dispositivos, servicios y/o aplicaciones, y por tanto es usada en el nivel de especificación (o tecnológico), ésta se denomina acorde a su alcance, por ejemplo, política de calidad de servicio o QoS (del inglés, Quality of Service), política de routing, o política de control de acceso. Las políticas de negocio tienen un nivel más alto de abstracción y son normalmente transformadas en una o varias políticas de nivel tecnológico. En la figura 2.1 se muestra el flujo de definición de una política.

Además la traducción entre las políticas de negocio y las políticas tecnológicas podrían requerir información adicional, tal como la topología de la red y la metodología. La topología de la red consiste en la especificación del conjunto de dispositivos de red que se quieren gestionar, mientras que la metodología establece los principios que deberían ser usados a la hora de realizar dicho proceso de gestión (por ejemplo decide entre el uso de criptografía de clave pública o criptografía de clave simétrica). Así la política tecnológica es especificada usando un lenguaje de políticas que genera una política formal, esto es una especificación no ambigua de la política incorporando información de la topología de red y la metodología. Finalmente, la política formal es convertida a la configuración del dispositivo basándose en la información particular y específica del dispositivo, tal como sus capacidades y roles.

El administrador tiene la responsabilidad de la gestión de las políticas de un conjunto de redes, servicios y aplicaciones que definen su dominio administrativo. De esta manera, un dominio administrativo podría incluir varios routers, servidores, hosts, puntos de acceso y conmutadores bajo la misma autoridad y/o procedimiento de gestión. Además, las reglas son definidas en una aplicación del dominio y en un nivel funcional, a lo que se denomina el alcance de la política. Por ejemplo, podemos hablar de políticas de seguridad cuando la seguridad es el alcance de la política. De forma similar, podemos hablar de políticas de QoS, políticas de routing, etc.

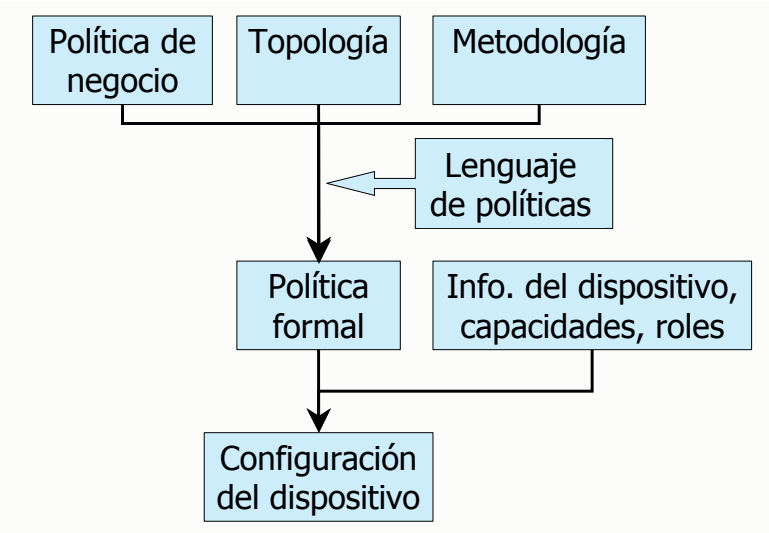

Figura 2.1: *Flujo de definición de una política*

El formato de política más usual define una política como un grupo de reglas de política. Cada regla es el enlace entre un conjunto de acciones y un

conjunto de condiciones. Si las condiciones son satisfechas, las acciones serán aplicadas. En otro caso, las acciones no serán realizadas.

Las políticas pueden ser disparadas (triggering) de dos formas, o estáticamente o dinámicamente. Las políticas estáticas aplican un conjunto concreto de acciones de una forma predeterminada acorde a un conjunto de parámetros definidos que determinan como la política es usada. Ejemplos de políticas estáticas son: 'el tráfico de red no es permitido fuera del horario de trabajo' o 'por razones de seguridad ciertas direcciones de red tienen denegado el acceso a los recursos de la red'.

Las políticas dinámicas son aplicadas sólo cuando son necesarias, y están basadas en condiciones cambiantes dentro del entorno donde son aplicadas tales como la congestión de la red, o la caída temporal de un servicio. Para soportar la dinamicidad de un dominio administrativo, las acciones pueden ser disparadas cuando un evento causa que una condición de la política se cumpla. Ejemplos de políticas dinámicas son: 'cuando la red está congestionada, el tráfico de video es denegado' o 'cuando el servicio de diagnostico es activado, no se aceptan peticiones de acceso al servicio web'.

Desde una perspectiva práctica, el administrador define y controla las políticas de un dominio administrativo desde lo que se da a conocer como consola de políticas. La gestión de las políticas será realizada a través de esta consola mediante un interfaz de usuario para permitir la definición de políticas y monitorizar el estado de los objetivos de las políticas. El objetivo de la política es una entidad, o un conjunto de entidades, cuyo comportamiento es definido usando políticas. Estas entidades se clasifican según su role; la principal razón para esto es que las políticas no son definidas para un objetivo individual, sino para un role. Esto permite gestionar y configurar dispositivos como un todo y no como una colección de dispositivos individuales, lo que también aporta escalabilidad a los sistemas de gestión. De esta manera, un role podría ser asignado a diferentes entidades, y cada entidad podría pertenecer a diferentes roles.

Una vez definida la política se debe realizar un análisis de los conflictos de políticas. Esta tarea se divide en dos tareas principales. Una primera tarea está centrada en la detección de conflictos en las políticas y una segunda tarea está dedicada a resolver (o guiar a los administradores en el proceso de resolución) estos conflictos. Se identifican en la bibliografía dos tipos de conflictos de políticas: conflictos semánticos y conflictos de modalidades [71]. El conflicto semántico es dependiente de la aplicación y se da cuando, en el contexto de una aplicación concreta, las acciones especificadas por dos políticas llevan al sistema a un estado inconsistente. En cambio el conflicto de modalidades es independiente de la aplicación y ocurre cuando dos políticas

tienen modalidad contraria, donde por modalidad entendemos el signo de la política donde lo positivo identifica a lo requerido y lo negativo a lo prohibido. Para llevar a cabo las tareas de análisis de conflictos es necesario disponer de una clasificación de los posibles conflictos de políticas que se desean analizar.

2.1.1. Requerimientos de un Lenguaje de Políticas

En el proceso de definición de políticas, el responsable principal de las políticas en una organización inicialmente proporciona la política de negocio usando un lenguaje informal del estilo de 'Los clientes requieren una protección fuerte en sus pagos on-line'. Entonces, la tarea del administrador de políticas es transformar las políticas de negocio en políticas de especificación usando una representación formal. Para hacer esto, el administrador usa un lenguaje de políticas que asegura que la representación de las políticas será no ambigua y verificable. Otros requerimientos importantes de un lenguaje de políticas son:

- Semántica clara y bien definida. La semántica del lenguaje de políticas puede ser considerada como bien definida si una política escrita en este lenguaje no es ambigua en el momento de su implementación.

- Flexible y extensible. Un lenguaje de políticas tiene que ser lo suficientemente flexible como para permitir que nueva información de políticas pueda ser expresada, y lo suficientemente extensible para permitir que nueva semántica sea añadida en futuras versiones del lenguaje.

- Independiente del dominio administrativo y, en particular, de los fabricantes/proveedores de los dispositivos y servicios. Un lenguaje de políticas puede ser considerado como independiente si una política escrita en este lenguaje es independiente de su implementación particular.

- Legibilidad. Un lenguaje de políticas debe ser legible y fácilmente interpretado por el administrador de políticas.

- Interoperabilidad con otros lenguajes. Existen normalmente varios lenguajes que puede ser usados en diferentes dominios para expresar políticas similares, y la interoperabilidad es un requerimiento para permitir que servicios y aplicaciones pueden comunicarse entre ellos acorde al comportamiento establecido en estas políticas.

Además de estos requerimientos, a un lenguaje de políticas también se le debe exigir que facilite de algún modo las funciones de la arquitectura de gestión. Así otros requerimientos exigibles al lenguaje serían:

- Facilita el análisis de conflictos. El lenguaje permite integrarse de manera natural en utilidades para el análisis de conflictos o dispone de elementos sintácticos y/o semánticos que facilitan este tipo de análisis.
- Permite el acceso a la información de la política. La propia representación de las políticas puede estar orientada a permitir consultas sobre la información de la política. Un ejemplo de consulta es preguntar por los privilegios de acceso asociados a un role determinado.
- Facilita la distribución y aplicación de las políticas. Es posible encapsular instancias de políticas en diferentes protocolos (por ejemplo, SNMP o COPS), y las instancias son fácilmente transformables a otras representaciones de menor nivel o a configuraciones particulares.

2.1.2. Representación Formal de Políticas

Los lenguajes de políticas deben proporcionar un representación formal de las políticas. Esta representación puede conseguirse de diferentes maneras, aunque las más importantes son la representación basada en lógica y la representación basada en ontología.

Representación basada en Lógica

La representación formal de políticas mediante notaciones de lógica formal es un importante avance dado que ofrece la oportunidad de tener una notación que dispone de una semántica libre de las ambigüedades de los lenguajes de alto nivel. Adicionalmente, un lenguaje formal para la especificación de políticas facilita las operaciones para la detección y resolución de conflictos mediante el razonamiento lógico, tal y como veremos a lo largo de esta tesis doctoral.

Hay gran número de propuestas para el desarrollo de representaciones de políticas basadas en lógica. Las más significativas son las que usan lógica deóntica estándar y subconjuntos especializados de lógica de primer orden, en particular, la lógica descriptiva.

La lógica deóntica proporciona operadores para la definición de permisos, prohibiciones y obligaciones. Si bien la lógica deóntica ha sido usada tradicionalmente como lógica del razonamiento jurídico, sus características también la hacen adecuada para la representación de políticas de seguridad.

De este modo, varias propuestas han tomado la lógica deóntica como base para la representación formal de políticas.

La lógica de primer orden es una clase avanzada de lógica, en el sentido que el elemento básico de representación es un predicado. Con un predicado podemos representar propiedades de una entidad o relaciones entre diferentes entidades. El uso de variables dentro de los predicados permite representar individuos. Además, podemos especificar el alcance de las variables usando cuantificadores como el cuantificador universal (para todos) y el cuantificador existencial (existe al menos uno).

Hablando sobre razonamiento, la lógica de predicados es consistente con las reglas de inferencia y es semi-decidible, es decir, si una expresión lógica puede ser deducida de un conjunto de expresiones lógicas, entonces existe una demostración de la primera expresión pero no siempre podemos encontrar la demostración de que una expresión lógica no puede ser deducida desde un conjunto de expresiones lógicas.

Una lógica más avanzada es la lógica descriptiva. Esta clase de lógica usa conceptos (o clases) en vez de predicados. Si la lógica de predicados está orientada a valores de verdad (TRUE o FALSE), la lógica descriptiva está orientada a conceptos y relaciones entre conceptos. Por ejemplo, podemos representar el hecho de que una usuario tiene un privilegio con un predicado binario

$\{(x|\exists y)(tienePrivilegio(x,y) \wedge Usuario(y))\}$

Pero en lógica descriptiva es ligeramente diferente ya que usamos propiedades para representarlo

$\exists tienePrivilegio.Usuario.$

Otra de las características importantes de la lógica descriptiva es la disponibilidad de un número de constructores para crear clases más complejas desde otras más básicas o complejas.

Representación basada en Ontología

Una definición de lo que es una ontología puede ser encontrado en [42]; según este autor 'una ontología es una especificación explícita de una conceptualización'. Desde el punto de vista de la representación del conocimiento en ordenadores, lo que es representado es lo que existe, en una forma declarativa. Y lo que es representado se realiza usando un modelo abstracto de una entidad concreta, en donde el modelo representa los conceptos más relevantes de esta entidad.

En la clasificación de las diferentes ontologías presentadas en [26] y usando como una dimensión para la clasificación la clase de fenómeno que se intenta representar, estamos interesados, en el contexto de esta tesis, en el dominio

de la ontología. Un dominio de ontología representa el conocimiento extraído de un dominio concreto de aplicación como el que estamos trabajando en esta tesis: la gestión de políticas de seguridad para sistemas distribuidos.

La tecnología de las ontologías encuentra su raíz en la lógica tradicional y, en particular, en la lógica descriptiva. La lógica formal proporciona dos importantes avances en la investigación ontológica:

- Lenguajes para la representación del conocimiento, y
- Modelos funcionales para la implementación de procesos de razonamiento.

En este sentido, con un apropiado mecanismo de razonamiento, podemos usar una especificación ontológica para inferir detalles interesantes sobre nuestro modelo del mundo, es decir, deducir y descubrir propiedades o relaciones entre conceptos que no definimos explícitamente.

La ontología puede ser, en principio, considerada como independiente del lenguaje usado para representarla. Como es argumentado en [42], en sistemas donde el conocimiento debe ser compartido entre entidades diferentes e independientes, es necesario un acuerdo común en tres diferentes niveles: formato del lenguaje para la representación del conocimiento, protocolo de comunicación para compartir el conocimiento y especificación del conocimiento a ser compartido. En este sentido, la tarea de definir una ontología está totalmente separada de la tarea de usar un lenguaje concreto para la representación del conocimiento. Así usamos lenguajes como XML, RDF y otros, que pueden ser vistos como no relacionados con las ontologías, para intercambiar conocimiento.

RDF [101] es un avance de interés dado que ofrece un camino directo y fácil para establecer hechos. Por otro lado, RDFS, el esquema de RDF, organiza los elementos de modelado en un camino consistente mediante clases y propiedades. Por ejemplo, se puede definir una jerarquía de clases en términos de composición de clases. También se puede añadir semántica añadiendo rangos, dominios y cardinalidad a las propiedades. OWL [97] es el lenguaje más actual hoy día para tratar con semántica, más específicamente en el contexto de la Web. Con OWL seremos capaces de hacer cosas como:

- Establecer que dos clases diferentes de políticas son disjuntas.
- Declarar que una nueva política es la inversa de otra política.
- Establecer nuevas políticas añadiendo restricciones a las propiedades de otras políticas.

OWL soporta razonamiento sobre inclusión (submission) en taxonomías de conceptos y permite la relación entre conceptos pudiendo expresar por ejemplo que X es parte de Y, o de forma más general, que existe una relación entre X e Y.

2.1.3. Tipos de Políticas de Seguridad

Tradicionalmente se identifican los siguientes servicios de seguridad: autenticación, autorización, confidencialidad, integridad y no-repudio. Una definición de alto nivel de estos servicios es: la autenticación es el proceso por el que se verifican las entidades cuando establecen una asociación; la autorización es usada para asegurar que la entidad tiene los privilegios apropiados para realizar una operación sobre otra entidad; la confidencialidad implica proteger la información intercambiada entre las entidades; la integridad garantiza que los mensajes intercambiados no han sido modificados; y el no-repudio asegura que la entidad emisora no pueda refutar que el mensaje ha sido enviada por ella.

Los servicios de seguridad pueden ser requeridos por aplicaciones, servicios o redes de un dominio administrativo. De esta manera un dominio administrativo podría requerir un servicio de autorización para el control de acceso a un dispositivo de la red o requerir un servicio de autenticación para la gestión de identidades y roles de los usuarios de la red. El administrador debe definir los requerimientos de estos servicios mediante la especificación de políticas de seguridad. La representación formal de estas políticas se consigue mediante un lenguaje de políticas de seguridad que puede ser relativo a uno o varios servicios de seguridad.

De esta forma, por ejemplo podemos hablar de lenguajes de políticas de autenticación donde los usuarios son normalmente identificados con su grupo o su role asociado, o de lenguajes de políticas de autorización donde las reglas de las políticas son definidas para especificar bajo qué condiciones un individuo puede acceder a ciertos recursos, información u otros elementos del sistema. Aunque podemos hablar también de lenguajes de políticas de confidencialidad, integridad o no-repudio, los lenguajes dedicados a la especificación de políticas de autorización y autenticación son los más comunes en cuanto a lenguajes de políticas de seguridad y la mayoría de los sistemas de gestión de políticas de seguridad los incorporan.

De cualquier modo un lenguaje de políticas de seguridad puede ser también utilizado para especificar políticas de obligación. Las políticas de obligación son políticas dinámicas para especificar qué acciones deben ser aplicadas cuando cierto estado de seguridad se da. Un ejemplo de política de obligación puede ser: 'si un usuario no consigue autenticarse frente a un

servicio tras tres intentos, el usuario es bloqueado y el administrador del servicio es informado mediante correo electrónico'.

2.1.4. Caso de Estudio

Una vez descritos los conceptos básicos de la gestión de políticas, además de las técnicas formales de representación y los requerimientos de los lenguajes de políticas, este análisis se completa con un caso de estudio, que será usado en el resto del capítulo para mostrar las ventajas, y también algunas de las cuestiones abiertas en el uso de los lenguajes de especificación de políticas existentes en la actualidad.

Nuestro caso de estudio se basa en los sistemas de comercio electrónico entre Empresa-Consumidor (B2C), los cuales son un ejemplo de sistema de información web (WIS) donde la provisión dinámica de seguridad es un aspecto fundamental a ser considerado. Un escenario de comercio electrónico seguro requiere la transmisión y recepción de información por Internet tal que:

- Es sólo accesible para el emisor y el receptor (privacidad/confidencialidad).
- No puede ser modificada durante la transmisión sin que el receptor detecte dicha modificación (integridad).
- El receptor tiene la garantía de que el emisor es quién dice ser (autenticación de origen).
- El emisor tiene la garantía de el emisor es quién dice ser (autenticación del destino).
- El emisor no puede denegar el haber realizado un envío (no-repudio).

Como decíamos, las políticas de seguridad pueden ser especificadas en diferentes niveles de abstracción. El proceso comienza con la definición de una política de seguridad de negocio. Éste puede ser el caso de la siguiente política de autorización que es expresada en lenguaje natural de la siguiente manera: 'Permitir el acceso al servicio de pago electrónico, si el usuario está en el grupo de clientes registrados para este servicio'.

Seguidamente, la política de seguridad es normalmente expresada por un administrador de políticas como un conjunto de reglas if-then como, por ejemplo:

IF ((<Requester>is member of Payment Customers) AND (<Server>is member of Payment Servers)) THEN (<Requester>granted access to <Server>)

Los lenguajes de políticas que son analizados tanto semánticos como no semánticos usarán esta política para permitir entender su descripción y analizar, de forma crítica, los aspectos positivos y negativos que presentan.

2.2. Lenguajes No Semánticos de Políticas de Seguridad

A continuación presentamos los lenguajes no semánticos de políticas de seguridad más destacados actualmente junto con sus frameworks de gestión. Estos son: Ponder que es un lenguaje orientado a objetos y declarativo; XACML y WS-Policy que son dos lenguajes orientados a Servicios Web basados en la representación XML de las políticas; y Policy CIM que define un modelo de información siguiendo un paradigma orientado a objetos.

2.2.1. Ponder

Ponder es el resultado del trabajo de investigación en el Imperial College London. Ponder [12] define un lenguaje orientado a objetos y declarativo que soporta la especificación de varios tipos de políticas de gestión para sistemas distribuidos y proporciona un framework para el desarrollo y la aplicación de políticas.

Ponder distingue entre políticas básicas y compuestas. Una política básica es considerada una regla que define el comportamiento para un conjunto de sujetos (subjects) y un conjunto de objetivos (targets). Ponder usa el término sujeto para referirse a los usuarios o componentes que pueden actuar sobre los objetivos, donde los objetivos son recursos o servicios con un conjunto de métodos en una interfaz bien definida. Las políticas básicas en Ponder son capaces de expresar autorizaciones, obligaciones, políticas de prohibición (definen acciones que los sujetos no deben realizar en objetivos) y políticas de delegación (definen que autorizaciones pueden ser delegadas). Las políticas compuestas permiten agrupar a las políticas básicas. Ejemplos de políticas compuestas son los roles y las políticas de relación. En Ponder, los roles son grupos de políticas que definen el comportamiento del mismo sujeto mientras las políticas de relación agrupan políticas que definen la relaciones entre roles.

Adicionalmente a las políticas básicas y compuestas, Ponder permite la definición de meta-políticas mediante OCL para especificar restricciones

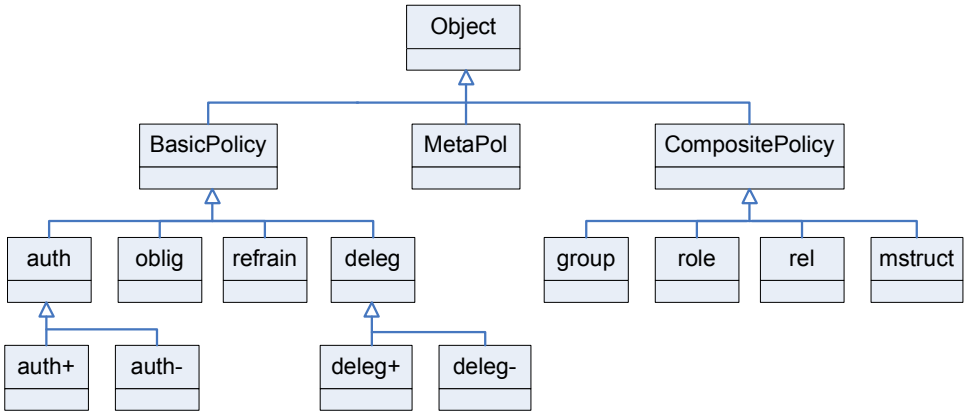

Figura 2.2: *Políticas en Ponder*

sobre el conjunto de políticas con respecto a los tipos de políticas permitidos o los elementos utilizados en las políticas.

El listado 2.1 presenta un ejemplo de política expresada en Ponder. Éste muestra como expresar con una política de autorización la política de seguridad definida en nuestro caso de estudio. Esta política de autorización define que el sujeto PayCostumer tiene permitido el acceso al objetivo PayServer.

Listado 2.1: *Ejemplo de política en Ponder*

```
1 inst auth+ PaymentAuthPolicy1 {
2                 subject s = people/PayCustomer ;
3                 target t = servers/PayServer ;
4                 action t.access (s);
5 }
```

Ponder proporciona varias herramientas gráficas para editar, actualizar, borrar y consular información sobre las políticas. Hay también herramientas para el análisis sintáctico y semántico de las especificaciones de las políticas y para transformar las especificaciones a XML o código Java para ser interpretadas en tiempo de ejecución. Además, los desarrolladores de Ponder han implementado una herramienta para detección de conflictos que detecta conflictos de modalidad que surgen cuando una acción es prohibida y requerida simultáneamente. Esta herramienta está basada en razonamiento abductivo y usa una representación Event Calculus de las políticas [2, 3].

Ponder especifica un framework completo desde la especificación de la política hasta la toma de decisiones para la aplicación de la política [13, 22]. En la aplicación de políticas distingue entre políticas de autorización y obligación. Los objetos de políticas de autorización son distribuidos junto con los objetivos de las políticas donde un agente denominado Controlador

de Acceso proporciona su interpretación y aplicación en tiempo de ejecución. Las objetos de políticas de obligación son distribuidos junto con los sujetos de las políticas donde un agente se encarga de la interpretación de los eventos. La figura 2.3 muestra gráficamente el modelo de aplicación de Ponder.

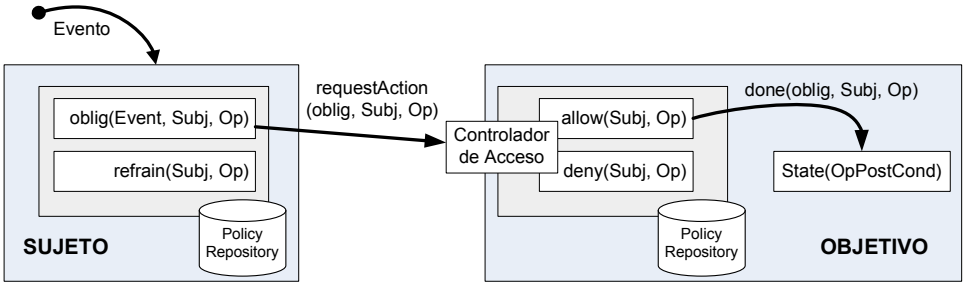

Figura 2.3: *Políticas en Ponder*

Aunque Ponder parece actualmente ser una de las soluciones más completas de los sistemas de gestión de políticas, Ponder carece de interoperabilidad con propuestas con similares objetivos, tal como CIM o XACML, aunque algún trabajo interesante en este sentido está siendo realizado por Imperial Collage y CISCO en el proyecto Polyander [83]. Por otro lado, Ponder proporciona un bajo nivel de abstracción ya que requiere que el administrador conozca los nombres específicos de las entidades gestionadas y sus interfaces.

2.2.2. XACML

El siguiente lenguaje de políticas de seguridad en nuestro análisis, eXtensible Access Control Markup Language (XACML) [74], es una especificación XML de OASIS para expresar políticas de control de acceso a información en Internet.

XACML describe un lenguaje de políticas de control de acceso y un lenguaje de solicitud/respuesta. El lenguaje de políticas proporciona un medio común para expresar políticas de control de acceso y el lenguaje de solicitud/respuesta permite expresar consultas sobre si un acceso particular debería ser permitido (solicitudes) y describe las respuestas a estas cuestiones (respuestas). XACML permite también definir políticas de obligación que especifican acciones que deben ser ejecutadas antes de permitir un acceso.

El modelo de clases del lenguaje de políticas usado en la versión 2.0. se muestra en la figura 2.4. Sus componentes principales son las reglas, las políticas y conjuntos de políticas:

- La regla (Rule), es la unidad elemental de una política y está compuesta de un sujeto (Target) relativo al conjunto de sujetos, recursos y acciones para los cuales la regla se debe aplicar; un efecto (Effect) que indica la consecuencia (permitir o denegar) de la evaluación positiva de la regla; y una condición (Condition) que refina la aplicabilidad de la regla.

- La política (Policy) es un conjunto de reglas con otros tres componentes: un objetivo (Target) con el mismo significado que se le daba en una regla; un identificador que indica como se evalúan las reglas; y obligaciones (Obligations) que indican operaciones específicas que deberían ser realizadas en conjunción con la aplicación de la decisión de autorización.

- El conjunto de políticas (Policy-set) está formado también por tres componentes: un objetivo (Target) con el mismo significado que anteriormente; un identificador que indica como se evalúan las políticas; y obligaciones (Obligations) que mantienen el significado anterior.

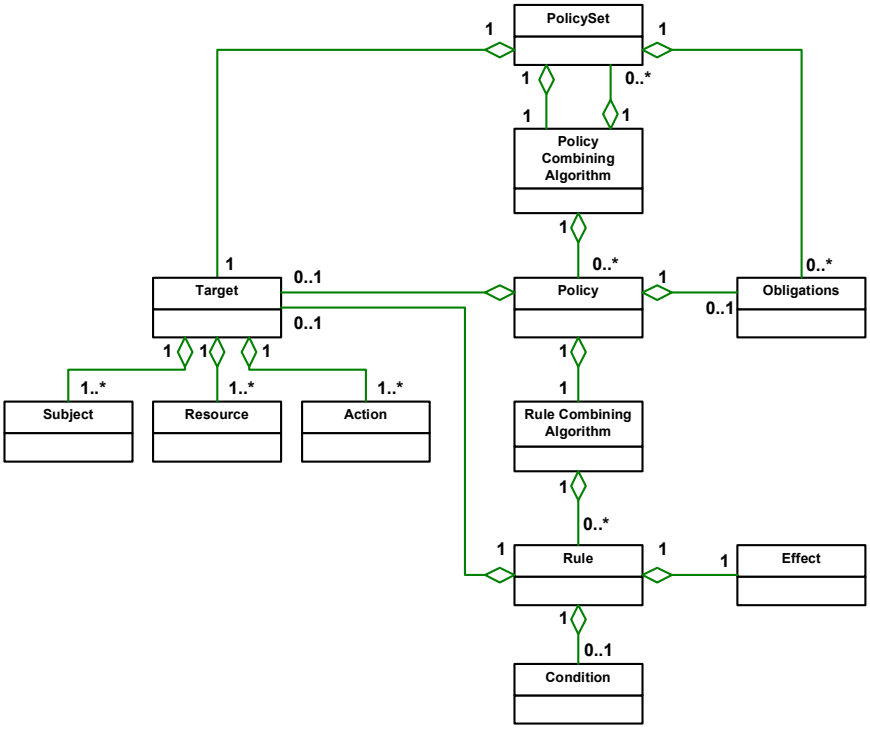

Figura 2.4: *Estructura de políticas en XACML*

Las políticas de control de acceso de XACML se especifican dentro del contexto de un documento XML particular. Sobre este contexto se soporta la

definición de roles o grupos en forma de expresiones XPath, o determinados por un conjunto de atributos que pueden ser del tipo sujeto (Subject), recurso (Resource) o acción (Action).

El listado 2.2 presenta un ejemplo de política expresado en XACML usando nuestro caso de estudio.

Listado 2.2: *Ejemplo de política en XACML*

```
<Policy PolicyId="PaymentAuthPolicy1">
  <Target>
    <Subjects><AnySubject/></Subjects>
    <Resources> <Resource>
      <ResourceMatch MatchId="function:anyURI-equal">
        <AttributeValue>
          http://payserver.ourcompany.com
        </AttributeValue>
        <ResourceAttributeDesignator/>
      </ResourceMatch>
    </Resource> </Resources>
    <Actions><AnyAction/></Actions>
  </Target>
  <Rule RuleId="ReadRule" Effect="Permit">
    <Target>
      <Subjects><AnySubject/></Subjects>
      <Resources><AnyResource/></Resources>
      <Actions> <Action>
        <ActionMatch MatchId="function:string-equal">
          <AttributeValue>access</AttributeValue>
          <ActionAttributeDesignator/>
        </ActionMatch>
      </Action> </Actions>
    </Target>
    <Condition FunctionId="function:string-equal">
      <Apply FunctionId="function:string-one-and-only">
        <SubjectAttributeDesignator AttributeId="group"/>
      </Apply>
      <AttributeValue>PayCustomer</AttributeValue>
    </Condition>
  </Rule>
</Policy>
```

XACML define un framework de gestión de políticas mostrado en la figura 2.5 que es un claro ejemplo de extensión del modelo IETF para un servicio particular de seguridad: control de acceso.

En un escenario XACML típico, un solicitante (por ejemplo, un usuario o una aplicación) quiere realizar alguna acción sobre un objetivo particular (por ejemplo, un recurso). El solicitante realiza la consulta a la entidad que gestiona el recurso, a ésta se denomina Policy Enforcement Point (PEP). Éste

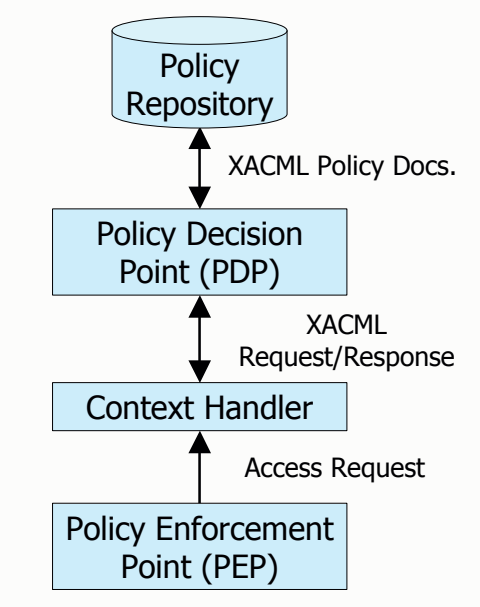

Figura 2.5: *Framework de gestión de políticas de XACML*

forma un mensaje de solicitud XACML basado en los atributos del solicitante, la acción, el objetivo, y alguna otra información relevante, y la envía al Policy Decisión Point (PDP). El PDP analiza la solicitud y determina si el acceso debería ser concedido o denegado acorde a las políticas XACML que son aplicables a esta solicitud. Entonces, un mensaje de respuesta XACML es devuelto al PEP, el cual entonces permitirá o denegará el acceso al solicitante.

En esta arquitectura, el PDP contesta las consultas de autorización basado en las políticas de control de acceso definidas por el administrador y el Context Handler es implementado como un módulo de autorización para el PEP. Así el Context Handler genera las solicitudes de autorización basado en las solicitudes de los recursos que el PEP recibe desde el nodo de red y proporciona la comunicación entre el PDP y el PEP.

En algunos entornos de aplicación se deben trasladar las solicitudes/respuestas que recibe el PEP del solicitante al contexto XACML. Por ejemplo, si consideramos SAML [75] que es el protocolo más extendido en servicios web para el intercambio de información de autenticación y autorización a través de web, una aplicación puede entonces proporcionar un mensaje SAML que incluya un conjunto de atributos relativos a una solicitud de acceso. Este mensaje tiene que ser convertido a formato XACML y análogamente la decisión XAML tiene entonces que ser convertida a formato SAML.

La principal ventaja de XACML es que usa XML. Esta tecnología es

un estándar adoptado muy extendido en los navegadores web que puede ser usado para mostrar y editar la especificación de la política. XACML también proporciona un control fino de las actividades (tales como leer, escribir, copiar, borrar) basado en varios criterios, incluyendo los siguientes: atributos del usuario que solicita el acceso (por ejemplo 'Sólo los administradores pueden ver este documento'), el protocolo sobre el cual se hace la solicitud (por ejemplo 'Estos datos pueden ser vistos solo si son accedidos mediante HTTPS'), y el mecanismo de autenticación (por ejemplo 'El solicitante debe haber sido autenticado usando un dispositivo digital'). Por otro lado, la especificación del framework XACML no describe como realizar la detección de conflictos ni, por tanto, como resolverlos.

2.2.3. WS-Policy

La siguiente solución en nuestro análisis, Web Services Policy Framework (WS-Policy) [5], es una especificación realizada por la colaboración entre Microsoft, IBM, BEA y SAP para conseguir un modelo y una sintaxis genéricas para describir y distribuir políticas en Servicios Web.

WS-Policy define un modelo de política que puede ser usado y extendido por otras especificaciones de Servicios Web. WS-Policy define una política como una colección de uno o más policy assertions donde un assertion representa una preferencia, un requerimiento, una capacidad, u otra propiedad individual. WS-Policy proporciona un gramática flexible y extensible para expresar políticas en XML. La representación XML de una política es enlazada a un servicio web tal y como se muestra en la figura 2.6. Esta asociación puede conseguirse mediante WS-PolicyAttachment que define como asociar políticas a servicios web.

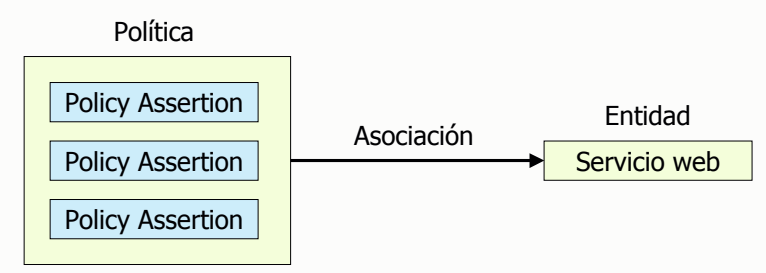

Figura 2.6: *Estructura de política en WS-Policy*

Hay dos especificaciones adicionales que definen conjuntos estándar de assertions que pueden ser usados para especificar una política. Uno es el lenguaje WS-PolicyAssertions que define un conjunto de assertions generales

y otro es el lenguaje WS-SecurityPolicy que define un conjunto de assertions relativos a WS-Security [46] que es la especificación del modelo de seguridad en Servicios Web. La figura 2.7 muestra el conjunto de especificaciones de Web Services para la seguridad. En la figura se indican que especificaciones están en desarrollo, propuestas o se esperan en un futuro próximo.

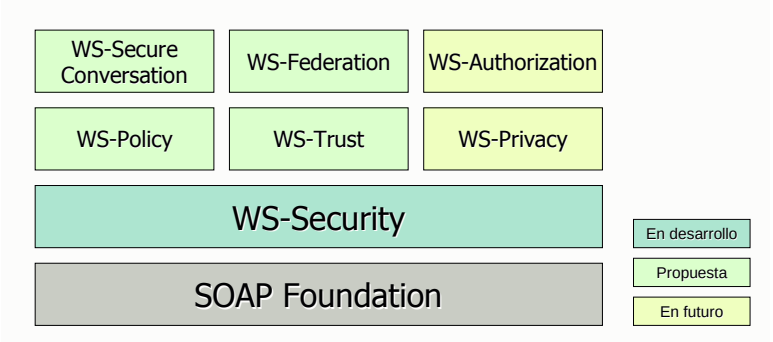

Figura 2.7: *Especificaciones de Servicios Web para la Seguridad*

En la capa inferior, varias tecnologías tales como SOAP, WSDL, firmas digitales XML, cifrado XML y SSL/TLS son el núcleo del modelo Web Services. Construido encima de éste, el modelo de seguridad (WS-Security) proporciona la base para las otras especificaciones de seguridad, las cuales incluyen un modelo para políticas (WS-Policy), un modelo de confianza (WS-Trust), un modelo para conversaciones seguras (WS-SecureConversation), un modelo de confianza federada (WS-Federation), un modelo de privacidad (WS-Privacy) y un modelo de autorización (WS-Authorization).

WS-Authorization, al igual que WS-Privacy, aún no ha sido propuesto y por esta razón no es posible expresar nuestro caso de estudio. De todas formas el listado 2.3 muestra un ejemplo de una política de autenticación que representa dos assertions que indican qué dos tipos de autenticación son soportados y que de los dos tipos, la autenticación mediante Kerberos tiene preferencia sobra la autenticación mediante certificados X.509.

Listado 2.3: *Ejemplo de política en WS-Policy*

```
1 <wsp:Policy xmlns:wsse="..." xmlns:wsp="...">
2   <wsp:ExactlyOne>
3       <wsse:SecurityToken
4           wsp:Usage="wsp:Required" wsp:Preference="100">
5           <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
6       </wsse:SecurityToken>
7       <wsse:SecurityToken
8           wsp:Usage="wsp:Required" wsp:Preference="1">
```

```
9            <wsse:TokenType>wsse:X509v3</wsse:TokenType>
10         </wsse:SecurityToken>
11      </wsp:ExactlyOne>
12   </wsp:Policy>
```

En relación a las políticas, conceptualmente cada política es convertida a una forma normal disyuntiva, es decir las políticas se convierten en secuencias de conjuntos de assertions alternativos. Cuando dos o más políticas son aplicables se deben intersectar generando una política nueva. Si la intersección es vacía, las dos políticas son incompatibles. De esta manera, un conjunto de assertions de una política es compatible con un conjunto de assertions de otra política si cada instancia de un tipo de assertion en una política es compatible con cada instancia de este tipo de assertion en la otra política. Si una instancia de un assertion dado ocurre sólo en un conjunto, entonces el comportamiento asociado con este tipo de assertion es prohibido en la intersección de estas políticas; así por ejemplo si una política requiere cifrado, y la otra no dice nada sobre el cifrado, entonces se prohibe el cifrado.

Para facilitar la intersección entre políticas, la especificación que define un assertion define todas las variaciones posibles de este elemento, qué genera la intersección de dos instancias de este assertion, qué combinaciones no son permitidas y cómo las distintas formas del assertion se relacionan con instancias del vocabulario específico del dominio. Cualquier procesador de políticas que deba verificar un mensaje o comparar instancias debe incorporar un módulo que implemente la semántica especificada para los assertions.

WS-Policy define un lenguaje de políticas fácilmente extensible donde cada nuevo dominio requiere su propio conjunto de assertions. Aunque, por otro lado, WS-Policy no describe como debería realizarse la distribución, aplicación y monitorización de políticas, ni como llevar a cabo el análisis de conflictos entre políticas

2.2.4. Policy CIM

El modelo CIM (Common Information Model) [18] es un estándar del DMTF que aplica las técnicas de conceptualización y estructuración básicas del paradigma orientado a objetos para proporcionar una definición común de la información relativa a la gestión de sistemas, redes, aplicaciones y servicios.

CIM define un conjunto de clases y asociaciones que proporcionan una estructuración conceptual dentro de la cual es posible organizar la información de gestión de un dominio administrativo. De esta manera, CIM permite ver al dominio administrativo como una colección de sistemas interrelacionados, donde cada uno está compuesto de un cierto número de elementos. CIM en sí mismo está estructurado en diferentes capas:

- Core Model. Un modelo de información que refleja los conceptos que son aplicables a todas las áreas de gestión.
- Common Model. Un modelo de información que refleja los conceptos que son comunes a un área particular de gestión, pero de manera independiente a una tecnología o implementación particular. Ejemplos de estas áreas comunes son sistemas, aplicaciones, redes y dispositivos.
- Esquemas de extensión. Representan extensiones específicas del Common Model. Estos esquemas son específicos de los entornos de aplicación, tal como es sistemas operativos (por ejemplo, UNIX o Microsoft Windows).

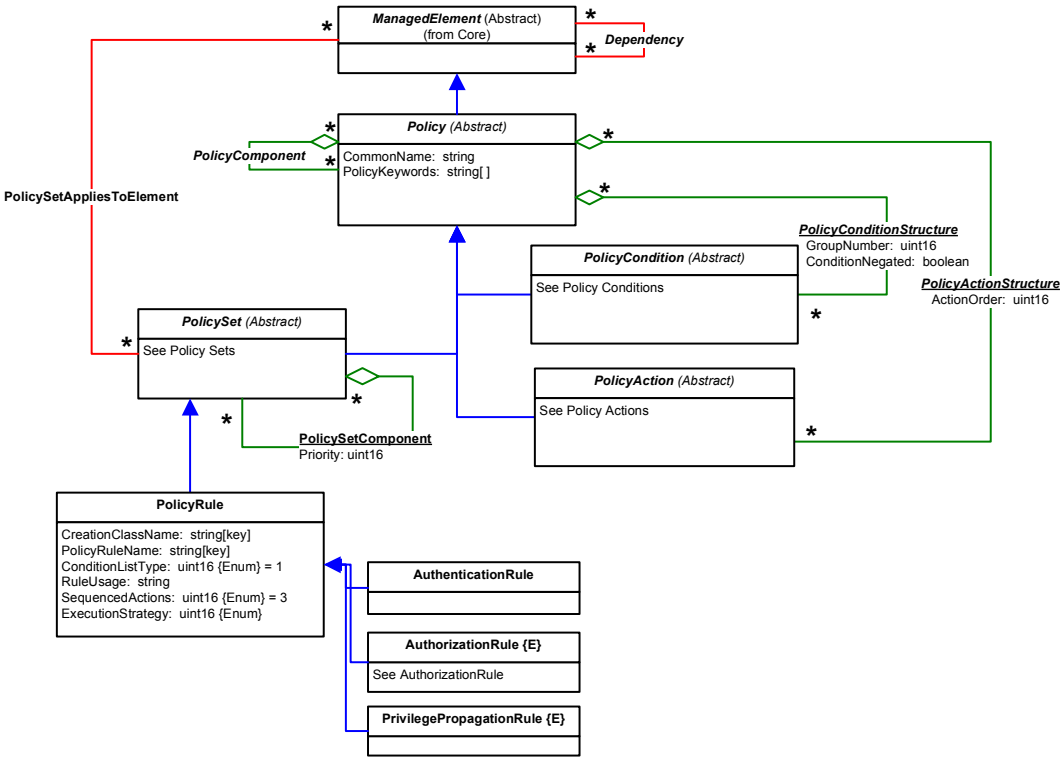

Figura 2.8: *Diagrama de clases de Policy CIM*

El Core Model y el Common Model forman el denominado esquema CIM. Este esquema ha llegado a ser tan extenso que ha sido segmentado para facilitar su uso. Así en la última versión v2.12 encontramos catorce subdivisiones donde una de ellas está dedicada a la información de gestión relativa a las políticas. Esta subdivisión, Policy CIM, proporciona una jerarquía de clases y asociaciones para especificación de políticas como reglas

compuestas de condiciones y acciones. Ésta incluye, entre otros elementos, reglas de políticas, grupos de políticas, acciones y condiciones de políticas, tanto en modo genérico como de forma específica. La figura 2.8 muestra un diagrama UML con algunas de las principales clases y asociaciones que forman la especificación Policy CIM.

Dado que Policy CIM tiene como intención ser genérica, las clases PolicyAction, PolicyCondition y PolicyRule son definidas como abstractas y sirven como base para la definición de políticas de obligación donde los eventos son implícitos; esto quiere decir que las políticas son disparadas cuando ciertos atributos de los elementos del dominio cambian. No obstante, algunas reglas esenciales son definidas, tales como AuthenticationRule para políticas de autenticación, AuthorizationRule para políticas de autorización y PrivilegePropagationRule para políticas de delegación de privilegios.

El listado 2.4 presenta un ejemplo de política expresada en CIM utilizando el formato básico definido por el DMTF, Management Object Format (MOF). Este ejemplo muestra como expresar nuestro caso de estudio con una regla de autorización.

Listado 2.4: *Ejemplo de política en CIM Policy*

```
1  Instance of CIM_AuthorizationRule {
2     CreationClassName = "CIM_AuthorizationRule";
3     PolicyRuleName = "PaymentAuthPolicy1";
4  };
5  Instance of CIM_Role {
6     CreationClassName = "CIM_Role";
7     Name = "PayCustomer";
8  };
9  Instance of CIM_AuthorizedPrivilege {
10    InstanceID = "AccessPrivilege";
11    Activities = {"Access"};
12 };
13 Instance of CIM_ProtocolService {
14    SystemCreationClassName = "CIM_ComputerSystem";
15    SystemName = "Server";
16    CreationClassName = "CIM_ProtocolService";
17    Name = "PayServer";
18 };
19 Instance of CIM_AuthorizationRuleAppliesToRole {
20    PolicySet="CIM_AuthorizationRule.
21 ␣␣␣␣␣␣␣␣CreationClassName=\"CIM_AuthorizationRule\",
22 ␣␣␣␣␣␣␣␣PolicyRuleName=\"PaymentAuthPolicy1\"";
23    ManagedElement="CIM_Role.
24 ␣␣␣␣␣␣␣␣CreationClassName=\"CIM_Role\",Name="PayCustomer\"";
25 };
26 Instance of CIM_AuthorizationRuleAppliesToPrivilege {
27    PolicySet="CIM_AuthorizationRule.
```

```
28 ␣␣␣␣␣␣␣␣CreationClassName=\"CIM_AuthorizationRule\",
29 ␣␣␣␣␣␣␣␣␣␣␣PolicyRuleName=\"PaymentAuthPolicy1\"";
30 ␣␣␣␣ManagedElement="CIM_AuthorizedPrivilege.
31 ␣␣␣␣␣␣␣␣InstanceID=\"AccessPrivilege\"";
32 ␣};
33 ␣Instance of CIM_AuthorizationRuleAppliesToTarget {
34 ␣␣␣␣PolicySet="CIM_AuthorizationRule.
35 ␣␣␣␣␣␣␣␣CreationClassName=\"CIM_AuthorizationRule\",
36 ␣␣␣␣␣␣␣␣PolicyRuleName=\"PaymentAuthPolicy1\"";
37 ␣␣␣␣ManagedElement="CIM_ProtocolService.
38 ␣␣␣␣␣␣␣␣␣SystemCreationClassName=\"CIM_ComputerSystem\",
39 ␣␣␣␣␣␣␣␣␣SystemName=\"Server\",
40 ␣␣␣␣␣␣␣␣␣CreationClassName=\"CIM_ProtocolService\",
41 ␣␣␣␣␣␣␣␣␣Name="PayServer\"";
42 ␣};
```

Además de MOF, CIM puede ser representado con otras especificaciones estructuradas que faciliten su uso en entornos de aplicación concretos. Por ejemplo, DMTF dentro de la iniciativa WBEM [19] define la representación de clases e instancias CIM en XML (XML-CIM). Esta representación se basa en una gramática XML escrita como un DTD (Document Type Definition), donde las clases e instancias CIM son documentos XML válidos para este DTD. En otras palabras el DTD es usado para describir de manera genérica la noción de un clase o instancia CIM y los nombres de los elementos CIM son mapeados a valores de elementos o atributos XML. Otra iniciativa dentro del DMTF es WS-CIM [20] que describe como representar y acceder al contenido del modelo CIM usando Servicios Web. Esta representación describe un mapeo completo del modelo CIM incluyendo clases, instancias y operaciones a Servicios Web usando la especificación WSDM-MUWS (Web Services Distributed Management: Management Using Web Services) [45].

En definitiva, CIM es un modelo de información práctico para la especificación de políticas, pero no define como deben realizarse las funciones típicas de una arquitectura de gestión. De esta manera CIM no especifica como puede conseguirse la aplicación y distribución de políticas, ni la detección y resolución de conflictos entre políticas. A pesar de iniciativas como WBEM o WS-CIM que han definido arquitecturas de gestión de recursos que permiten la distribución de información de gestión basadas en CIM, éstas no consiguen una solución que cubra las funciones típicas de la gestión de políticas identificadas al principio de este capítulo.

2.3. Lenguajes Semánticos de Políticas de Seguridad

A continuación presentamos los lenguajes semánticos de políticas de seguridad más destacados actualmente junto con sus frameworks de gestión. Estos son: KAoS que define un lenguaje basado en ontología con representación OWL; Rei que describe un lenguaje basado en lógica deóntica y ontología con representación OWL; RuleML que es un lenguaje para la representación XML de reglas lógicas; y SWRL que dentro de la iniciativa RuleML proporciona una sintaxis abstracta para escribir reglas lógicas que incluyen ontologías OWL.

2.3.1. KAoS

KAoS [56, 92] es una colección de servicios y herramientas que permiten la especificación, gestión, análisis y aplicación de políticas dentro de dominios que describen organizaciones de seres humanos, agentes, u otros actores computacionales. Aunque inicialmente fue orientado a los requerimientos complejos y dinámicos de las aplicaciones de agentes, los servicios KAoS también han sido extendidos para trabajar en computación grid y servicios web.

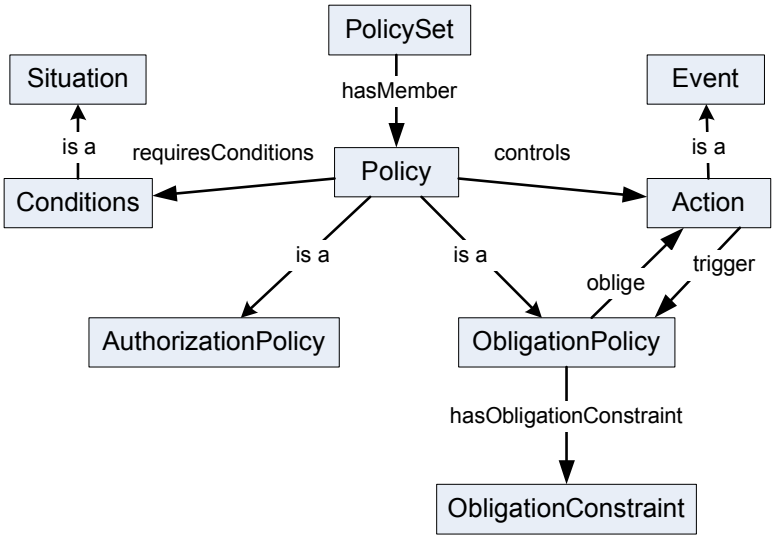

Figura 2.9: *Ontología de políticas de KAoS*

KAoS propone una solución basada en ontología para la especificación de políticas y las políticas son representadas usando la notación OWL. La

definición de la ontología KAoS hace una distinción entre las políticas de autorización y las políticas de obligación y cada tipo de política puede ser especializado con un operador de modalidad positiva o negativa indicando respectivamente si es permitida o prohibida. La aplicabilidad de la política es definida mediante un conjunto de condiciones, y su definición puede contener componentes especificando los antecedentes requeridos, el estado y las acciones asumidas actualmente. En el caso de una política de obligación la acción puede ser asociada con diferentes limitaciones (por ejemplo para no permitir la ejecución de acciones simultáneamente) restringiendo las posibilidades de su aplicación.

La versión actual de la ontología KAoS define las ontologías básicas para acciones, condiciones, actores, varias entidades relacionadas con las acciones, y políticas, tal como muestra la figura 2.9. La ontología puede ser extendida para soportar conceptos específicos de aplicación. Así, se espera que para una aplicación dada, la ontología sea extendida con clases adicionales, individuos y reglas.

El listado 2.5 muestra un ejemplo del tipo de política que los administradores pueden especificar usando KAoS. Éste es relativo al caso de estudio descrito anteriormente. Como se puede ver se define en primer lugar la acción de acceso PaymentAuthAction y posteriormente la política de autorización PaymentAuthPolicy1 asociada a esta acción.

Listado 2.5: *Ejemplo de política en KAoS*

```
<owl:Class rdf:ID="PaymentAuthAction">
<owl:intersectionOf rdf:parseType="owl:collection">
   <owl:Class rdf:about="&action;AccessAction"/>
   <owl:Restriction>
    <owl:onProperty rdf:resource="&action;#performedBy"/>
    <owl:toClass rdf:resource="&domains;MembersOfPayCustomer"/>
   </owl:Restriction>
   <owl:Restriction>
    <owl:onProperty rdf:resource="&action;#performedOn"/>
    <owl:toClass rdf:resource="&domains;MembersOfPayServer"/>
   </owl:Restriction>
</owl:intersectionOf>
</owl:Class>
<policy:PosAuthorizationPolicy rdf:ID="PaymentAuthPolicy1">
   <policy:controls rdf:ID="PaymentAuthAction"/>
   <policy:hasSiteOfEnforcement rdf:resource="#TargetSite"/>
   <policy:hasPriority>1</policy:hasPriority>
</policy:PosAuthorizationPolicy>
```

KAoS proporciona un framework, presentado en la figura 2.10, que incluye tres elementos básicos; una herramienta, denominada KAoS Policy Administration Tool (KPAT), que sirve de interfaz al administrador o

usuario; el servico KAoS Directory encargado de la distribución de políticas; y el Guard que actúan como Policy Decision Point (PDP). La funcionalidad que incluye el framework es la siguiente:

- Configuración del Sistema de Políticas. Durante el inicio del sistema, KAoS carga su ontología que define los conceptos usados para describir un dominio genérico y las políticas. Una configuración puede ser guardada o cargada posteriormente con políticas y definiciones.

- Consultar la Ontología. KPAT permite navegar y consultar por las ontologías cargadas para examinar su contenido (clases, propiedades, instancias y espacios de nombres importados). Esto permite añadir dinámicamente nuevas ontologías en tiempo de ejecución extendiendo conceptos de la ontología genérica, con nociones específicas del dominio gestionado.

- Creación de los Actores de las Políticas. Las políticas pueden ser definidas para un sujeto de diferentes maneras: mediante el uso de roles o con instancias concretas de actores. Estos conceptos también deben ser incluidos a través del KPAT.

- Creación de Políticas. Las políticas deben ser creadas usando KPAT. Esta herramienta guía al usuario mediante un proceso de creación. La codificación OWL de la política es generada utilizando el entorno de desarrollo proporcionado por Jena.

- Distribución de Políticas. Después de que una política es creada, debe ser distribuida mediante el servicio KAoS Directory a los Guard, que actúan como Policy Decision Points (PDPs). La política antes de ser enviada al Guard que corresponda es transformada de formato OWL a tablas hash más eficientes para la distribución de información.

- Toma de Decisiones de Políticas. La toma de decisiones (es decir, determinar qué políticas aplicar en una situación dada) no se limita a determinar si una acción es autorizada o prohibida o si alguna obligación es disparada, sino también permite una exploración de las opciones relativas a la política. Por ejemplo, es posible encontrar los valores permitidos que puede tomar una propiedad de una acción. KAoS proporciona una API que permite a una entidad describir su acción usando conceptos ontológicos y suministrarla al Guard para tomar una decisión de política sobre la acción.

- Análisis de Políticas. KAoS incluye soporte para detectar y resolver conflictos de modalidad que ocurren si dos políticas con modalidades opuestas especifican las mismas acciones y son aplicadas simultáneamente. Para conseguir esto, el razonamiento basado en Lógica Descriptiva es usado para encontrar relaciones entre las condiciones y las acciones de diferentes políticas de cara a determinar si están en conflicto. Los conflictos detectados son resueltos mediante la asignación de prioridades (como se puede ver en el ejemplo anterior con la etiqueta <policy:hasPriority>) y la derivación de un nuevo conjunto de políticas sin conflictos (este proceso se denomina armonización de políticas). KAoS integra el motor de razonamiento Java Theorem Prover (JTP)[91] para las tareas de razonamiento.

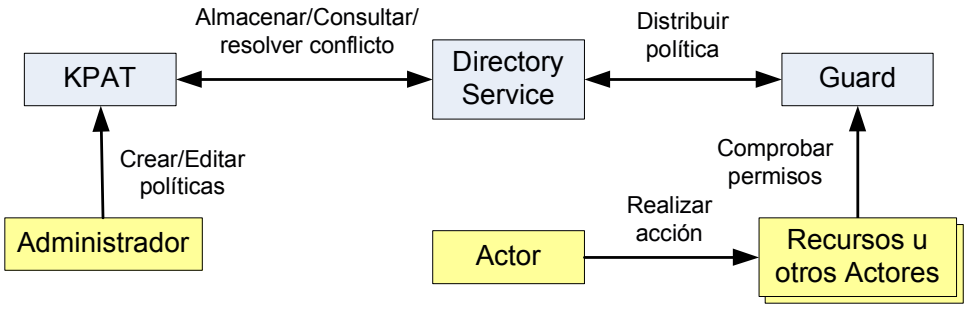

Figura 2.10: *Framework de KAoS*

Cada actor en el sistema es asociado con un Guard. Cuando una acción es solicitada, el Guard es automáticamente consultado para comprobar si la acción es permitida o rechazada. Así, la aplicación de políticas requiere la capacidad de monitorizar e interceptar acciones, y permitirlas o rechazarlas basadas en el conjunto de políticas dado.

El uso de OWL como una representación de políticas posibilita la extensibilidad y adaptabilidad del sistema en tiempo de ejecución, como también la habilidad de analizar políticas relativas a entidades descritas en diferentes niveles de abstracción. La representación facilita el razonamiento que permite el descubrimiento de políticas, la detección de conflictos de modalidad y la armonización de políticas.

2.3.2. Rei

Rei [57, 58] es un framework de políticas que permite la especificación y análisis de políticas en diversos dominios de gestión, aunque su principal dominio de desarrollo ha sido para soportar gestión basada en políticas en aplicaciones de sistemas pervasivos.

Su lenguaje de políticas basado en lógica deóntica permite a los usuarios expresar y representar los conceptos de derechos, prohibiciones, obligaciones y abstenciones que conceptualmente corresponden a las autorizaciones positivas y negativas y a las obligaciones positivas y negativas descritas en KAoS y Ponder. El lenguaje Rei también incluye los conceptos de actos de discurso (speech acts) y meta-políticas. Los actos de discurso son un mecanismo para permitir a los objetos del sistema transferir dinámicamente los derechos y obligaciones a otras entidades. Las meta-políticas son reglas sobre otras políticas y son usadas para resolver conflictos que son detectados por el motor de análisis de políticas Rei en tiempo de ejecución.

La especificación de políticas en Rei propone una solución basada en ontología para representar cada tipo de política y además las reglas de política son especificadas usando la representación OWL, tal y como hace KAoS. La figura 2.11 muestra el núcleo de esta ontología.

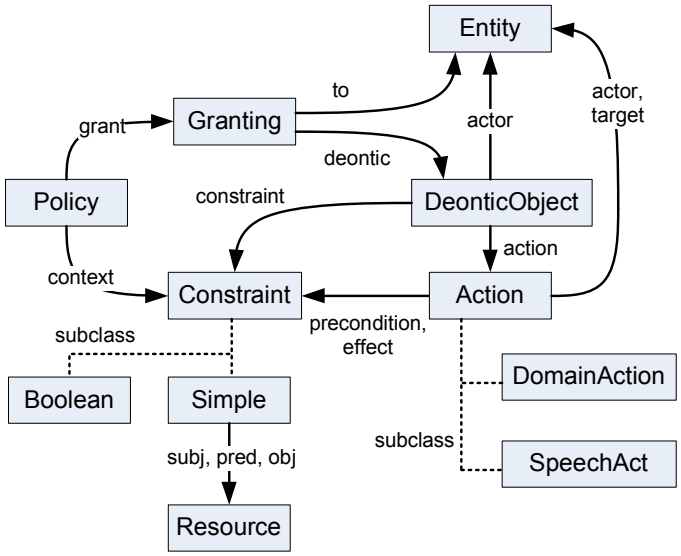

Figura 2.11: *Ontología de políticas de Rei*

En la ontología Rei una política incluye una lista de concesiones (grantings) y un contexto que define el dominio de aplicabilidad de la política. Una concesión asocia un conjunto de restricciones con un objeto deóntico para formar una regla de política. Esto permite reutilizar los objetos deónticos en diferentes políticas con diferentes restricciones y actores. Un objeto deóntico representa derechos, prohibiciones, obligaciones y abstenciones sobre entidades en el dominio de la política. Éste incluye constructores para describir la acción (o conjunto de acciones), el potencial actor (o conjunto de actores) de la acción y bajo qué condiciones es el objeto deóntico aplicable.

El listado 2.6 muestra un ejemplo, relativo al caso de estudio descrito anteriormente, del tipo de política que los administradores pueden especificar usando Rei en OWL.

Listado 2.6: *Ejemplo de política en Rei*

```
<constraint:SimpleConstraint rdf:ID="IsPayCustomer"
    constraint:subject="#RequesterVar"
    constraint:predicate="&example;memberOf"
    constraint:object="&example;payCustomer"/>
<constraint:SimpleConstraint rdf:ID="IsPayServer"
    constraint:subject="#PayServerVar"
    constraint:predicate="&example;memberOf"
    constraint:object="&example;payServer"/>
<constraint:And rdf:ID="ArePayCustomerAndPayServer"
    constraint:first="#IsPayCustomer"
    constraint:second="#IsPayServer"/>
<deontic:Permission rdf:ID="PayServerPermission">
    <deontic:actor rdf:resource="#RequesterVar"/>
    <deontic:action rdf:resource="&example;access"/>
    <deontic:constraint
            rdf:resource="#ArePayCustomerAndPayServer"/>
</deontic:Permission>
<policy:Policy rdf:ID="PaymentAuthPolicy1">
    <policy:grants rdf:resource="#PayServerPermission"/>
</policy:Policy>
```

El framework de Rei proporciona un motor de razonamiento basado en Prolog para el análisis de las políticas Rei. La funcionalidad que proporciona este framework es la siguiente:

- Consultar la Ontología. El motor de políticas Rei acepta información adicional dependiente del dominio en algún otro lenguaje semántico que pueda ser convertido en tripletas de la forma (sujeto, predicado, objeto). El motor permite consultas acordes al lenguaje Prolog sobre políticas, meta-políticas y conocimiento dependiente del dominio que ha sido cargado en su base de conocimiento.

- Creación de los Actores de las Políticas. Al igual que en KAoS, las políticas pueden ser definidas basadas en roles, además de en instancias concretas de actores.

- Creación de Políticas. La codificación OWL de la política es creada por el usuario. De cualquier modo, Rei también proporciona una notación basada en Prolog.

- Toma de Decisiones de Políticas. Rei proporciona un motor de toma de decisiones que deduce los derechos y obligaciones de los objetos en el

sistema gestionado en respuesta a solicitudes que especifican el estado actual del sistema.

- Análisis de Políticas. Rei usa un motor de razonamiento de Prolog para analizar la especificación de políticas y detectar conflictos de modalidad. Estos conflictos son resueltos con las meta-políticas que fuerzan al motor de políticas a tomar una decisión positiva o negativa. Dado que el motor soporta razonamiento deductivo sobre las reglas, los conflictos pueden ser detectados y resueltos en tiempo de ejecución.

Por último destacar que el framework de Rei no proporciona un modelo de aplicación. De hecho, el motor de políticas no ha sido diseñado para aplicar políticas sino solo para razonar sobre ellas y responder a consultas. Otra limitación es la falta de soporte para el refinamiento de políticas y la ausencia de herramientas para ayudar a los administradores a especificar reglas de políticas.

2.3.3. RuleML/SWRL

La iniciativa Rule Markup Language (RuleML) [53] define una familia de lenguajes de reglas basado en XML. Dentro de esta iniciativa el grupo de trabajo Policy RuleML [51] propone el uso de RuleML como un medio para intercambiar información de políticas entre diferentes sistemas de gestión. Este grupo está centrado inicialmente en aplicaciones en el área de servicios financieros, aunque su propuesta puede ser extendida a otros dominios de aplicación como la Seguridad.

Policy RuleML tiene como principal motivación el desarrollo de RuleML como un medio de sintaxis intermedia e interoperabilidad semántica para intercambio de información de políticas entre los lenguajes, estándares o mecanismos, tanto los existentes como los que puedan ser desarrollados en un futuro. Este grupo, aunque aún no dispone de ninguna propuesta para conseguir este objetivo, tiene previsto el desarrollo de herramientas para facilitar el intercambio de políticas entre diferentes sistemas haciendo uso de tecnologías XML como son las transformaciones XSL.

Inicialmente, Policy RuleML planea representar políticas usando la jerarquía de reglas de RuleML. Esta jerarquía de reglas incluye reglas de reacción (reglas evento-condición-acción), reglas de transformación (reglas función-ecuación), reglas de derivación (reglas implicación-inferencia), que también pueden ser especializadas para hechos (reglas de derivación sin premisas) y para consultas (reglas de derivación sin conclusión). Una vista gráfica de las reglas RuleML es el árbol de reducción que se muestra en la figura 2.12.

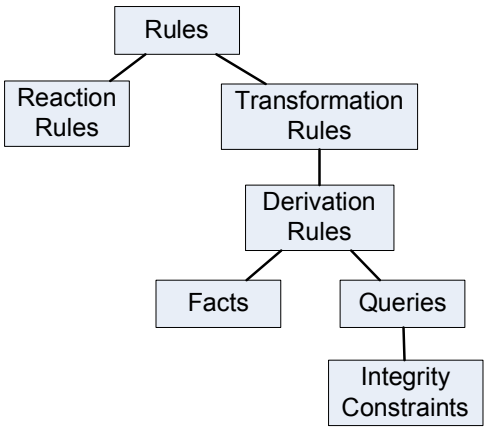

Figura 2.12: *Vista gráfica de reglas RuleML*

RuleML permite la definición de reglas y hechos en XML para describir restricciones. Una primera propuesta en Policy RuleML [86] distingue entre los siguientes hechos/reglas para la representación de políticas:

- Hechos y reglas estructurales/organizacionales. Estas reglas son usadas para codificar ontologías específicas del dominio.
- Hechos y reglas de definición de servicio, disponen de enlaces a las reglas y hechos estructurales.
- Hecho y reglas específicas de la tarea, proporcionadas por los clientes del servicio.

El beneficio de clasificar las reglas de políticas de esta forma es que una organización puede utilizar una ontología común que puede ser compartida entre servicios y clientes de servicios. Los hechos y reglas (es decir, las políticas) informan a los clientes sobre el uso de los servicios para alcanzar los objetivos de negocio.

Consideremos un ejemplo para ilustrar la representación de políticas en RuleML con el lenguaje ur-datalog [53]. Este ejemplo es relativo al caso de estudio descrito anteriormente.

Listado 2.7: *Ejemplo de política en RuleML*

```
1 <rulebase direction="backward">
2 <imp>
3   <_head>
4       <atom>
5          <_opr href="#GrantedAccess"/>
6          <var>requester</var>
7          <var>server</var>
```

```
        </atom>
    </_head>
    <_body>
        <and>
            <atom>
                <_opr href="#isMember"/>
                <var>requester</var>
                <ind>PayCustomer</ind>
            </atom>
            <atom>
                <_opr href="#isMember"/>
                <var>server</var>
                <ind>PayServer</ind>
            </atom>
        </and>
    </_body>
</imp>
</rulebase>
```

Otra propuesta dentro de la iniciativa RuleML es el lenguaje SWRL [52], acrónimo de Semantic Web Rule Language. Este lenguaje está basado en una combinación del lenguaje OWL con el lenguaje RuleML datalog. SWRL proporciona una sintaxis abstracta para escribir reglas que incluyen ontologías OWL. Esto lo consigue extendiendo la sintaxis abstracta de OWL para incluir una sintaxis abstracta de alto nivel para reglas Horn de la forma if-then.

En el mismo sentido que lo hace Policy RuleML, las reglas SWRL pueden ser usadas para la representación de políticas. Los átomos en las reglas SWRL pueden ser de la forma C(x), P(x,y), sameAs(x,y) o differentFrom(x,y), donde C es una descripción OWL, P es una propiedad OWL, y x,y son variables, individuos OWL o valores de datos OWL.

SWRL es definido mediante una sintaxis XML basada en RuleML y la representación XML de OWL. La sintaxis de la regla es ilustrada con el siguiente ejemplo relativo al caso de estudio descrito anteriormente.

Listado 2.8: *Ejemplo de política en SWRL*

```
<ruleml:imp>
    <ruleml:_head>
        <swrlx:individualPropertyAtom
            swrlx:property="GrantedAccess">
            <ruleml:var>requester</ruleml:var>
            <ruleml:var>server</ruleml:var>
        </swrlx:individualPropertyAtom>
    </ruleml:_head>
    <ruleml:_body>
        <swrlx:classAtom>
```

```
11          <owlx:Class owlx:name="User" />
12          <ruleml:var>requester</ruleml:var>
13       </swrlx:classAtom>
14       <swrlx:classAtom>
15          <owlx:Class owlx:name="Server" />
16          <ruleml:var>server</ruleml:var>
17       </swrlx:classAtom>
18       <swrlx:individualPropertyAtom swrlx:property="Member">
19          <ruleml:var>requester</ruleml:var>
20          <owlx:Individual owlx:name="#PayCustomer" />
21       </swrlx:individualPropertyAtom>
22       <swrlx:individualPropertyAtom swrlx:property="Member">
23          <ruleml:var>server</ruleml:var>
24          <owlx:Individual owlx:name="#PayServer" />
25       </swrlx:individualPropertyAtom>
26    </ruleml:_body>
27 </ruleml:imp>
```

Tanto el trabajo de Policy RuleML como la propuesta para la representación de políticas mediante SWRL, no definen un framework de gestión de políticas y centran la motivación principal del uso de estos lenguajes en el intercambio de políticas entre sistemas. Sin embargo, esto facilita la integración de este lenguaje en sistemas de gestión de políticas ya creados y, en particular, en sistemas de análisis de conflictos de políticas.

2.4. Comparativa

Esta sección pretende hacer un análisis comparativo entre los diferentes lenguajes de políticas presentados en la sección anterior tomando como criterios fundamentales los requerimientos para un lenguaje de políticas establecidos en la primera sección de este capítulo. Presentamos en primer lugar una comparativa entre los lenguajes de políticas no semánticos y los lenguajes de políticas semánticos donde resaltamos las ventajas que ofrecen estos últimos. Seguidamente destacamos las principales diferencias y deficiencias de cada uno de los lenguajes semánticos presentados.

Esta comparativa toma como base trabajos previos de otros autores como son un informe técnico presentado en [11] donde se hace un análisis de diversos lenguajes de políticas entre los que se encuentran Ponder y XACML; y otro trabajo destacado presentado en [90] que realiza un análisis comparativo entre Ponder, Rei y KAoS en el cual se discuten sus principales diferencias. Basándonos en estos trabajos y en nuestra investigación que incluye el análisis de CIM Policy y RuleML/SWRL [33, 32], la tabla 2.1 hace un resumen de nuestra comparativa entre los lenguajes de políticas no

Tabla 2.1: Tabla comparativa entre lenguajes semánticos y no semánticos

	Lenguajes semánticos	**Lenguajes no-semánticos**
Abstracción	Multiples niveles	Medio y bajo nivel
Extensibilidad	Fácil y en tiempo de ejecución	Complejo y en tiempo de compilación
Legibilidad	Herramientas especializadas	Directa
Interoperabilidad	Mediante una ontología común	Mediante interfaces
Análisis de conflictos	Fácil, directamente soportada	Compleja
Acceso a políticas	Consulta a la ontología	Mediante una API especifica
Aplicación	Compleja	Fácil

semánticos y los lenguajes de políticas semánticos.

El conjunto de lenguajes de políticas presentados, tanto semánticos como no semánticos, cumplen con dos requerimientos básicos que son definir una semántica clara y bien definida y ser independientes del dominio administrativo y, en particular, de los fabricantes/proveedores de los dispositivos y servicios. Aunque difieren de manera importante en otra serie de características que comentamos a continuación:

- **Abstracción**. Los lenguajes semánticos tienen la habilidad de analizar políticas relativas a entidades descritas en un alto nivel de abstracción, mientras los no semánticos tales como Ponder y XACML necesitan la identificación de los recursos del sistema y además deben prestar atención a los detalles de implementación. Por ejemplo, en la política expresada en XACML usando nuestro caso de estudio, la etiqueta <Resources>es usada para identificar los recursos mediante su URI. En cambio, en SWRL, las entidades son descritas mediante la ontología facilitando un mayor nivel de abstracción y, por tanto, una mayor potencia semántica en su descripción.

- **Extensibilidad**. El uso de ontologías permite añadir nueva información de políticas en tiempo de ejecución en los lenguajes semánticos. La ontología puede ser extendida en cualquier momento y esto no involucra el rediseño o la redefinición del lenguaje de políticas.

En cambio en los lenguajes no semánticos, una extensión en el modelo de información o en el formato de la política debe hacerse en tiempo de compilación ya que su realización en tiempo de ejecución pueden provocar errores en el formato de las políticas ya definidas. Esto hace que la extensibilidad sea más compleja y propensa a errores en los lenguajes no semánticos.

- **Legibilidad**. Las políticas definidas mediante un lenguaje semántico requiere el uso de herramientas especializadas para ser leídas e interpretadas por el administrador. A modo de ejemplo, podemos comparar la legibilidad de la política Ponder con la política Rei en nuestro caso de estudio. Aunque también es cierto que los lenguajes no semánticos con representación XML de políticas como XACML generan definiciones muy extensas que tampoco facilitan la legibilidad de las políticas.

- **Interoperabilidad**. La interoperabilidad entre lenguajes semánticos se consigue fácilmente compartiendo una ontología común. De esta forma los conceptos de política son comunes entre ambos lenguajes aunque la definición de reglas de políticas sigan metodologías diferentes. En cambio los lenguajes no semánticos precisan de interfaces entre ellos para compartir conceptos y reglas de política. Por ejemplo, la interoperabilidad entre dos aplicaciones heterogéneas, una usando Ponder y otra usando XACML debería estar basada en la creación de interfaces complejos para permitir el intercambio de políticas entre ellas. En cambio, una usando KAoS y otra usando SWRL sólo necesitan tener una ontología común para interoperar entre ellas.

- **Análisis de conflictos**. En los lenguajes semánticos el uso de ontologías facilita el razonamiento sobre las políticas y, por tanto, facilita la detección de conflictos. En cambio en los lenguajes no semánticos es complejo introducir técnicas para el análisis de conflictos y la mayoría de ellos no las incorporan. El único lenguaje no semántico presentado que introduce capacidades de análisis de conflictos es Ponder pero lo hace de una manera propietaria.

- **Acceso a información**. El acceso a la información de la política y su reutilización entre entidades es más simple mediante ontologías. En cambio en los lenguajes no semánticos se debe recurrir al camino tradicional mediante APIs específicas. Por ejemplo, es simple usar un lenguaje de consulta como RDQL [99] o SPARQL [102] para consultar a una ontología utilizada en SWRL sobre la relación entre dos sujetos

Tabla 2.2: Tabla comparativa de los lenguajes semánticos

	KAoS	Rei	RuleML/SWRL
Base Semántica	Ontología	Ontología + Lógica deóntica	Ontología + Reglas Horn
Tipos de políticas	Autorización Obligación	Autorización Obligación	Depende de la ontología
Lenguaje de especificación	OWL	Sintaxis tipo Prolog + OWL	Sintaxis tipo Prolog + OWL
Herramientas especificación	KPAT	No	No
Razonamiento	Motor KAoS	Motor Prolog	Motor externo
Aplicación	Soportada	Funcionalidad externa	Funcionalidad externa

utilizados en la definición de dos políticas diferentes. No ocurre lo mismo en Ponder, donde la realización de consultas sobre la información de la política depende de una API específica.

- **Aplicación**. La capacidad de los lenguajes semánticos para expresar políticas en múltiples niveles de abstracción hace que los mecanismos de automatización de la política para su aplicación sean complejos. En cambio en los lenguajes no semánticos la aplicación de políticas es fácil ya que en la misma definición tienen en cuenta los detalles de implementación. Por ejemplo, en XACML los recursos quedan explícitamente definidos en la política.

En resumen, los lenguajes semánticos destacan por su capacidad de abstracción, su extensibilidad, su interoperabilidad y destacan fundamentalmente por sus facilidades para el análisis de conflictos y el acceso a la información de las políticas. En cambio, los lenguajes no semánticos lo hacen por facilitar los mecanismos de aplicación de la política y porque son más legibles en general.

Parece que los lenguajes semánticos presentan mayor número de ventajas que los no semánticos. Siguiendo con nuestros análisis comparativo presentamos la tabla 2.2 donde comparamos entre sí los lenguajes semánticos descritos.

Muchos aspectos pueden ser identificados como parte de esta comparación; algunos de ellos son comunes y han sido resaltados

anteriormente. Aunque entre ellos se diferencian en otra serie de características que comentamos a continuación:

- **Base Semántica**. KAoS está basado en una ontología que define los conceptos relativos a las políticas. También Rei está basado en una ontología pero utiliza conceptos de lógica deóntica para definirla. Por su parte, RuleML/SWRL permite la definición de políticas mediante reglas lógicas y usa una ontología para la definición de los elementos de las políticas. Esta ontología no es definida junto con el lenguaje y debe el administrador definirla o utilizar una existente.

- **Tipos de políticas**. KAoS y Rei permiten la definición de políticas de autorización y obligación tal y como sus ontologías establecen. En cambio, RuleML/SWRL no dispone de una ontología que establezca a priori los tipos de políticas que pueden ser definidos y por tanto depende de la ontología utilizada.

- **Lenguaje de especificación**. Todos los lenguajes presentados utilizan una representación OWL para la ontología. Además, Rei también proporciona una notación basada en Prolog. Y por su parte, RuleML/SWRL define las reglas de políticas con una sintaxis tipo Prolog.

- **Herramientas para la especificación**. Únicamente KAoS dispone de una herramienta para la definición de políticas y que sirva de interfaz gráfica al administrador. Ni Rei ni RuleML/SWRL proporcionan herramientas para la especificación de políticas.

- **Razonamiento**. KAoS y Rei disponen de un motor de razonamiento para el análisis y verificación de las políticas. En cambio RuleML/SWRL no está integrados directamente con ningún motor de razonamiento, aunque sus características lo hacen fácilmente integrable en motores ya existentes.

- **Aplicación**. Sólo KAoS dispone de mecanismos para la aplicación de la política. En cambio, Rei y RuleML/SWRL carecen de ellos ya que ambos no fueron diseñados para aplicar políticas.

En resumen, KAoS presenta una solución completa que incluye desde el lenguaje de políticas a la aplicación de las políticas, mientras el resto de avances carecen de algún componente (por ejemplo herramientas para la especificación de políticas). Rei ha sido diseñado para razonar sobre las políticas y responder a consultas y no proporciona ningún modelo de

aplicación. Y RuleML/SWRL está orientado para ser usado en el intercambio de políticas entre sistemas. Aún así RuleML/SWRL extiende el conjunto de axiomas de OWL incluyendo reglas Horn lo cual lo hace más potente semánticamente que KAoS y Rei. Además una característica fundamental de RuleML/SWRL es que no está limitado a políticas de autorización y obligación como ocurre en Rei y KAoS. Por lo tanto, y como resultado de esta comparativa, concluimos que RuleML/SWRL debe de constituir la base para el lenguaje de políticas semántico que se va a presentar en esta tesis doctoral.

2.5. Conclusiones

Este capítulo ha descrito los conceptos fundamentales del proceso de definición de políticas y el papel que juega el lenguaje de políticas en este proceso para la definición de políticas formales. Del mismo modo se han descrito los requerimientos de un lenguaje de políticas de seguridad y como conseguir mediante este lenguaje una representación formal de políticas utilizando tanto una representación basada en lógica como una basada en ontología. También, se han presentado los principales tipos de políticas de seguridad que un lenguaje de políticas puede llegar a especificar.

Además, este capítulo ha presentado y comparado los lenguajes de políticas de seguridad más relevantes existentes en la actualidad; los lenguajes no semánticos Ponder, XACML, WS-Policy y CIM Policy y los lenguajes semánticos KAoS, Rei y RuleML/SWRL. Para esta comparación se ha utilizado un caso de estudio que nos ha permitido comparar sus representaciones y características. Las conclusiones de esta comparación determinan que los lenguajes semánticos son los más adecuados para la definición de políticas de seguridad fundamentalmente por sus facilidades para el análisis de conflictos y el acceso a la información de las políticas. Entre los lenguajes semánticos, RuleML/SWRL parece el más potente semánticamente ya que incluye soporte para definir reglas Horn aunque carece de una ontología sobre la cual se definan los conceptos relativos a las políticas. El uso de una ontología adecuada en RuleML/SWRL puede permitir definir políticas de seguridad de manera más fácil a como se consigue con KAoS o Rei.

Capítulo 3

Propuesta de un Lenguaje de Políticas enriquecido Semánticamente para la Gestión de la Seguridad

Este capítulo presenta una propuesta de lenguaje semántico de políticas que mantenga las ventajas de los lenguajes semánticos presentados en el capítulo anterior pero a la vez salve sus principales desventajas y limitaciones. Para ello nuestra propuesta toma como base el lenguaje SWRL para la representación de reglas de políticas y el modelo de información CIM como ontología común. Hemos denominado a esta combinación SWRL-CIM.

SWRL-CIM combina las ventajas de ambos, SWRL y CIM; por un lado SWRL es semánticamente más potente que el resto de lenguajes al incluir la definición de reglas Horn, además de no estar limitado a expresar ningún tipo de política y de ser fácilmente integrable en los motores de inferencia actuales; por otro lado CIM proporciona un modelo de información común y extendido que representado formalmente usando OWL, tal y como proponemos en este capítulo, puede ser fácilmente integrado con SWRL para la definición de los componentes de las políticas. Además veremos como SWRL-CIM facilita la aplicación de técnicas de razonamiento basado en reglas que nos permiten especificar un procedimiento para la detección de conflictos sobre las políticas definidas.

3.1. Introducción

Los lenguajes semánticos parecen ser, tras el análisis comparativo presentado en el capítulo anterior, los lenguajes con mejores características para la representación de políticas en un sistema de gestión de la seguridad. Aunque existen diversos lenguajes semánticos, el lenguaje SWRL destaca del resto por su potencial semántico por el hecho de extender OWL para incluir la definición de reglas Horn, ya que OWL no permite especificar reglas directamente. Otras propuestas alternativas a SWRL, como por ejemplo Rei, utilizan OWL para definir la estructura de las reglas como clases en la ontología pero de forma limitida y restringida.

Aunque SWRL destaca por su potencial semántico, sin embargo no define una ontología común para la definición de las políticas, pero una ontología común es necesaria para ser compartida entre los diferentes elementos del dominio administrativo y poder interpretar las políticas con un significado común. Un estándar que proporciona un modelo común para describir los conceptos necesarios a ser considerados dentro de un dominio de política es el modelo CIM [18]. Este modelo proporciona una ontología semi-formal que no soporta interoperabilidad y razonamiento. Estas limitaciones son debidas, en parte, a las restricciones impuestas por los lenguajes en los cuales el modelo CIM es normalmente representado (por ejemplo, XML mediante un DTD o mediante un XSD). Para superar las desventajas mencionadas en esta investigación proponemos modelar, representar y compartir CIM en la forma de una ontología formal. En este sentido, la primera sección del capítulo presenta la ontología CIM-OWL (es decir, la representación de la ontología CIM en OWL).

La representación OWL de CIM permite integrar su uso en la especificación de reglas con SWRL. Esta combinación de CIM-OWL y SWRL que denominamos SWRL-CIM constituye la base de nuestra propuesta de lenguaje de representación de políticas. A continuación presentamos las principales características de SWRL-CIM tomando como base los requerimientos de un lenguaje de políticas establecidos en la sección 2.1.1.

- **Semántica clara y bien definida.** SWRL define una semántica clara y no ambigua para la definición de reglas, mientras CIM-OWL lo hace para la definición de los conceptos relativos a las políticas tales como role, privilegio, identidad, etc.

- **Múltiples niveles de abstracción.** El nivel de abstracción de las políticas viene determinado por el grado de detalle que sea utilizado en la definición de las reglas de políticas. Por ejemplo, si definimos

una política donde utilizamos conceptos de CIM-OWL que especifican detalles concretos de los recursos como son su dirección IP o el algoritmo de cifrado utilizado, entonces estamos definiendo políticas de bajo nivel de abstracción. Pero si por el contrario utilizamos conceptos de menor detalle tecnológico como son roles o privilegios, estamos definiendo políticas de alto nivel de abstracción.

- **Extensibilidad.** La ontología CIM-OWL permite añadir nuevos conceptos de políticas en tiempo de ejecución. La ontología puede ser extendida en cualquier momento y esto no involucra el rediseño o la redefinición del lenguaje de políticas. De esta manera la ontología puede ser extendida cuando se necesita representar un recurso que no dispone de conceptualización en la ontología.

- **Interoperabilidad.** La interoperabilidad entre lenguajes semánticos se consigue fácilmente compartiendo una ontología común. En este caso, CIM-OWL puede ser utilizada como ontología común entre diferentes lenguajes aunque la definición de reglas de políticas sigan metodologías diferentes.

- **Razonamiento.** SWRL-CIM no está integrado directamente con ningún motor de razonamiento, pero SWRL es fácilmente integrable en motores ya existentes que permitan razonamiento basado en reglas. Por otro lado, un motor de inferencia podría hacer razonamiento sobre los conceptos CIM-OWL para conseguir acceso a la información de las políticas.

Estas características del lenguaje SWRL-CIM le permiten la definición de cualquier tipo de política de seguridad ya que cualquier limitación en la especificación puede ser resuelta con la extensión de los conceptos representados por CIM-OWL. Además, su facilidad para integrarse con motores de razonamiento permiten el análisis de conflictos de políticas de una manera directa mediante razonamiento basado en reglas y razonamiento basado en ontología.

Por otro lado, SWRL-CIM tiene las limitaciones típicas de un lenguaje semántico como son la complejidad de los mecanismos para la aplicación de políticas y las especificaciones de políticas muy extensas que no facilitan la legibilidad. En este sentido estas limitaciones han sido analizadas y resueltas para ciertos escenarios como los que se presentarán en el siguiente capítulo.

En las siguientes secciones presentamos de manera más detallada SWRL-CIM. En primer lugar en la sección 2 describimos como obtener una representación formal de CIM usando OWL (CIM-OWL). Además en esta

sección introducimos una representación semi-formal de CIM usando XML que denominamos xCIM y que fue el primer paso en nuestra investigación y que supone una versión mejorada de las propuestas basadas en XML del DMTF para la representación de CIM usando XML. En la sección 3 nos centramos en cómo SWRL-CIM puede representar diferentes tipos de políticas de seguridad. Aquí describimos los principales conceptos de CIM utilizados en cada tipo de política y cómo las reglas SWRL permiten definir políticas que combinen todos estos conceptos. A continuación presentamos como SWRL-CIM se integra fácilmente en motores de inferencia que permiten el razonamiento sobre las políticas en la sección 4. Utilizando estas técnicas de razonamiento definimos un procedimiento de detección de conflictos que en la sección 5 presentamos junto a la clasificación de los diferentes tipos de conflictos que pueden aparecer.

3.2. Representación formal de CIM

CIM es un estándar del DMTF que aplica las técnicas básicas de modelado del paradigma orientado a objetos para proporcionar una definición común de la información relativa a la gestión de redes y sistemas. El esquema CIM es organizado en subdivisiones entre la que destaca Policy CIM que en el capítulo anterior hemos presentado como una solución para la representación de políticas. Policy CIM es sólo una parte del esquema CIM. Nuestra propuesta no determina sólo el uso de esta subdivisión sino que permite el uso de cualquiera de los conceptos definidos en sus catorce subdivisiones.

La característica más importante de CIM es que propone un modelo independiente de cualquier implementación o representación específica. De cualquier modo, para que un modelo de información sea útil tiene que ser representado usando alguna sintaxis concreta como son las definidas para MIB [6], PIB [9], MOF, XML y OWL (véase figura 3.1). En este sentido, CIM utiliza MOF como formato base para la representación de objetos CIM, pero este formato no se integra fácilmente en entornos de aplicación como por ejemplo Servicios Web. Para ello dentro del DMTF surgieron nuevas iniciativas para conseguir una representación de CIM usando XML, ya que así se consigue una sintaxis estructurada válida para aplicaciones y servicios web. La primera iniciativa fue WBEM con XML-CIM que es una representación de CIM usando XML junto con un mecanismo de transporte de las operaciones CIM sobre HTTP. Posteriormente a esta iniciativa surgió una nueva propuesta denominada WS-CIM [45] que también presenta una representación de CIM usando XML pero donde las operaciones CIM son definidas mediante WSDL.

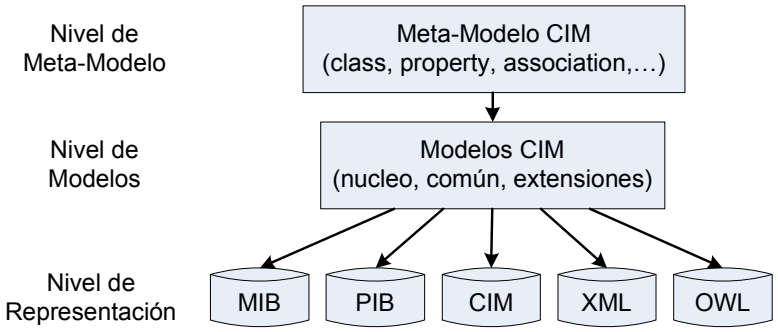

Figura 3.1: *Niveles del modelo CIM*

Comparando ambas iniciativas encontramos dos enfoques distintos en el momento de representar CIM. Por un lado, XML-CIM propone una representación de objetos CIM basada en meta-esquema. Esta representación define una gramática XML escrita como un DTD (Document Type Definition), donde las clases e instancias CIM son documentos XML válidos para este DTD. En otras palabras el DTD es usado para describir de manera genérica la noción de una clase o instancia CIM. Los nombres de los elementos CIM son traducidos a valores de elementos o atributos XML, en cambio de especificar nombres de elementos XML. Por el contrario, WS-CIM propone una representación basada en un esquema XML (XSD) para describir las clases CIM; en este avance las instancias CIM son representadas con documentos XML válidos para este esquema. Esencialmente esto significa que cada clase CIM genera su propio fragmento XSD cuyo nombre del elemento XML es el mismo que corresponde al nombre del elemento CIM.

Por ejemplo, el siguiente listado muestra un fragmento de la representación de la clase CIM_Role (que define el concepto de role) utilizando el enfoque XML-CIM basado en meta-esquema.

```
1  <CLASS NAME="CIM_Role">
2      <PROPERTY CLASSORIGIN="CIM_Role"
3                NAME="Name"
4                TYPE="string">
5      </PROPERTY>
6      <PROPERTY CLASSORIGIN="CIM_Role"
7                NAME="BusinessCategory"
8                TYPE="string">
9      </PROPERTY>
10 </CLASS>
```

Y el siguiente muestra un fragmento de la representación de la clase CIM_Role utilizando un enfoque WS-CIM basado en esquema.

```
1  <xs:complexType name="CIM_Role_Type">
```

```
2     <xs:sequence>
3        <xs:element ref="class:Name"/>
4        <xs:element ref="class:BusinessCategory"/>
5     </xs:sequence>
6  </xs:complexType>
7  <xs:element name="CIM_Role" type="class:CIM_Role_Type"/>
```

La representación basada en meta-esquema fue principalmente adoptada por el DMTF, ya que ésta sólo requiere un DTD estandarizado para todo CIM independientemente de la versión del modelo de información usado en una implementación particular. De cualquier modo, WS-CIM identifica mayores beneficios relativos al uso de la representación basada en esquema. Los más importantes son el mayor poder de validación y una representación más intuitiva de los objetos CIM. Además WS-CIM está integrado con las últimas propuestas definidas en WSDM (Web Services Distributed Management) [73] tales como WS-ResourceFramework, WS-Notifications y WS-MetadataExchange. Por tanto, lo convierten en la alternativa idónea para integrar el uso de CIM en entornos de aplicación basados completamente en Servicios Web. Pero para ciertas aplicaciones el uso de WS-CIM introduce demasiada complejidad, aunque el uso de una representación de CIM basada en esquema sea la más adecuada. Esto ocurre, por ejemplo, cuando queremos utilizar los objetos CIM como conceptos de las políticas en un lenguaje de políticas basado en Policy CIM.

En este sentido y para solventar estas deficiencias, en esta investigación proponemos una representación que denominamos xCIM basada en esquema pero que no hace uso de ninguna especificación de Servicios Web. Así conseguimos una representación sin dependencias a especificaciones externas y con la ventajas que se derivan de estar basada en esquema. La representación xCIM la presentamos en la subsección 3.2.2.

Una representación XML permite sacar ventaja de todas las tecnologías XML como son transformaciones XSL [96] y las consultas mediante XPath [95], pero aún así tiene limitada su habilidad para soportar ciertos requerimientos importantes para una lenguaje de políticas. Así una representación CIM usando XML no puede ser combinada arbitrariamente con otras especificaciones de manera flexible y, con respecto a operaciones más avanzadas, las especificaciones codificadas en XML no abarcan los constructores para facilitar tareas como parsing, deducción lógica o interpretación semántica. En este sentido y para integrar estas características, otra representación específica de CIM podría ser realizada usando OWL. OWL proporciona un semántica simple para especificar objetos CIM y las relaciones entre ellos, y además puede ser representado en sintaxis XML por lo que aún con OWL podemos sacar partido de las tecnologías XML.

Es importante subrayar que OWL proporciona tres sublenguajes con un grado incremental de expresividad; estos son: OWL Lite que soporta restricciones semánticas simples (por ejemplo, soporta restricciones de cardinalidad pero sólo con valores 0 ó 1); OWL DL que permite la máxima expresividad manteniendo la completitud computacional (todas las conclusiones son computables) y la decibilidad (todas las operaciones computacionales acaban en un tiempo finito); y OWL Full que permite la máxima expresividad posible pero sin garantías computacionales.

Así existen diversas propuestas de representación CIM usando OWL. Destaca el trabajo realizado en [14] que propone una representación parcial de CIM donde no todos los clasificadores son representados y, además, usa OWL-Full; esto significa que necesita la mayor expresividad posible de OWL e implica que no se pueda integrar en herramientas de razonamiento ya que no existe un software de razonamiento capaz de soportar completamente OWL Full. Otro trabajo destacado es [84] que propone una primera aproximación a una representación de CIM usando OWL-DL donde tampoco se proporciona una representación para todos clasificadores y se propone una representación para las asociaciones CIM incompleta donde existe la posibilidad de perder la integridad de las asociaciones. Para salvar las limitaciones de las propuestas actuales y utilizando como base la experiencia en la definición de xCIM, nuestra propuesta de representación CIM usa OWL-DL y proporciona una representación del esquema incluyendo a gran parte de los clasificadores. La subsección 3.2.3 detalla como conseguimos esta representación que denominamos CIM-OWL.

Tanto CIM-OWL como xCIM son avances que no introducen información adicional y la representación de los constructores CIM es realizada uno-a-uno (es decir, cada constructor tiene una representación única) con las excepciones que son apuntadas más tarde. La representación uno-a-uno soporta no sólo el esquema CIM existente, sino cualquier extensión hecha al esquema. Si hay una necesidad de extensiones (por ejemplo, nuevas clases CIM), ésta puede ser hecha bien usando CIM con una representación OWL o XML directamente o bien usando CIM con otra representación y luego realizando una transformación.

Antes de describir nuestras propuestas xCIM y CIM-OWL, incluimos una subsección breve donde se presenta el meta-esquema CIM y sus elementos sobre los cuales se define la representación en cada una de nuestras propuestas.

3.2.1. Meta-esquema CIM

El meta-esquema CIM [21] establece una definición formal de cómo deben ser definidos los conceptos en el modelo CIM. Para ello establece los términos usados para expresar el modelo y su uso y semántica. Los elementos del modelo son esquemas, clases, propiedades y métodos. El modelo también soporta indicaciones y asociaciones como tipos de clases y soporta referencias como tipos de propiedades. La figura 3.2 muestra la estructura del meta-esquema.

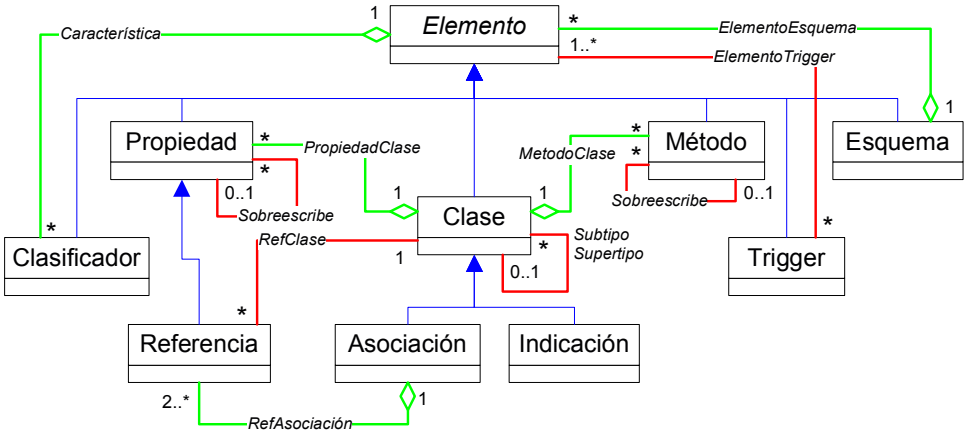

Figura 3.2: *Estructura del meta-esquema de CIM*

- Un *esquema* es un grupo de clases que describen a un conjunto de objetos para ser gestionados. Los nombres de las clases deben ser únicos dentro sus propios esquemas.

- Una *clase* es una colección de objetos que soportan el mismo tipo, esto es, las mismas propiedades y métodos. Las clases pueden ser organizadas en una jerarquía de generalización que representa relaciones de subtipo entre las clases, pero esta jerarquía no soporta herencia múltiple. Además una clase puede participar en las asociaciones siendo el objetivo de las referencias de la asociación.

- Una *indicación* es un objeto creado como resultado de un cambio de estado (trigger) de una clase o una propiedad de clase. Dado que las indicaciones son subtipos de clases, pueden tener propiedades y métodos, y ser organizadas en una jerarquía.

- Una *asociación* es una clase que contiene dos o más referencias. Ésta representa una relación entre dos o más objetos. Sólo las asociaciones

pueden tener referencias y una asociación no puede ser una subclase de una clase que no es una asociación.

- Los *clasificadores* son usados para caracterizar los elementos. Los clasificadores proporcionan un mecanismo que hace el meta-esquema extensible de una manera controlada y limitada. Además es posible añadir nuevos tipos de clasificadores. En el Anexo A puede encontrarse una tabla resumen con el listado de clasificadores y su significado.

Así, los principales componentes de una especificación son descripciones de clases, asociaciones, propiedades, referencias, métodos, clasificadores y las propias instancias de las clases.

3.2.2. Representación XML de CIM (xCIM)

En esta sección presentamos una representación de CIM usando XML que denominamos xCIM. Nuestra propuesta define un esquema XML (XSD) para describir los elementos CIM donde las instancias CIM son documentos XML válidos para este esquema.

En xCIM la representación se puede conseguir de diferentes formas. Así una representación de un concepto CIM en un constructor específico indica que existe una correspondencia semántica total. En otros casos el concepto CIM es representado utilizando un conjunto de constructores que aporta aproximadamente la semántica del concepto. En otros casos no es posible representar el concepto o no se ha considerado su representación.

Tabla 3.1: *Representación xCIM de los conceptos CIM*

Concepto CIM	Total	Aprox.	No definido
Tipos de datos	√		
Clase	√		
Propiedad	√		
Instancia	√		
Método			√
Generalización	√		
Identificación	√		
Asociación		√	
Indicación			√
Cardinalidad		√	
Clasificadores	√	√	√

La tabla 3.1 muestra un resumen de los principales conceptos CIM y hasta que nivel se consigue su representación en xCIM.

Tipos de datos

La tabla 3.2 presenta los tipos de datos CIM con su nombre y su significado junto con la representación en tipos de datos estandarizados por W3C usados en el esquema XML.

Tabla 3.2: *Representación xCIM de los tipos de datos*

Tipo de dato CIM	Interpretación	Tipo de dato W3C
uint8	Entero sin signo de 8-bit	unsignedByte
sint8	Entero con signo de 8-bit	byte
uint16	Entero sin signo de 16-bit	unsignedShort
sint16	Entero con signo de 16-bit	short
uint32	Entero sin signo de 32-bit	unsignedInt
sint32	Entero con signo de 32-bit	int
uint64	Entero sin signo de 64-bit	unsignedLong
sint64	Entero con signo de 64-bit	long
string	Cadena de texto	string
boolean	Booleano	boolean
real32	Punto flotante IEEE 4-byte	float
real64	Punto flotante IEEE 8-byte	double
datetime	Cadena con una fecha y hora	string
char16	Carácter de 16-bit	string

Clases, propiedades e instancias

Una clase CIM y sus propiedades pueden ser representadas directamente a un elemento xsd:complexType. Por ejemplo, la clase CIM_ManagedElement (que proporciona una superclase común para todas las clases CIM) tiene la siguiente descripción en formato MOF:

```
1 [Abstract, Version ( "2.10.0" ), Description ("...")]
2 class CIM_ManagedElement {
3       [Description ("..."), MaxLen ( 64 )]
4    string Caption;
5       [Description ("...")]
6    string Description;
7       [Description ("...")]
```

```
8     string ElementName;
9 };
```

La representación xCIM de CIM_ManagedElement según nuestra propuesta sin considerar los clasificadores sería la siguiente:

```
1 <xs:complexType name="CIM_ManagedElement" abstract="true">
2         <xs:sequence>
3                 <xs:element name="Caption" type="xs:string"/>
4                 <xs:element name="Description" type="xs:string"/>
5                 <xs:element name="ElementName" type="xs:string"/>
6         </xs:sequence>
7 </xs:complexType>
```

Las propiedades de la clase son agrupadas en un elemento xs:sequence creando un elemento xs:element para cada propiedad indicando el tipo de dato correspondiente.

Una instancia de clase es el documento XML generado usando el fragmento del esquema que corresponda a la clase. Por ejemplo, ésta es la representación de una instancia para la clase CIM_Role.

```
1 <xCIM_Role>
2         <CreationClassName>CIM_Role</CreationClassName>
3         <Name>Admin</Name>
4         <CommonName>Administrador</CommonName>
5 </xCIM_Role>
```

Solo las clases que no son abstractas como CIM_Role pueden ser instanciadas. Si una clase es abstracta se indica mediante un clasificador concreto.

Generalización

Una generalización tiene una representación directa en un esquema XML con la declaración de un elemento xs:extensión indicando la clase base sobre la que se está extendiendo. Pero esta representación no es conveniente al representar CIM ya que imposibilita la definición de propiedades en la clase que sobrescriban a propiedades de las superclases.

La solución adoptada en xCIM es definir dentro de la clase todos las propiedades heredadas y no utilizar el elemento xs:extensión. Por ejemplo, la clase CIM_Role extiende la clase CIM_Collection y ésta a su vez extiende a CIM_ManagedElement.

```
1 [Version ( "2.8.1000" ), Description ("...")]
2 class CIM_Role : CIM_Collection {
3       [Key, Description ("..."), MaxLen ( 256 )]
4    string CreationClassName;
5       [Key, Description ("..."), MaxLen ( 1024 )]
```

```
6     string Name;
7        [Description ("..."), MaxLen ( 128 )]
8     string BusinessCategory;
9        [Required, Description ("...")]
10    string CommonName;
11 };
```

La representación xCIM de CIM_Role sin considerar los clasificadores sería la siguiente, donde las propiedades Caption, Description y ElementName son heredadas:

```
<xs:complexType name="CIM_Role">
 <xs:sequence>
        <xs:element name="Caption" type="xs:string"/>
        <xs:element name="Description" type="xs:string"/>
        <xs:element name="ElementName" type="xs:string"/>
        <xs:element name="CreationClassName" type="xs:string"/>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="BusinessCategory" type="xs:string"/>
        <xs:element name="CommonName" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

Identificación

Las instancias de las clases CIM son identificadas de manera única mediante sus propiedades clave, su nombre de clase y un identificador del espacio de nombres. El conjunto de propiedades clave debe ser único para todas las instancias de las clases. La representación XML utiliza el elemento xs:key para identificar únicamente un elemento XML, en nuestro caso una instancia CIM.

Por ejemplo, la clase CIM_Role tiene las propiedades clave CreationClassName y Name. En este caso no existen propiedades clave heredadas, si las hubiera habría que tenerlas en cuenta. La representación de las claves sería la siguiente:

```
<xs:key name="key_CIM_Role">
        <xs:selector xpath="xCIM_Role"/>
        <xs:field xpath="CreationClassName"/>
        <xs:field xpath="Name"/>
</xs:key>
```

Por otro lado, el espacio de nombres proporciona el dominio donde las instancias de las clases son únicas por su propiedades clave. En nuestra representación asumimos un único espacio de nombres y, por tanto, el espacio de nombres es el mismo para todas las instancias CIM y no es útil para la identificación dentro del contexto de la representación.

Una cuestión importante aquí es la representación XML de los caminos CIM usados para apuntar de manera unívoca a un elemento CIM. La sintaxis del camino CIM es:

```
<NombreClase>.<clave1>=<valor1>[,<claveX>=<valorX>]*
```

La representación del camino CIM puede conseguirse utilizando el lenguaje XPath [95] que permite direccionar partes de un documento XML. De manera que la sintaxis del camino CIM usando XPath sería la siguiente:

```
<NombreElemento>'['<clave1>=<valor1> [and <claveX>=<valorX>]*']'
```

Por ejemplo, un camino que selecciona un elemento CIM_Role:

```
CIM_Role.CreationClassName=\"CIM_Role\",Name=\"Admin\"
```

La representación XPath de este camino es la siguiente:

```
CIM_Role[CreationClassName="CIM_Role" and Name="Admin"]
```

Una importante ventaja del uso de la tecnología XPath dentro de la descripción xCIM es que nos facilita la realización de operaciones de direccionamiento y recuperación de documentos xCIM en un repositorio.

Asociaciones/Agregaciones

El modelo CIM usa las clases de asociación para expresar relaciones entre las instancias de las clases. Una asociación permite a dos clases previamente no relacionadas ser asociadas sin modificarlas. Esto se consigue creando una nueva clase de asociación que tiene referencias a cada una de las clases. La referencia es simplemente un puntero a una instancia y su valor es el camino CIM a la instancia.

Por ejemplo, la asociación CIM_MemberOfCollection relaciona la clase CIM_Collection con la clase CIM_ManagedElement. Esta asociación es utilizada para asociar elementos (instancias de CIM_ManagedElement) a una collección (instancia de una subclase de CIM_Collection).

```
1 [Association, Aggregation, Version("2.6.0"), Description("...")]
2 class CIM_MemberOfCollection {
3       [Key, Aggregate, Description ("...")]
4    CIM_Collection REF Collection;
5       [Key, Description ("...")]
6    CIM_ManagedElement REF Member;
7 };
```

La representación xCIM de CIM_MemberOfCollection sin considerar los clasificadores sería la siguiente, donde las propiedades Collection y Member se definen como cadenas de texto.

```
<xs:complexType name="CIM_MemberOfCollection">
   <xs:sequence>
      <xs:element name="Collection" type="xs:string"/>
      <xs:element name="Member" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

Para ver la representación XML de la instancia de una asociación utilizamos el siguiente ejemplo donde se asocia a una identidad con un role. Tanto la identidad como el role son referenciados mediante sus caminos CIM.

```
Instance of CIM_MemberOfCollection
{
   Collection="CIM_Role.
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣CreationClassName=\"CIM_Role\",Name=\"Admin\"";
   Member="CIM_Identity.InstanceID=\"umu:fgarcia\"";
};
```

La representación XML de este ejemplo es la siguiente, donde los caminos CIM se definen usando XPath.

```
<xCIM_MemberOfCollection>
   <Collection>
      xCIM_Role[CreationClassName="CIM_Role" and Name="Admin"]
   </Collection>
   <Member>
      xCIM_Identity[InstanceID="umu:fgarcia"]
   </Member>
</xCIM_MemberOfCollection>
```

Otras soluciones para la representación XML de CIM son el uso de una estructura de árbol para expresar las relaciones y la de-normalización del modelo de información, pero ambas tienen el inconveniente de no poder representar de manera adecuada las relaciones con cardinalidad muchos-a-muchos. En cambio, nuestra representación puede representar de manera adecuada todo tipo de relaciones.

Aún así nuestra propuesta para representar las asociaciones CIM dispone de una serie de limitaciones derivadas de las propias limitaciones de XML. No es posible proporcionar soporte para la cardinalidad de las asociaciones y tampoco a la integridad de las referencias. Ambas pueden ser superadas utilizando XPath como un lenguaje de restricciones. Es decir, la cardinalidad puede ser contemplada definiendo una consulta XPath que compruebe la ocurrencia de una referencia en el conjunto de asociaciones. Por ejemplo, supongamos que una identidad no puede pertenecer a más de un role, es decir una instancia de la clase CIM_Identity no puede tener más de una referencia en el conjunto de instancias de la asociación CIM_MemberOfCollection. Esta restricción de cardinalidad puede expresarse mediante XPath para una

identidad concreta como sigue:

```
Count(xCIM_MemberOfCollection[
        Member='xCIM_Identity[InstanceID="umu:fgarcia"]'
]) <= 1
```

Si esta consulta XPath obtiene un valor de FALSE, significa que no se está cumpliendo con la cardinalidad definida.

Del mismo modo, la integridad de las referencias puede hacerse comprobando que todas las referencias XPath de las asociaciones existen. Para ello debe de realizarse un chequeo de integridad cada vez que una instancia es modificada evaluando las referencias de todas las asociaciones involucradas o relativas a la modificación.

Clasificadores

Hasta ahora hemos visto como representar clasificadores como Key y Association de manera indirecta, ya que por ejemplo para indicar si una clase es una asociación se hace con el clasificador Association, o para indicar que una propiedad es una clave se hace con el clasificador Key. Aún hay muchos clasificadores más de los cuales sólo unos pocos más han sido representados usando XML en nuestra propuesta. Estos son los involucrados en los siguientes conceptos:

- *Ocurrencia de las propiedades.* La ocurrencia de una propiedad en una instancia depende del tipo del atributo. Las propiedades clave y las propiedades requeridas deben aparecer obligatoriamente; el resto de propiedades son opcionales. Además, las propiedades definidas como un array de un tipo de dato básico pueden tener una ocurrencia múltiple mientras el resto de atributos puede tener como mucho una ocurrencia. Para representar la ocurrencia de la propiedades, xCIM usa los elementos minOccurs y maxOccurs para la ocurrencia de los elementos XML que representan a las propiedades de una clase.

- *Descripción.* Una descripción CIM puede aparecer tanto en una clase como en una propiedad. Esta descripción es representada en xCIM usando los elementos xs:annotation y xs:documentation.

- Clase abstracta. Si la clase CIM es abstracta lo representamos en xCIM indicando en el elemento principal de la clase la propiedad abstract.

- *Valor por defecto.* Algunas propiedades CIM tienen un valor por defecto. En xCIM utilizamos la propiedad default del elemento que representa la propiedad para indicar el valor por defecto.

El resto de clasificadores no han sido considerados porque proporcionan información semántica bastante reducida y su inclusión en la representación xCIM introducen mayor complejidad en la propuesta.

3.2.3. Representación OWL de CIM (CIM-OWL)

Una representación XML de CIM como es xCIM nos permite sacar ventaja de las tecnologías XML en la especificación y distribución de políticas de seguridad, pero aún así xCIM tiene ciertas limitaciones como son: no puede ser combinada arbitrariamente con otras especificaciones de manera flexible; y no facilita tareas cruciales en el análisis de políticas como la deducción lógica o interpretación semántica. En este sentido y para integrar estas características, nuestra investigación nos ha llevado a definir una representación de CIM usando OWL que hemos denominado CIM-OWL. CIM-OWL proporciona un semántica simple para especificar objetos CIM y las relaciones entre ellos, y es la representación para la especificación de conceptos dentro del lenguaje semántico de políticas propuesto en esta tesis.

Esta sección trata todos los aspectos de nuestra propuesta CIM-OWL para la representación de CIM usando OWL. Esta propuesta define una representación formal mediante OWL del conjunto de clases e instancias CIM. Al igual que en la sección anterior, utilizamos la tabla 3.3 para resumir cómo se consigue la representación OWL de los principales conceptos CIM.

Tabla 3.3: *Representación CIM-OWL de los conceptos CIM*

Concepto CIM	Total	Aprox.	No definido
Tipos de datos	√		
Clase	√		
Propiedad	√		
Instancia	√		
Método			√
Generalización	√		
Identificación	√		
Asociación	√		
Indicación			√
Cardinalidad	√		
Clasificadores	√	√	√

Tipos de datos

La representación de los tipos de datos CIM en tipos de datos estandarizados por W3C se realiza de igual manera a como se hacía en xCIM (véase tabla 3.2).

Clases, propiedades e instancias

Una clase CIM tiene una representación directa en OWL usando el elemento owl:Class. Por ejemplo, la clase CIM_ManagedElement tiene la siguiente representación OWL sin considerar los clasificadores y las propiedades:

```
<owl:Class rdf:ID="CIM_ManagedElement"/>
```

Las propiedades de la clase son declaradas individualmente mediante el elemento owl:DatatypeProperty indicando con el elemento rdfs:domain la clase a la que pertenece y con el elemento rdfs:range el tipo de dato de la propiedad. Por ejemplo, las propiedades de la clase CIM_ManagedElement tienen la siguiente representación:

```
<owl:DatatypeProperty rdf:ID="Caption">
        <rdfs:domain rdf:resource="#CIM_ManagedElement"/>
        <rdfs:range rdf:resource="xs:string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Description">
        <rdfs:domain rdf:resource="#CIM_ManagedElement"/>
        <rdfs:range rdf:resource="xs:string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ElementName">
        <rdfs:domain rdf:resource="#CIM_ManagedElement"/>
        <rdfs:range rdf:resource="xs:string"/>
</owl:DatatypeProperty>
```

Al definir las propiedades surge el problema de nombres duplicados. Por ejemplo, la clase CIM_Role y la clase CIM_Service tienen la propiedad Name y, por tanto, el mismo ID según nuestra representación. Para evitar esta duplicidad se define una espacio de nombres diferente para cada clase. Estos son generados usando el nombre de la clase, la versión, el prefijo http://ants.um.es y el sufijo .owl. Esto permite a los espacios de nombres tener valores que pueden ser interpretados para encontrar el documento que representa el espacio de nombres. Por ejemplo, el espacio de nombres usado para la clase CIM_ComputerSystem es:

http://ants.um.es/schema_2.12/CIM_ComputerSystem.owl

La representación de una instancia CIM corresponde con la instanciación de una clase OWL. Por ejemplo, ésta es la representación de una instancia para la clase CIM_Role.

```
<role:CIM_Role>
        <role:CreationClassName>CIM_Role</role:CreationClassName>
        <role:Name>Admin</role:Name>
        <role:CommonName>Administrador</role:CommonName>
</role:CIM_Role>
```

Generalización

La generalización puede ser directamente representada en OWL mediante el elemento rdfs:subClass indicando la clase base sobre la que se está extendiendo. Por ejemplo, la clase CIM_Role extiende la clase CIM_Collection; la representación OWL de CIM_Role sin considerar los clasificadores sería la siguiente, donde las propiedades Caption, Description y ElementName son heredadas:

```
<owl:Class rdf:ID="CIM_Role">
        <rdfs:subClassOf rdf:resource="#CIM_Collection" />
</owl:Class>
<owl:DatatypeProperty rdf:ID="Caption">
        <rdfs:domain rdf:resource="#CIM_Role"/>
        <rdfs:range rdf:resource="xs:string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Description">
        <rdfs:domain rdf:resource="#CIM_Role"/>
        <rdfs:range rdf:resource="xs:string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ElementName">
        <rdfs:domain rdf:resource="#CIM_Role"/>
        <rdfs:range rdf:resource="xs:string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="CreationClassName">
        <rdfs:domain rdf:resource="#CIM_Role"/>
        <rdfs:range rdf:resource="xs:string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Name">
        <rdfs:domain rdf:resource="#CIM_Role"/>
        <rdfs:range rdf:resource="xs:string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="BusinessCategory">
        <rdfs:domain rdf:resource="#CIM_Role"/>
        <rdfs:range rdf:resource="xs:string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="CommonName">
        <rdfs:domain rdf:resource="#CIM_Role"/>
```

```
30          <rdfs:range rdf:resource="xs:string"/>
31  </owl:DatatypeProperty>
```

Identificación

Como indicábamos en el apartado anterior, las instancias de las clases CIM son identificadas de manera única mediante sus propiedades clave, su nombre de clase y un identificador del espacio de nombres. Aquí también asumimos un único espacio de nombres. Así el espacio de nombres no es utilizado para la identificación dentro del contexto de la representación.

Para conseguir representar propiedades clave que sean únicas para todas las instancias de las clases, la representación OWL utiliza el elemento owl:FunctionalProperty. Con esto podemos conseguir que una propiedad tenga un valor único en el conjunto de instancias, aunque tiene la limitación de no poder definir un conjunto de claves en una instancia.

Por ejemplo la clase CIM_Role tiene las propiedades clave CreationClassName y Name. La representación de las claves sería la siguiente:

```
1  <owl:DatatypeProperty rdf:ID="CreationClassName">
2     <rdf:type     rdf:resource="&owl;FunctionalProperty" />
3     <rdfs:domain rdf:resource="#CIM_Role"/>
4     <rdfs:range rdf:resource="xs:string"/>
5  </owl:DatatypeProperty>
6  <owl:DatatypeProperty rdf:ID="Name">
7     <rdf:type     rdf:resource="&owl;FunctionalProperty" />
8     <rdfs:domain rdf:resource="#CIM_Role"/>
9     <rdfs:range rdf:resource="xs:string"/>
10 </owl:DatatypeProperty>
```

Por otro lado, los caminos CIM no disponen en nuestra propuesta de ninguna representación. Esto se debe a que su funcionalidad es sustituida por el uso de los identificadores web (URIs) de las instancias.

Por ejemplo, la instancia siguiente de CIM_Role se identifica con la URI indicada en rdf:about:

```
1  <role:CIM_Role
2  rdf:about="http://ants.um.es/schema_2.12/CIM_Role.owl/user">
3     <role:CreationClassName>CIM_Role</role:CreationClassName>
4     <role:Name>User</role:Name>
5     <role:CommonName>Service User</role:CommonName>
6  </role:CIM_Role>
```

Asociaciones/Agregaciones

Como indicábamos anteriormente, en el modelo CIM las asociaciones entre clases se representan mediante una clase que tiene referencias a cada una de las clases involucradas en la asociación. Los valores de estas referencias son el camino CIM a la instancia que referencian. En nuestra representación, las referencias serán URIs a las instancias y serán definidas mediante el elemento owl:ObjectProperty indicando con el elemento rdfs:domain la clase a la que pertenece y con el elemento rdfs:range a la clase que referencia. Por ejemplo, la asociación CIM_MemberOfCollection relaciona la clase CIM_Collection con la clase CIM_ManagedElement. La representación OWL de CIM_MemberOfCollection sin considerar los clasificadores sería la siguiente, donde las propiedades Collection y Member se definen como elementos owl:ObjectProperty.

```
<owl:Class rdf:ID="CIM_MemberOfCollection"/>
<owl:ObjectProperty rdf:ID="Collection">
   <rdfs:domain rdf:resource="#CIM_MemberOfCollection"/>
   <rdfs:range rdf:resource="#CIM_Collection"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Member">
   <rdfs:domain rdf:resource="#CIM_MemberOfCollection"/>
   <rdfs:range rdf:resource="#CIM_ManagedElement"/>
</owl:ObjectProperty>
```

En el siguiente ejemplo podemos ver la representación OWL de la instancia de una asociación.

```
<memberofcollection:CIM_MemberOfCollection
rdf:about="http://ants.um.es/schema_2.12/
CIM_MemberOfCollection.owl/adminfgarcia">
   <memberofcollection:Collection
      rdf:resource="http://ants.um.es/schema_2.12/
      CIM_Role.owl/admin"/>
   <memberofcollection:Member
      rdf:resource="http://ants.um.es/schema_2.12/
      CIM_Identity.owl/fgarcia"/>
</memberofcollection:CIM_MemberOfCollection>
```

Para completar la representación de las asociaciones CIM necesitamos también representar la cardinalidad. Para ello definimos una propiedad inversa a la referencia utilizada en la asociación. Esta propiedad inversa será un propiedad de la clase referenciada por la asociación. Es decir, una clase tendrá una propiedad inversa por cada asociación que la referencie. La cardinalidad de la referencia vendrá determinada por la propia cardinalidad de la propiedad inversa. Por ejemplo, supongamos que una instancia de la clase CIM_Identity no puede tener más de una referencia en el conjunto de

instancias de la asociación CIM_MemberOfCollection.

```
<owl:ObjectProperty rdf:ID="CIM_MemberOfCollection.Collection">
   <rdfs:domain rdf:resource="#CIM_Collection"/>
   <rdfs:range rdf:resource="#CIM_MemberOfCollection"/>
   <owl:inverseOf rdf:resource="#Collection" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="CIM_MemberOfCollection.Member">
   <rdfs:domain rdf:resource="#CIM_ManagedElement"/>
   <rdfs:range rdf:resource="#CIM_MemberOfCollection"/>
   <owl:inverseOf rdf:resource="#Member" />
</owl:ObjectProperty>
<owl:Class rdf:ID="CIM_Identity">
   <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty
       rdf:resource="#CIM_MemberOfCollection.Member"/>
     <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
       1
     </owl:maxCardinality>
   </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Las propiedades CIM_MemberOfCollection.Collection y CIM_MemberOfCollection.Member son propiedades inversas a las propiedades Collection y Member de CIM_MemberOfCollection, respectivamente. CIM_MemberOfCollection.Collection es una propiedad de CIM_Collection y CIM_MemberOfCollection.Member es una propiedad de CIM_ManagedElement. La cardinalidad de la asociación entre CIM_Identity (subclase de CIM_ManagedElement) y CIM_MemberOfCollection queda determinada por la cardinalidad de la propiedad inversa CIM_MemberOfCollection.Member que en este caso es 1.

Utilizando esta representación la integridad y la cardinalidad son garantizadas por un razonador OWL que compruebe la consistencia de las definiciones.

Clasificadores

En esta propuesta son tenidos en cuenta los meta-clasificadores y los clasificadores estándar. Además, nuestra representación se centra sólo en los clasificadores usados en clases, asociaciones, propiedades y referencias, excluyendo a los clasificadores de los conceptos no considerados (métodos, parámetros e indicaciones).

Las principales reglas para esta representación son las siguientes:

- Si un clasificador tiene una representación equivalente en OWL, ésta es usada; éste es el caso de los clasificadores Deprecated (indica que el elemento es tolerado pero no recomendado y podría ser sustituido por otro) o Description (proporciona una descripción del elemento), por ejemplo.

- Si un clasificador de clase de tipo Boleano no tiene una representación equivalente en OWL, entonces proponemos una nueva clase para representar este clasificador, como ocurre por ejemplo con los clasificadores Indication o Abstract.

- Si un clasificador de propiedad de tipo Boleano no tiene una representación equivalente en OWL, entonces proponemos una nueva clase que representa una nueva propiedad que modela el clasificador. Por ejemplo, éste es el caso del clasificador ArrayType (indica el tipo de una array).

- Si un clasificador de clase o propiedad de tipo cadena o array de cadenas no tiene una representación equivalente en OWL, entonces declaramos una propiedad de anotación (AnnotationProperty) para modelar el clasificador. Éste es el caso de Source (indica la localización de una instancia) o NullValue (indica para que valor la propiedad es NULL), por ejemplo.

Una tabla resumen de la representación de todos los clasificadores considerados se encuentra en el Anexo B. Ahora detallamos la representación de algunos de los clasificadores.

- *Ocurrencia de las propiedades.* La ocurrencia de las propiedades clave, las propiedades requeridas y las propiedades definidas como un array de un tipo de dato básico se representan mediante restricciones de cardinalidad en las clases para las propiedades utilizando los elementos restriction, maxCardinality y minCardinality.

- *Descripción.* Una descripción CIM tanto de una clase como de una propiedad es representada usando el elemento rdfs:comment.

- *Clase abstracta.* No existe una correspondencia directa entre el clasificador Abstract y un elemento o constructor en OWL. Para representar que una clase es abstracta, usamos la subclase QrAbstract, de manera que cualquier clase que tenga como clase padre QrAbstract es una clase abstracta.

- *Valor por defecto.* Aquí tampoco existe una correspondencia directa y usamos una propiedad de anotación para representar el valor por defecto.

3.3. Representación lógica de reglas de políticas (SWRL-CIM)

La ontología CIM-OWL contiene una secuencia de hechos y axiomas siguiendo las pautas de representación establecidas anteriormente pero no dispone de la posibilidad de definir reglas Horn. Si bien CIM dispone de la subdivisión Policy que podría servir para definición de reglas de políticas, estas definiciones no pueden ser integradas de manera directa en un razonador de reglas. En cambio nuestra solución SWRL-CIM propone extender CIM-OWL con axiomas de reglas mediante SWRL lo que nos permitirá definir políticas en un formato que facilita la traducción de reglas a o desde sistemas de reglas existentes o futuros, como por ejemplo Prolog.

En SWRL un axioma de regla consiste en un antecedente (cuerpo) y un consecuente (cabeza), donde cada uno de los cuales está compuesto por un conjunto de átomos. De tal manera que la sintaxis de regla SWRL permite la definición de reglas if-then, donde si se cumplen un conjunto de condiciones (antecedente) entonces se llevan a cabo un conjunto de acciones (consecuente). En SWRL-CIM proponemos una restricción útil en la forma de las reglas que consiste en limitar el uso de átomos de clase a clases nombradas definidas directamente en CIM-OWL. Esta restricción determina un formato de las reglas que facilita en gran medida la traducción de las reglas en sistemas donde poder aplicar técnicas de razonamiento que es una de nuestras principales herramientas de cara a afrontar los procesos de análisis de conflictos.

Para mostrar como se interpreta una regla SWRL-CIM, presentamos un pequeño ejemplo. Este ejemplo utiliza la clase CIM_SystemDevice del esquema CIM, que posee dos instancias asociadas a dos puertos, y activa uno si el otro no funciona, siguiendo la regla:

IF LogicalPort #1 está desactivado, THEN Activar LogicalPort #2

En SWRL-CIM:

$$CIM\_LogicalPort(LP1?) \wedge DeviceID(LP1?,'Lport1') \wedge CIM\_LogicalPort(LP2?) \wedge DeviceID(LP2?,'Lport2') \wedge StatusInfo(LP1?,4) \Rightarrow StatusInfo(LP2?,3)$$

Esta regla se aplica sobre unas determinadas instancias de la clase CIM_LogicalPort, que representa conceptualmente a un puerto o punto de conexión de un dispositivo, la instancia con la propiedad DeviceID con valor Lport1 y la instancia con el valor Lport2. Si la primera instancia tiene la propiedad StatusInfo con valor 4 (valor entero asignado por el modelo de información CIM al estado desactivado), entonces la segunda instancia para a tener un valor 3 (valor entero asignado al estado activado) en su propiedad StatusInfo.

Como puede verse en el ejemplo, cuando la regla tiene un hecho (o conjunto de hechos) en el antecedente que se cumplen, el razonador de reglas generará el nuevo hecho (o conjunto de hechos) definidos en el consecuente. Este hecho deberá ser añadido a la ontología y, a su vez, podrá provocar que otra regla se dispare y pueda provocar un nuevo cambio en la ontología.

SWRL-CIM requiere que el administrador encargado de definir el comportamiento de los elementos gestionados siga un proceso en la especificación de las políticas (véase figura 3.3). En primer lugar el administrador debe describir el sistema y sus capacidades de seguridad usando la ontología CIM-OWL; sólo entonces, el administrador puede usar estos elementos dentro de las reglas SWRL que representan las políticas de seguridad que controlan el comportamiento del sistema gestionado.

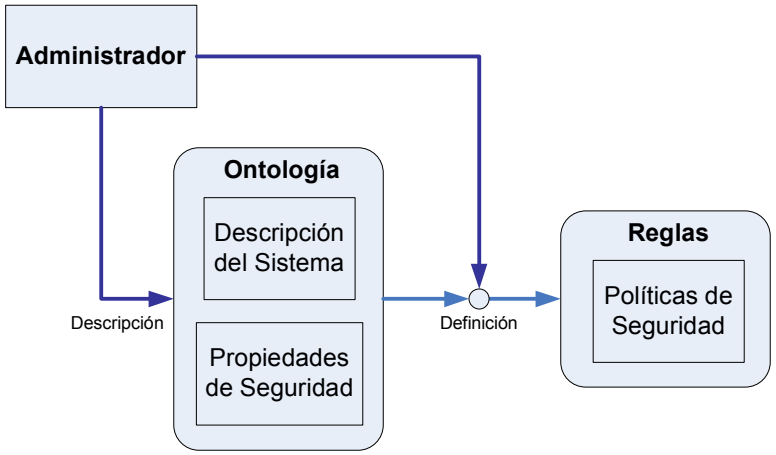

Figura 3.3: *Especificación de políticas de seguridad*

Dado que CIM permite representar conceptos de alto nivel y bajo nivel (por ejemplo, datos de nivel de aplicación y puntos finales de nivel de red), podemos usar CIM-OWL junto con SWRL para representar políticas de alto nivel y bajo nivel (por ejemplo, 'todas las entidades deben autenticarse mediante certificados digitales' y 'bloquear las conexiones al puerto 53 del host X'). Ciertamente, el esquema CIM da a los administradores de políticas

la suficiente flexibilidad para elegir el nivel apropiado para sus necesidades. El esquema CIM y sus subdivisiones son presentados en las sección 3.3.1.

La flexibilidad que proporciona el uso del esquema CIM, permite que SWRL-CIM pueda definir diferentes tipos de políticas de seguridad. En nuestra propuesta nos hemos centrado en los siguientes:

- Políticas de autenticación basadas en credenciales.
- Políticas de control de acceso que incluyen políticas de autorización y políticas de delegación.
- Políticas de obligación basadas en eventos implícitos.
- Políticas de red que incluyen políticas de filtrado y políticas de seguridad IP.

En las siguientes apartados presentamos, además de nuestra propuesta para estos tipos de políticas, como hemos extendido el esquema CIM para introducir restricciones de tiempo en las políticas y la clasificación de políticas mediante niveles de seguridad.

Es importante subrayar que nuestra propuesta no es la única manera de plantear la combinación de CIM y SWRL. En [43] encontramos un trabajo de investigación en la misma línea que nuestra propuesta definiendo el comportamiento de gestión de la red con reglas SWRL utilizando una ontología OWL basada en CIM. Aunque esta propuesta difiere de manera importante de nuestra propuesta. En primer lugar porque utiliza una ontología OWL-Full mientras nuestra propuesta utiliza una ontología OWL-DL, lo cual limita de manera importante sus capacidades para aplicar técnicas de razonamiento. Y además define las acciones de la política sólo como llamadas a un servicio externo a la ontología mientras en nuestra propuesta las acciones implican la generación de nueva información que se incluye en la ontología. Esta forma de definir las acciones no facilita la aplicación de técnicas de razonamiento sobre las reglas que permita la detección de conflictos.

3.3.1. Esquema CIM

El esquema CIM define el conjunto de conceptos CIM y debe ser bien conocido para hacer un uso correcto de los conceptos al describir el dominio. CIM está estructurado en tres capas: modelo núcleo, modelo común y esquemas de extensión. Los dos primeros, modelo núcleo y modelo común, forman el denominado esquema CIM. Este esquema ha llegado a ser tan

extenso que para facilitar su uso ha sido segmentado en catorce subdivisiones en su última versión v2.12. A continuación vamos a describir brevemente cada una de las subdivisiones.

- **Aplication.** Esta subdivisión incluye clases para representar elementos, productos y sistemas software también como los elementos requeridos para gestionar el desarrollo de software, incluyendo características software, información sobre drivers, información BIOS y elementos instalados.

- **Core.** Esta subdivisión incluye todos los conceptos básicos y aplicables al resto de subdivisiones del esquema. Incluye el concepto de elemento gestionable, sistema gestionable, colección, localización, etc.

- **Database.** Proporciona un conjunto de clases y asociaciones para gestionar sistemas de bases de datos, incluyendo el sistema de base de datos en sí mismo, como es desarrollado, sus aspectos de ejecución, las entidades relevantes para almacenamiento de información en la base de datos, y la estadísticas usadas para gestionar una base de datos.

- **Device.** Esta subdivisión extiende los conceptos de gestión relativos a un dispositivo lógico, incluyendo dispositivos de almacenamiento, dispositivos USB, controladores, sensores, puertos y adaptadores de red, dispositivos de usuario (por ejemplo monitores, teclados, etc.), memoria, impresoras y procesos de impresión, conexión entre dispositivos y estadísticas de error.

- **Event.** Describe los tipos de notificaciones (relativas a clases CIM, instancias de estas clases CIM, llamadas a métodos, etc.) y las clases para suscribirse a eventos publicados por proveedores e instrumentación, y para describir el tratamiento de estos eventos.

- **Interop.** Este conjunto describe las características de gestión y los componentes de un servidor WBEM [19].

- **IpsecPolicy.** Define las clases necesarias para negociar una asociación de seguridad IPsec, incluyendo propuestas IPsec y IKE, acciones y reglas.

- **Metrics.** Esta subdivision describe elementos genéricos relativos a la métrica. Para hacer este conjunto lo más genérico posible, su especificación permite expresiones de métrica definidas externamente de objetos existentes y la definición de métrica en tiempo de ejecución.

- **Network.** Define conceptos de red específicos y genéricos, incluyendo sistemas autónomos, servicios de red, interfaces de protocolos, conmutación, servicios de enrutamiento y redirección, colecciones (es decir, redes lógicas y rangos de direcciones IP) y filtrado. También incluye algunas clases para gestionar diferentes protocolos. Estos son MPLS, OSPF, BGP, DiffServ/IntServ QoS, IPsec y SNMP.

- **Physical.** Esta subdivisión define conceptos relativos a componentes hardware, incluyendo chips, tarjetas, medios de almacenamiento, conectores y enlaces físicos.

- **Policy.** Proporciona un framework para especificar información asociada a políticas de configuración y operacionales usando reglas compuestas de condiciones y acciones.

- **Support.** Define un modelo de objetos para el intercambio de conocimiento y un modelo de transacción para este conocimiento. Éste incluye la definición de problemas, soluciones, datos de diagnóstico, persona de contacto a cargo del soporte, acuerdos relativos al servicio de soporte, etc.

- **System.** Esta subdivisión extiende los conceptos que son relativos a un sistema, y especialmente a un sistema de computación; incluye procesos software, hilos, trabajos, servicios de ayuda, sistemas operativos, sistemas de ficheros, sistemas virtuales y clusters.

- **User.** Esta subdivisión extiende los conceptos de gestión relativos a usuarios y seguridad, incluyendo organizaciones, personas, grupos, roles, credenciales, identidades, niveles de seguridad, requerimientos para la autenticación, privilegios, y control de acceso.

Todas las subdivisiones son susceptibles de ser usadas en la especificación de políticas de seguridad, si bien los principales conceptos podemos encontrarlos en las subdivisiones User, System, Network, Device y Core. Será a partir de ellas que vayamos a definir las especificaciones de políticas que se presentan a continuación.

3.3.2. Políticas de autenticación

Una política de autenticación representa los requerimientos de autenticación para una entidad en el dominio de gestión. Mediante SWRL-CIM definimos políticas de autenticación basadas en credenciales e

identidades, de manera que una credencial determinará la identidad de una entidad dentro de la ontología.

Las clases del esquema CIM involucradas en este tipo de políticas se encuentran en la subdivisión User. La figura 3.4 muestra las principales clases y asociaciones que presentamos brevemente:

- CIM_Identity. Una instancia de la clase CIM_Identity representa a una identidad. Tiene la propiedad CurrentlyAuthenticated para indicar si la identidad está actualmente autenticada o no. La clase CIM_IPNetworkIdentity es una especialización de CIM_Identity usada para representar identidades de red que podrían usarse en un servicio de red (éste es representado mediante la clase CIM_IPProtocolEndPoint).

- CIM_Credential. Esta clase define materiales, información u otros datos que son usados para establecer la identidad. Generalmente, habrá alguna información compartida o material de credencial que es usado para identificar y autenticar una entidad en el proceso de conseguir acceso a, o permiso para usar, recursos. Tal material de credencial puede ser usado para autenticar la identidad de una entidad inicialmente.

- CIM_AssignedIdentity. Esta asociación relaciona una identidad a un elemento de gestión específico. En nuestro caso, a una credencial.

- CIM_IdentityContext. Esta asociación define el contexto (por ejemplo, un sistema o un servicio) de una identidad. Del mismo modo que un contexto puede disponer de varias identidades, una identidad puede pertenecer a varios contextos.

Utilizando estas clases y asociaciones podemos definir políticas de autenticación basadas en credenciales. Así, la política describe los requerimientos bajo los cuales una credencial es asociada a una identidad y ésta es autenticada.

Existen cinco tipos de credenciales en el esquema CIM que nos permiten establecer diferentes tipos de políticas de autenticación. Mediante CIM_SharedSecret definimos políticas de autenticación basadas en secretos compartidos, con CIM_KerbertosTicket definimos políticas de autenticación Kerberos, con CIM_NamedSharedIKESecret establecemos políticas de autenticación IKE, con CIM_PublicKeyCertificate establecemos políticas de autenticación basadas en certificados digitales y con CIM_UnsignedPublicKey políticas de autenticación basadas en clave pública.

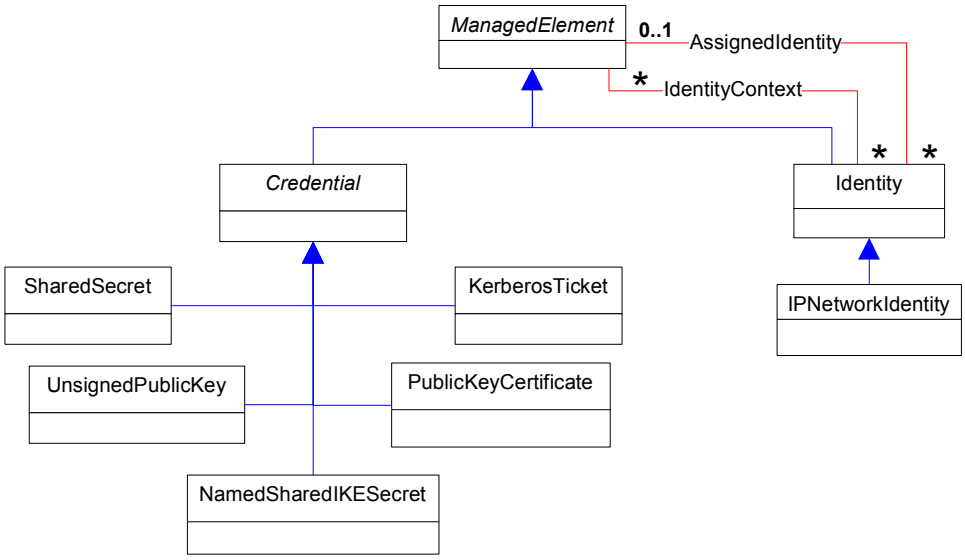

Figura 3.4: *Clases CIM usadas en las políticas de autenticación*

Para mostrar como se especifica una política de autenticación con SWRL-CIM, presentamos un ejemplo. Este ejemplo utiliza las clases CIM_SharedSecret y CIM_Identity para especificar que una instancia de CIM_Identity es autenticada cuando existe una instancia de CIM_SharedSecred con ciertos valores; el ejemplo se puede concretar de la siguiente forma:

IF SharedSecret #SS1 con identificador 'fgarcia' y
secreto 'abcde' usando HMAC-MD5,
THEN Identity #ID1 con identificador 'fgarcia' es autenticada

En SWRL-CIM:

$$CIM\_SharedSecret(SS1?) \wedge RemoteID(SS1?,"fgarcia") \wedge Secret(SS1?,"abcde") \wedge Algorithm(SS1?,"HMAC-MD5") \wedge CIM\_Identity(ID1?) \wedge InstanceID(ID1?,"fgarcia") \Rightarrow CIM\_AssignedIdentity(AI1?) \wedge IdentityInfo(AI1?,ID1) \wedge ManagedElement(AI1?,SS1?) \wedge CurrentlyAuthenticated(ID1?,TRUE)$$

Con lo que cuando un usuario con login 'fgarcia' presente frente al sistema de autenticación un reto 'abcde' obtenido mediante HMAC-MD5 quedará autenticado frente al sistema.

3.3.3. Políticas de control de acceso

Las reglas de las políticas de control de acceso son definidas para especificar bajo que condiciones un individuo puede acceder a ciertos recursos, información u otros elementos del sistema. Presentamos dos tipos de políticas de control de acceso: políticas de autorización y políticas de delegación.

Políticas de autorización

Una política de autorización representa los derechos y las prohibiciones de acceso a un recurso por parte de una identidad del dominio de gestión. Mediante SWRL-CIM definimos políticas de autorización basadas en privilegios y roles.

Las clases del esquema CIM involucradas en este tipo de políticas se encuentran en la subdivisión User. La figura 3.5 muestra las principales clases y asociaciones que presentamos brevemente:

- CIM_Privilege. La clase CIM_Privilege es la base para todas las actividades que son permitidas o denegadas a una identidad o un role. Tiene la propiedad PrivilegeGranted para indicar si el privilegio es concedido o denegado. La subclase CIM_AuthorizedPrivilege es la especialización para los privilegios de autorización.

- CIM_Role. La clase CIM_Role es usada para representar un role, es decir una posición o un conjunto de responsabilidades dentro de una organización o unidad organizativa. Las identidades que desempeñan un role son explícitamente asociadas mediante la asociación CIM_MemberOfCollection.

- CIM_AuthorizedSubject. Esta asociación relaciona un sujeto (role o identidad) a un privilegio de autorización.

- CIM_AuthorizedTarget. Esta asociación define el objetivo (por ejemplo, un sistema o un servicio) del privilegio de autorización.

Utilizando estas clases y asociaciones podemos definir políticas de autorización basadas en privilegios y roles. Así, la política describe los requerimientos bajo los cuales una identidad o role es asociado a un privilegio y éste es permitido.

La políticas de autorización y autenticación tienen un punto común en las instancias de la clase CIM_Identity y concretamente en la propiedad CurrentlyAuthenticated. Así cuando una identidad no está autenticada (CurrentlyAuthenticated = FALSE), entonces los privilegios asociados no deben ser autorizados.

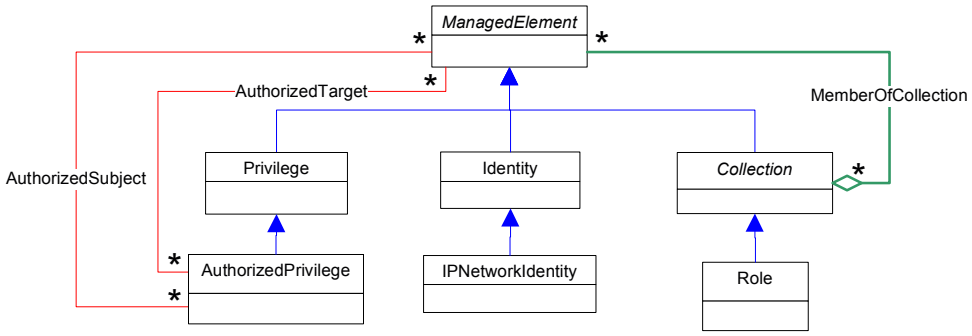

Figura 3.5: *Clases CIM usadas en las políticas de autorización*

Podemos hablar de diferentes tipos de políticas de autorización. Mediante el uso de roles definimos políticas de control de acceso basadas en role (RBAC); con el uso de identidades (sin roles) estamos definiendo políticas que van explícitamente ligadas a las identidades; también existe la posibilidad de ligar la política a otro tipo de colecciones como son los grupos. Por otro lado, los objetivos de las políticas también nos permiten hablar de políticas de autorización para servicios, sistemas o redes.

A continuación presentamos un ejemplo para mostrar como se especifica una política de autorización usando SWRL-CIM. Este ejemplo utiliza las clases CIM_ComputerSystem, CIM_Role y CIM_AuthorizedPrivilege para especificar que una instancia de CIM_Role es autorizada a realizar una actividad sobre una instancia de CIM_ComputerSystem, la regla se puede definir a alto nivel como sigue:

IF ComputerSystem #CS1 tiene un privilegio de ejecución,
THEN Role #R1 dispone de este privilegio

En SWRL-CIM:

$$CIM\_ComputerSystem(CS1?) \wedge Name(CS1?, "Firewall") \wedge$$
$$CIM\_AuthorizedPrivilege(AP1?) \wedge InstanceID(AP1?, "positif : grantExecute") \wedge$$
$$CIM\_Role(R1?) \wedge Name(R1?, "admin") \wedge$$
$$CIM\_AuthorizedTarget(AT1?) \wedge Privilege(AT1?, AP1?) \wedge$$
$$TargetElement(AT1?, CS1?)$$
$$\Rightarrow$$
$$CIM\_AuthorizedSubject(AS1?) \wedge Privilege(AS1?, AP1?) \wedge$$
$$PrivilegedElement(AS1?, R1?)$$

En esta representación en SWRL-CIM tenemos que si el sistema Firewall tiene el privilegio *positif:grantExecute*, el role Administrador dispone de este privilegio.

Políticas de delegación

La delegación se utiliza con frecuencia en los sistemas de control de acceso para transferir derechos o prohibiciones de acceso entre identidades. Mediante SWRL-CIM definimos políticas de delegación mediante la propagación de privilegios entre objetivos (es decir, entre servicios o sistemas) o sujetos (es decir, identidades o roles).

Las clases y asociaciones del esquema CIM involucradas en este tipo de políticas son las mismas que las que aparecen en las políticas de autorización en la figura 3.5.

A continuación presentamos un ejemplo para mostrar como se especifica una política de delegación usando SWRL-CIM. Este ejemplo utiliza las clases CIM_Role y CIM_AuthorizedPrivilege para especificar que el privilegio de realizar una actividad se propaga entre dos instancias de CIM_Role, siguiendo la regla:

IF Role #R1 dispone de un privilegio,
THEN Role #R2 dispone también de ese privilegio

En SWRL-CIM:

$$CIM\_AuthorizedPrivilege(AP1?) \wedge CIM_{R}ole(R1?) \wedge Name(R1?, "admin") \wedge CIM\_AuthorizedSubject(AS1?) \wedge Privilege(AS1?, AP1?) \wedge PrivilegedElement(AS1?, R1?) \wedge CIM\_Role(R2?) \wedge Name(R2?, "manager") \Rightarrow CIM\_AuthorizedSubject(AS2?) \wedge Privilege(AS2?, AP1?) \wedge PrivilegedElement(AS2?, R2?)$$

Con lo que si el role Administrador dispone de un privilegio, el role Manager también dispondrá de él.

3.3.4. Políticas de obligación

Las políticas de obligación especifican las acciones que deben ser realizadas cuando se producen ciertos cambios en el dominio de gestión. Así las políticas de seguridad especifican que acciones deben ser especificadas cuando ciertas violaciones de seguridad ocurren (por ejemplo, una máquina cae o una puerta es abierta) y que actividades auditoras e informativas deben ser realizadas (por ejemplo, activar una alarma o registrar la información del evento).

Cuando una violación de seguridad ocurre puede detectarse mediante un evento implícito donde los cambios de estado determinan el evento en el dominio de gestión. SWRL-CIM se basa en eventos implícitos ya que los cambios de estado de los elementos de la ontología son los que provocan que una regla se dispare o no.

En cuanto a las clases del esquema CIM involucradas en la acciones auditoras e informativas se encuentran en las subdivisiones Device y System. La figura 3.6 muestra las principales clases y asociaciones que presentamos brevemente:

- CIM_MessageLog. La clase CIM_Log representa cualquier registro de evento, error o informativo. Su subclase CIM_MessageLog permite mediante la asociación CIM_RecordInLog la agregación de registros representados mediante CIM_LogRecord. Esta clase dispone de la propiedad RecordData donde se almacena la información relativa al registro.

- CIM_AlarmDevice. Una instancia de CIM_AlarmDevice representa un tipo de dispositivo que emite indicaciones audibles o visibles relativas a una situación de alarma.

Para la definición de políticas de obligación se ven involucradas todas las subdivisiones del esquema CIM. A continuación presentamos dos ejemplos para mostrar como se especifican política de obligación usando SWRL-CIM y clases de diferentes subdivisiones del esquema CIM.

El siguiente ejemplo utiliza las clases CIM_ComputerSystem y CIM_LogRecord para especificar que cuando una instancia de CIM_ComputerSystem tiene un estado de caído, un registro informativo es almacenado, siguiendo la regla:

IF ComputerSystem #CS1 está caído,
THEN un registro informativo es insertado en MessageLog #ML1

En SWRL-CIM:

$$CIM\_ComputerSystem(CS1?) \wedge Dedicated(CS1?, 11) \wedge Name(CS1?, "Impresora1") \wedge OperationalStatus(CS1?, 6) \wedge CIM\_MessageLog(ML1?) \wedge Name(ML1?, "WarningLogs") \Rightarrow CIM\_LogRecord(LR1?) \wedge RecordData(LR1?, "Impresora1-ko") \wedge RecordInLog(RIL1?) \wedge MessageLog(RIL1?, ML1?) \wedge LogRecord(RIL1?, LR1?)$$

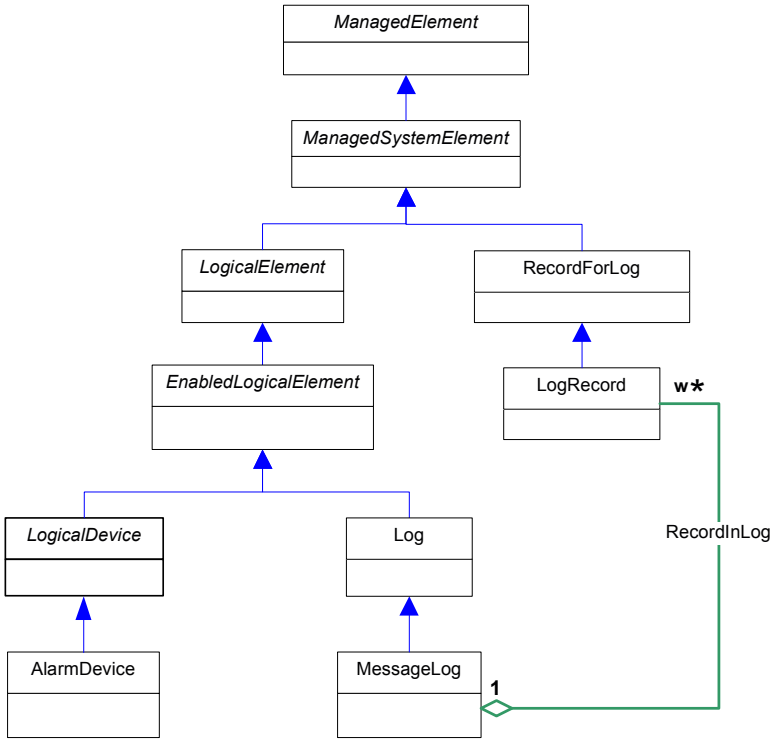

Figura 3.6: *Clases CIM usadas en las políticas de obligación*

En este ejemplo, las clases utilizadas son de la subdivisión System. También podemos definir políticas de obligación para la subdivisión Device. El siguiente ejemplo utiliza las clases CIM_Door y CIM_AlarmDevice para especificar que si una puerta está abierta, entonces una alarma es activada, siguiendo la regla:

IF Door #D1 está abierta, THEN AlarmDevice #AD1 es activada

En SWRL-CIM:

$$CIM\_Door(D1?) \wedge DeviceID(D1?, "Puerta1") \wedge$$
$$Open(D1?, TRUE) \wedge$$
$$CIM\_AlarmDevice(AD1?) \wedge DeviceID(AD1?, "Alarma1")$$
$$\Rightarrow$$
$$AlarmState(AD1?, 3)$$

En esta representación en SWRL-CIM tenemos que si la puerta Puerta1 está abierta, la alarma Alarma1 se activa.

3.3.5. Políticas de red

Entre estas políticas encontramos políticas de filtrado y políticas de seguridad IP. Ambos tipos de políticas se caracterizan por disponer de un conjunto de clases y asociaciones en el esquema CIM que permiten ser definidas de manera exhaustiva. Este hecho permite que podemos disponer de reglas de filtrado o reglas de seguridad IP predefinidas en la ontología de manera que la política sólo deba asociarlas a los elementos del sistema que deban aplicarlas.

Políticas de filtrado

El esquema CIM dispone de un conjunto de clases para la representación de filtros de todo tipo. Las políticas de filtrado que SWRL-CIM consisten en determinar cuando estos filtros son o no son asociados a los sistemas del dominio.

En cuanto a las clases del esquema CIM involucradas en la definición de filtros se encuentran en la subdivisión Network. La figura 3.7 muestra las principales clases y asociaciones que presentamos brevemente:

- CIM_FilterEntryBase. La clase abstracta CIM_FilterEntryBase representa el concepto de filtros. Sus subclases definen diferentes tipos de filtros donde destacan los filtros basados en los campos de la cabecera IP.

- CIM_FilterList. Una instancia de CIM_FilterList agrega instancias de CIM_FilterEntryBase (o de sus susbclases) mediante la asociación CIM_EntriesInFilterList.

- CIM_HostedFilterList y CIM_HostedFilterEntryBase. Estas asociaciones permiten asociar a los filtros bien mediante CIM_FilerList o bien directamente al sistema que debe aplicarlos.

Para conseguir una especificación del esquema adecuado a nuestras necesidades, hemos modificado el esquema quitando la restricción de cardinalidad que no permite asociar un filtro o lista de filtros a más de una instancia de ComputerSystem. Así las asociaciones CIM_HostedFilterList y CIM_HostedFilterEntryBase son asociaciones muchos-a-muchos. Véase el Anexo C para ver la nueva descripción en formato MOF de estas asociaciones.

A continuación presentamos un ejemplo para mostrar como se especifica una política de filtrado usando SWRL-CIM. Este ejemplo utiliza las clases CIM_ComputerSystem y CIM_FilterList para especificar que una lista de

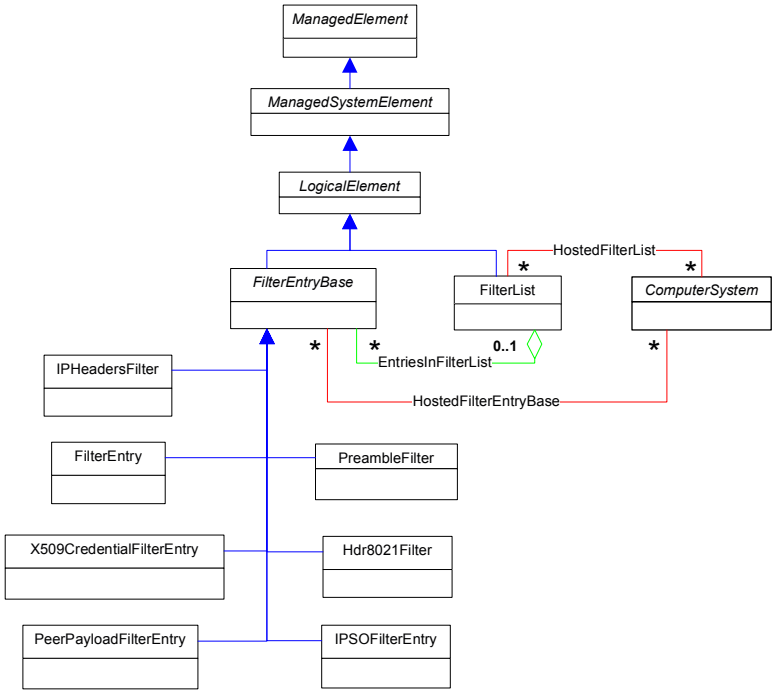

Figura 3.7: *Clases CIM usadas en las políticas de filtrado*

filtros es asociada a una determinada instancia de CIM_ComputerSystem, siguiendo la regla:

IF ComputerSystem es un firewall,
THEN FilterList #FL1 se asocia al firewall

En SWRL-CIM:

$$CIM\_ComputerSystem(CS1?) \wedge Dedicated(CS1?, 10) \wedge$$
$$CIM\_FilterList(FL1?) \wedge Name(FL1?, "filtrosparafirewalls")$$
$$\Rightarrow$$
$$CIM\_HostedFilterList(HFL?) \wedge$$
$$HostedFilterList(HFL1?, CS1?) \wedge$$
$$Dependent(HFL1?, FL1?)$$

En esta representación en SWRL-CIM tenemos que si un sistema está dedicado a funciones de firewall, entonces se le asocia el listado de filtros filtrosparafirewalls.

Políticas de seguridad IP

El esquema CIM dispone de un conjunto de clases y asociaciones agrupadas en la subdivisión IpsecPolicy que representan todos los conceptos necesarios para representar las políticas de seguridad IP de un elemento de red. Las políticas de seguridad IP en SWRL-CIM consisten en determinar cuando estas configuraciones son o no son asociados a los sistemas del dominio.

La figura 3.8 muestra las principales clases y asociaciones que presentamos brevemente:

- CIM_SARule. La clase SARule es la clase base para definir las reglas IKE e IPsec. CIM_IKERule contiene las condiciones y acciones para la fase 1 de negociación y CIM_IpsecRule contiene las relativas a la fase 2.
- CIM_PolicySetAppliesToElement. Esta asociación representa explícitamente que regla es aplicada a un elemento particular de la ontología.

A continuación presentamos un ejemplo para mostrar como se especifica una política de seguridad IP usando SWRL-CIM. Este ejemplo utiliza las clases CIM_ComputerSystem, CIM_IKERule y CIM_IPsecRule para especificar que una configuración IPsec es asociada a una determinada instancia de CIM_ComputerSystem, siguiendo la regla:

IF ComputerSystem es un gateway,
THEN IPsecRule #FL1 y IKERule se asocian al gateway

En SWRL-CIM:

$$
\begin{gathered}
CIM\_ComputerSystem(CS1?) \wedge Dedicated(CS1?, 20) \wedge \\
CIM\_IPsecRule(IPR1?) \wedge \\
PolicyRuleName(IPR1?, "reglaipsec1") \wedge \\
CIM\_IKERule(IKER1?) \wedge \\
PolicyRuleName(IKER1?, "reglaike1") \\
\Rightarrow \\
CIM\_PolicySetAppliesToElement(PSATE1?) \wedge \\
ManagedElement(PSATE?, CS1?) \wedge \\
PolicySet(PSATE?, IPR1?) \wedge \\
CIM\_PolicySetAppliesToElement(PSATE2?) \wedge \\
ManagedElement(PSATE2?, CS1?) \wedge \\
PolicySet(PSATE2?, IKER1?)
\end{gathered}
$$

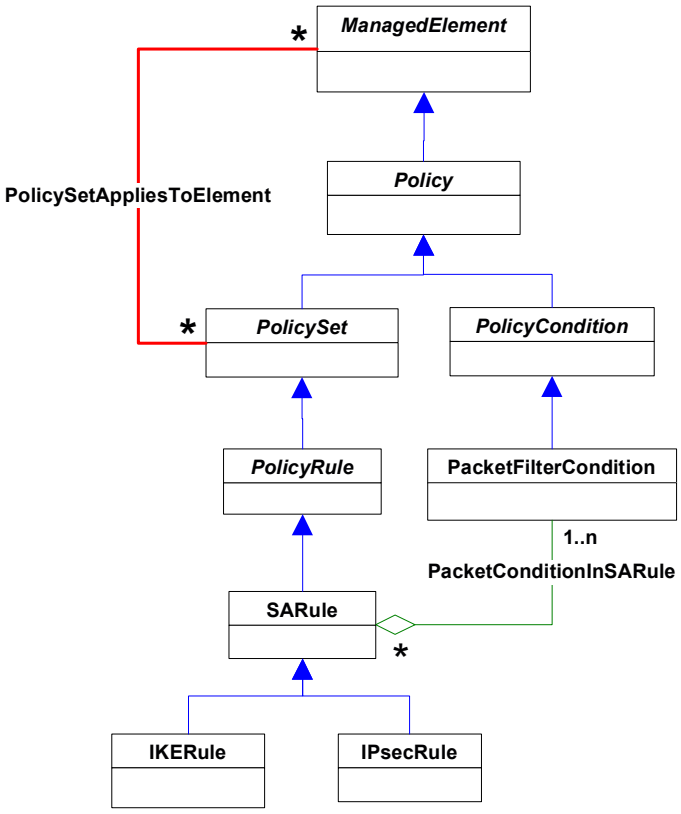

Figura 3.8: *Clases CIM usadas en las políticas de seguridad IP*

En esta representación en SWRL-CIM tenemos que si un sistema está dedicado a funciones de gateway, entonces se le asocia la configuración IPsec determinada por las instancias reglaipsec1 y reglaike1.

3.3.6. Restricciones de tiempo en políticas

La restricciones de tiempo especifican los periodos de validez de una política. En nuestra propuesta extendemos el esquema CIM para incluir un dispositivo que nos permita conocer la hora actual y, por tanto, determinar periodos de tiempo que pueden ser utilizados en la especificación de las políticas.

La figura 3.9 muestra la clase CIM_ClockDevice. Esta clase representa a un reloj que nos permite dispone de una propiedad DateTime de solo lectura con la fecha y hora actuales. Véase el Anexo C para una descripción detallada de esta nueva clase.

Para mostrar como se especifica una política con una restricción de tiempo con SWRL-CIM, presentamos un ejemplo con una política de obligación. Este

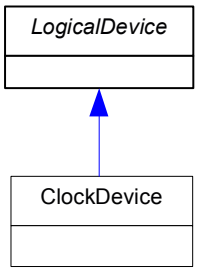

Figura 3.9: *Clases CIM usadas para la restricciones de tiempo*

ejemplo utiliza las clases CIM_Door y CIM_AlarmDevice para especificar que si una puerta está abierta entre unas determinadas horas del día, entonces una alarma es activada, siguiendo la regla:

IF Hoy es Domingo AND Door #D1 está abierta,
THEN AlarmDevice #AD1 es activada

En SWRL-CIM:

$$CIM\_ClockDevice(CD1?) \wedge DeviceID(CD1?, "Reloj1") \wedge$$
$$Day(CD1?, "Domingo") \wedge CIM\_Door(D1?) \wedge$$
$$DeviceID(D1?, "Puerta1") \wedge Open(D1?, TRUE) \wedge$$
$$CIM\_AlarmDevice(AD1?) \wedge DeviceID(AD1?, "Alarma1")$$
$$\Rightarrow$$
$$AlarmState(AD1?, 3)$$

En esta representación en SWRL-CIM tenemos que si el día actual es Domingo y la puerta Puerta1 está abierta, entonces la alarma Alarma1 es activada.

3.3.7. Niveles de seguridad en las políticas

Dentro de un contexto de seguridad, una política puede ser impartida en un nivel de confianza o seguridad. Los niveles de seguridad son definidos por el administrador en el dominio de gestión y las políticas pueden contemplarlo en su definición.

El esquema CIM dispone de la clase CIM_SecuritySensitivity y la asociación CIM_ElementSecuritySensitivity para representar los niveles de seguridad. CIM_SecuritySensitivity define un nivel de seguridad que puede ser asociado a un elemento del dominio. Un ejemplo simple es definir niveles de seguridad clasificándoles como 'bajo', 'moderado' y 'alto'. Un elemento es asignado al nivel de seguridad mediante la asociación

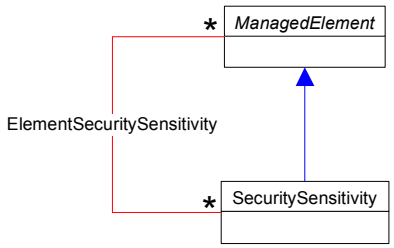

Figura 3.10: *Clases CIM usadas para representar el nivel de seguridad*

CIM_ElementSecuritySensitivity. El nivel puede ser asociado a cualquier elemento, tales como Identidades, Roles, Sistemas y Servicios.

Para mostrar como se especifica en una política el nivel de seguridad con SWRL-CIM, presentamos un ejemplo con una política de filtrado. Este ejemplo utiliza las clases CIM_ComputerSystem y CIM_FilterList para especificar que una lista de filtros es asociada a las instancias de CIM_ComputerSystem con un nivel de seguridad alto, siguiendo la regla:

IF ComputerSystem tiene un nivel de seguridad alto,
THEN FilterList #FL1 se asocia al ComputerSystem

En SWRL-CIM:

$$CIM\_SecuritySensitivity(SS1?) \wedge InstanceID(SS1?, "positif : red") \wedge$$
$$CIM\_ComputerSystem(CS1?) \wedge Dedicated(CS1?, 10) \wedge$$
$$CIM\_ElementSecuritySensitivity(ESS1?) \wedge$$
$$SecurityLevel(ESS1?, SS1?) \wedge$$
$$ManagedElement(ESS1?, CS1?) \wedge CIM\_FilterList(FL1?) \wedge$$
$$Name(FL1?, "filtrosRed")$$
$$\Rightarrow$$
$$CIM\_HostedFilterList(HFL?) \wedge$$
$$HostedFilterList(HFL1?, CS1?) \wedge$$
$$Dependent(HFL1?, FL1?)$$

En esta representación en SWRL-CIM tenemos que si un sistema está dedicado a funciones de firewall y tiene nivel de seguridad *positif:red*, entonces se le asocia el listado de filtros filtrosRed.

3.4. Razonamiento

El hecho de que SWRL-CIM surja de la combinación de la ontología CIM-OWL y el lenguaje SWRL ofrece una clara ventaja: esto permite dos

tipos diferentes de razonamiento automático (ver figura 3.11). El primero es el razonamiento sobre la ontología (es decir, el razonamiento sobre la estructura y las instancias de la ontología) y el segundo es el razonamiento basado en reglas (es decir, aplicar razonamiento sobre reglas de políticas de las que se hace uso en las tareas de gestión). Por tanto, identificamos la necesidad de usar un razonador OWL y una razonador basado en reglas para cada tipo de razonamiento. Utilizamos el término razonador para referirnos a un programa específico que realiza las tareas de inferencia (es decir, proceso de derivar información adicional que no ha sido explícitamente especificada).

En la fase de especificación de las políticas, el razonador de reglas es usado para obtener conocimiento adicional que no ha sido explícitamente especificado durante la definición del sistema. Este razonador proporciona un razonamiento encadenado hacia delante y hacia atrás que permite inferir nuevo conocimiento a partir del conjunto de reglas SWRL. De esta manera el razonador de reglas extrae información que no ha sido explícitamente especificada, lo que nos permitirá aumentar nuestro conocimiento sobre las propiedades de seguridad del sistema. Por su parte, el razonador OWL es usado para realizar tareas de inferencia sobre la ontología CIM-OWL. Este razonador permitir validar la base de conocimiento y también puede ser utilizado para realizar consultas sobre la descripción y las propiedades de seguridad del sistema.

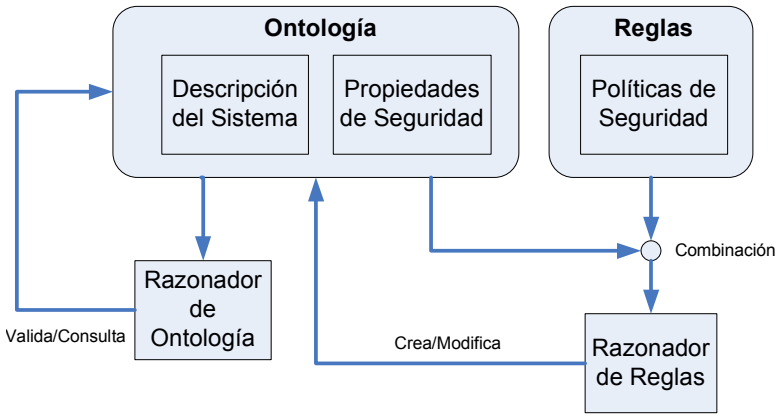

Figura 3.11: *Razonamiento sobre políticas de seguridad*

3.4.1. Razonamiento sobre la ontología CIM-OWL

CIM-OWL tiene como sintaxis a OWL-DL que a su vez está basado en Lógica Descriptiva [1]. Este tipo de lógica proporciona diferentes técnicas de razonamiento que nos permiten superar las limitaciones semánticas de la

representación XML. Esta clase simple, pero a la vez con gran capacidad, de lógica de primer orden permite razonar no sólo en individuos sino también sobre su estructura con algoritmos robustos y eficientes.

En la primera fase de la especificación de políticas, como muestra la figura 3.11, el razonador OWL podría ser usado para:

- **Validación**. El lenguaje OWL permite expresar restricciones; la operación de validación es usada para detectar si estas restricciones son incumplidas por algún conjunto de datos, es decir, la validación consiste en una comprobación global a través del esquema y las instancias buscando inconsistencias.

- **Consultar** el modelo para reconocimiento de instancias (por ejemplo, comprobar si un individuo es una instancia de una clase) y reconocimiento de herencia (por ejemplo, comprobar si una clase es una subclase de otra clase y si una propiedad es una subpropiedad de otra). De esta manera el razonador facilita el acceso a la información que compone las políticas y a las propiedades de seguridad del sistema.

Para mostrar como el razonador OWL puede ser usado para estas tareas, presentamos un ejemplo de una ontología CIM-OWL. Este ejemplo presenta algunas de las clases de la ontología CIM-OWL e instancias de las clases CIM_Role y CIM_Identity, además de la asociación CIM_MemberOfCollection entre ambas instancias que recordemos sirve para representar la pertenencia de una identidad a un role. Utilizando la sintaxis abstracta [98] de OWL tenemos la siguiente representación:

```
1  Namespace(moc=
2     <http://ants.um.es/schema_2.12/CIM_MemberOfCollection.owl#>)
3  Namespace(me=
4     <http://ants.um.es/schema_2.12/CIM_ManagedElement.owl#>)
5  Namespace(c=
6     <http://ants.um.es/schema_2.12/CIM_Collection.owl#>)
7  Namespace(r=
8     <http://ants.um.es/schema_2.12/CIM_Role.owl#>)
9  Namespace(i=
10    <http://ants.um.es/schema_2.12/CIM_Identity.owl#>)
11 Namespace(ex=
12    <http://ants.um.es/examples1#>)
13
14 Ontology(
15 ObjectProperty(moc:Collection
16    domain(moc:CIM_MemberOfCollection)
17    range (c:CIM_Collection))
18 ObjectProperty(moc:Member
19    domain(moc:CIM_MemberOfCollection)
```

```
20      range (me:CIM_ManagedElement))
21  DatatypeProperty(r:Name
22      domain(r:CIM_Role)
23      range (xs:string))
24  DatatypeProperty(i:InstanceID
25      domain(i:CIM_Identity)
26      range (xs:string))
27  Class(i:CIM_Identity partial me:CIM_ManagedElement)
28  Class(c:CIM_Collection partial me:CIM_ManagedElement)
29  Class(me:CIM_ManagedElement partial)
30  Class(r:CIM_Role partial c:CIM_Collection)
31  Class(moc:CIM_MemberOfCollection partial)
32  Class(moc:ComplementOfCIM_MemberOfCollection complete
33      complementOf(moc:CIM_MemberOfCollection))
34  Individual(ex:admin
35      type(r:CIM_Role)
36      value(r:Name "administrator"))
37  Individual(ex:user
38      type(r:CIM_Role)
39      value(r:Name "usuario"))
40  Individual(ex:felixgarcia
41      type(i:CIM_Identity)
42      value(i:InstanceID "umu:fgarcia"))
43  Individual(ex:lucaslopez
44      type(i:CIM_Identity)
45      value(i:InstanceID "umu:llopez"))
46  Individual(ex:felixadmin
47      type(moc:CIM_MemberOfCollection)
48      value(moc:Collection ex:admin)
49      value(moc:Member ex:felixgarcia))
50  Individual(ex:lucaslopezuser
51      type(moc:CIM_MemberOfCollection)
52      value(moc:Collection ex:user)
53      value(moc:Member ex:lucaslopez))
54  Individual(ex:lucaslopezadmin
55      type(moc:ComplementOfCIM_MemberOfCollection)
56      value(moc:Collection ex:admin)
57      value(moc:Member ex:lucaslopez))
58  )
```

Esta ontología incluye dos instancias de CIM_Role para representar dos roles, uno de administrador y otro de usuario, y dos instancias de CIM_Identity para representar dos identidades, una para felixgarcia y otra para lucaslopez. La primera identidad está asociada al role de administrador y la segunda identidad está asociada al role de usuario. Además, la segunda identidad tiene una asociación complemento al role administrador para representar que esta identidad tiene prohibida su pertenencia a este role.

Si un razonador OWL validara esta ontología, determinaría que no existe

ninguna inconsistencia. Si por el caso contrario, la ontología incorporara la siguiente instancia:

```
Individual(ex:lucaslopezadmin
    type(moc:CIM_MemberOfCollection)
    value(moc:Collection ex:admin)
    value(moc:Member ex:lucaslopez))
```

El razonador OWL detecta una inconsistencia ya que la instancia pertenece tanto al tipo CIM_MemberOfCollection como al complemento de este tipo ComplementOfCIM_MemberOfCollection. Otro tipo de inconsistencias pueden venir determinadas por incumplimiento en las cardinalidades de las propiedades o por usar referencias incorrectas en las asociaciones.

Las consultas pueden realizarse mediante el lenguaje RDQL [99] o SPARQL [102]. Estos lenguajes proporcionan un modelo de consulta orientado a los datos pero no realizan ningún tipo de inferencia. Es el razonador OWL el que realiza la inferencia que puede ser necesaria para responder a una consulta. SPARQL puede procesar consultas más complejas que RDQL; aún así disponen de un sintaxis muy similar. Un ejemplo RDQL de consulta sobre la estructura de clases puede ser determinar todas las subclases de CIM_ManagedElement.

SELECT ?x WHERE (?x <rdfs:subClassOf><me:CIM_ ManagedElement>)

Las consultas pueden ser más complejas como determinar cual es el conjunto de clases de asociación que referencia a una clase determinada o si existe una clase de asociación entre dos clases determinadas. También pueden realizarse consultas RDQL sobre las instancias por ejemplo para determinar todas las instancias de una clase de CIM_Role.

SELECT ?x WHERE (?x <rdf:type><r:CIM_ Role>)

Éste es otro ejemplo sencillo; otros más complejos pueden involucrar a diferentes individuos y interrelaciones entre ellos. RDQL junto con el razonador OWL facilitan la realización de cualquier tipo consulta que se crea conveniente sobre el dominio de gestión.

3.4.2. Razonamiento sobre la reglas SWRL-CIM

En la segunda fase de la especificación de la política, como muestra la figura 3.11, el razonador de reglas es usado para crear nuevas instancias o

modificar alguna instancia existente del conjunto de instancias de nuestra ontología CIM-OWL. Para ello en SWRL-CIM limitamos el uso de átomos de clase a clases nombradas, es decir a clases que son definidas puramente en CIM-OWL. Esta restricción determina un formato de las reglas que permite ser traducido fácilmente a motores de inferencia para aplicar técnicas de razonamiento basadas en reglas, incluyendo Prolog y Jena [27].

Por ejemplo, veamos el caso de una política de delegación. Para ello partimos de una ontología que incluye dos instancias de CIM_Role para representar dos roles, uno de administrador y otro de manager, y una instancia de CIM_AuthorizedPrivilege para representar un privilegio de ejecución. El role de administrador tiene asociado este privilegio, mientras el role manager no dispone de ningún privilegio asociado.

```
Namespace(ap=
   <http://ants.um.es/schema_2.12/CIM_AuthorizedPrivilege.owl#>)
Namespace(me=
   <http://ants.um.es/schema_2.12/CIM_ManagedElement.owl#>)
Namespace(c=
   <http://ants.um.es/schema_2.12/CIM_Collection.owl#>)
Namespace(r=
   <http://ants.um.es/schema_2.12/CIM_Role.owl#>)
Namespace(p=
   <http://ants.um.es/schema_2.12/CIM_Privilege.owl#>)
Namespace(as=
   <http://ants.um.es/schema_2.12/CIM_AuthorizedSubject.owl#>)
Namespace(ex=
   <http://ants.um.es/examples2#>)

Ontology(
ObjectProperty(as:Privilege
   domain(as:CIM_AuthorizedSubject)
   range (ap:CIM_AuthorizedPrivilege))
ObjectProperty(as:PrivilegedElement
   domain(as:CIM_AuthorizedSubject)
   range (me:CIM_ManagedElement))
DatatypeProperty(r:Name
   domain(r:CIM_Role)
   range (xs:string))
Class(ap:CIM_AuthorizedPrivilege partial p:CIM_Privilege)
Class(p:CIM_Privilege partial me:CIM_ManagedElement)
Class(r:CIM_Role partial c:CIM_Collection)
Class(c:CIM_Collection partial me:CIM_ManagedElement)
Class(me:CIM_ManagedElement partial)
Class(as:CIM_AuthorizedSubject partial)
Class(as:ComplementOf_CIM_AuthorizedSubject complete
   complementOf(as:CIM_AuthorizedSubject))
Individual(ex:authpriv
```

```
35    type(ap:CIM_AuthorizedPrivilege)
36    value(ap:InstanceID "positif:grantExecute"))
37  Individual(ex:admin
38    type(r:CIM_Role)
39    value(r:Name "administrator"))
40  Individual(ex:mgr
41    type(r:CIM_Role)
42    value(r:Name "manager"))
43  Individual(ex:adminauthpriv
44    type(as:CIM_AuthorizedSubject)
45    value(as:Privilege ex:authpriv)
46    value(as:PrivilegedElement ex:admin))
47  )
```

En nuestro ejemplo la política de delegación define una propagación de los privilegios del role administrador al role manager. Esta política se define en SWRL-CIM como sigue:

$$CIM\_AuthorizedPrivilege(AP1?) \wedge CIM\_Role(R1?) \wedge Name(R1?, "admin") \wedge CIM\_AuthorizedSubject(AS1?) \wedge Privilege(AS1?, AP1?) \wedge PrivilegedElement(AS1?, R1?) \wedge CIM\_Role(R2?) \wedge Name(R2?, "manager") \Rightarrow CIM\_AuthorizedSubject(AS2?) \wedge Privilege(AS2?, AP1?) \wedge PrivilegedElement(AS2?, R2?)$$

El razonador de reglas genera nuevos datos en el proceso de inferencia. Los datos generados sobre la ontología son una nueva instancia de la asociación CIM_AuthorizedSubject que asocia el role de manager y el privilegio de autorización que existe en la ontología. La representación en sintaxis abstracta de esta instancia es la siguiente:

```
1  Individual(ex:mgrauthpriv
2    type(as:CIM_AuthorizedSubject)
3    value(as:Privilege ex:authpriv)
4    value(as:PrivilegElement ex:mgr))
```

Una característica importante de nuestro razonador de reglas es que debe ser capaz de detectar cuando se está generando una instancia con unos valores clave que ya existen en la ontología, de manera que si la regla tiene como consecuente el generar una instancia, el razonador debe determinar si esta instancia existe ya o no en la ontología en base a las propiedades de las instancia. Si existe, el identificador de la instancia será el mismo que el de la instancia existente; en caso contrario se generará un identificador aleatorio que no exista en la ontología.

Así por ejemplo, supongamos que en nuestra ontología inicial se hubiera incluido la instancia:

```
Individual(ex:managerauthpriv
    type(as:ComplementOfCIM_AuthorizedSubject)
    value(as:Privilege ex:authpriv)
    value(as:PrivilegElement ex:mgr))
```

Ante la regla de delegación anterior el razonador hubiera utilizado el mismo nombre que se usa en esta instancia y, por tanto, el razonador hubiera modificado esta instancia obteniendo la siguiente:

```
Individual(ex:managerauthpriv
    type(as:CIM_AuthorizedSubject)
    type(as:ComplementOfCIM_AuthorizedSubject)
    value(as:Privilege ex:authpriv)
    value(as:PrivilegElement ex:mgr))
```

Además, la ontología es actualizada y entonces el razonador OWL puede ser utilizado para realizar cualquier tipo de consulta y, más importante aún, para detectar cualquier tipo de inconsistencia resultante de los cambios producidos por la regla en la ontología. En este ejemplo detectaría una inconsistencia en la instancia al pertenecer al mismo tiempo a CIM_AuthorizedSubject y a su complemento.

3.5. Detección de conflictos

Un conflicto ocurre cuando la definición de la política asigna diferentes especificaciones en el comportamiento de un elemento del sistema; por ejemplo, una permite a un usuario iniciar un servicio y otra prohíbe al mismo usuario iniciar el mismo servicio. El análisis de los conflictos de políticas se divide en dos tareas principales. Una primera tarea está centrada en la detección de conflictos en las políticas de seguridad y una segunda tarea está dedicada a resolver (o guiar a los administradores en el proceso de resolución de) estos conflictos. Para llevar a cabo estas tareas es necesario disponer de una clasificación de los posibles conflictos de políticas que se desean analizar. Para ello en el apartado 3.5.1 presentamos una clasificación de los conflictos de políticas que ha servido como marco de referencia en nuestra investigación.

En SWRL-CIM, identificamos un conflicto entre diferentes políticas de seguridad desde el punto de vista semántico cuando los datos inferidos de estas políticas por el razonador de reglas generan alguna inconsistencia en la base de conocimiento que es detectada por la validación del razonador de ontología. De manera que el razonador de reglas junto con el razonador de ontología facilitan el análisis de conflictos. Para ello hemos definido un

procedimiento que presentamos en el apartado 3.5.2 para la detección de conflictos de modalidad (o signo) que puede ser extendida para detectar cualquier tipo de conflicto entre políticas. Además este procedimiento permite identificar la regla o reglas que dan lugar al conflicto para asistir al administrador en el proceso de resolución del conflicto.

3.5.1. Clasificación de conflictos de políticas

La clasificación de conflictos de políticas que presentamos en este apartado toma como base los trabajos [64, 65], donde se identifican dos tipos de conflictos de políticas: conflictos semánticos y conflictos de modalidad. El conflicto semántico es dependiente de la aplicación y se da cuando, en el contexto de una aplicación concreta, las acciones especificadas por dos políticas llevan al sistema a un estado inconsistente. En cambio el conflicto de modalidades es independiente de la aplicación y ocurre cuando dos políticas tienen modalidad contraria, donde por modalidad entendemos el signo de la política donde el signo positivo identifica a lo requerido y el signo negativo a lo prohibido. La figura 3.12 muestra la clasificación de los conflictos de políticas.

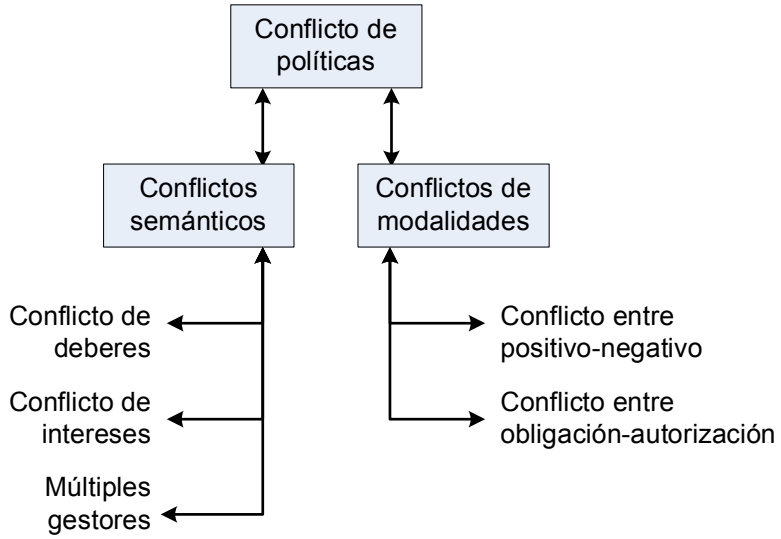

Figura 3.12: *Clasificación de conflictos de políticas*

Conflictos de modalidad

Podemos identificar dentro del control de acceso autorizaciones positivas (A+) que especifican permisos para una entidad y autorizaciones negativas

(A-) que especifican prohibiciones. También podemos hablar de grupo o role positivo (R+) que especifica el grupo o role al que pertenece una entidad y grupo o role negativo (R-) que especifica el grupo o role al que no puede pertenecer una entidad. Del mismo modo podemos hablar de obligaciones positivas (O+) que especifican las acciones que una entidad debe realizar y obligaciones negativas (O-) que especifican las acciones no debe realizar. Como puede verse la modalidad puede ser extendida a todo tipo de políticas de seguridad.

Los conflictos de modalidad típicos son los conflictos entre positivo y negativo; estos ocurren cuando dos políticas son opuestas en modo pero iguales en el resto. Así podemos identificar los siguientes conflictos:

- *Conflictos de autorización (A+/A-).* Este conflicto ocurre cuando una política especifica que a una entidad se le permite realizar una operación y otra política especifica que la misma entidad tiene prohibida la realización de la misma operación.

- C*onflictos de obligación (O+/O-).* Ocurre si una política especifica que una entidad está obligada a realizar una operación cuando otra política indica que la entidad se abstenga de realizar la misma operación.

- *Conflictos de grupo o role (R+/R-).* Éste ocurre si una política especifica que una entidad pertenece a un grupo o role y otra política especifica que la misma entidad tiene prohibido pertenecer a este grupo o role.

En las políticas de control de acceso también surge un conflicto entre las políticas de autorización y obligación en el caso de una obligación positiva y una autorización negativa (O+/A-). Este tipo de conflicto ocurre si una entidad es obligada a realizar una operación pero hay otra política de autorización que prohíbe a la entidad la realización de la operación. Por ejemplo, si un dispositivo tiene la obligación de iniciar un interfaz de red pero por otro lado tiene la prohibición de poder hacer esa operación nos encontramos ante este tipo de conflicto.

La detección de conflictos de modalidad no obliga a conocer detalles sobre el dominio administrativo, sólo es necesario detectar políticas iguales con modalidad opuesta. Esto permite definir técnicas de detección de conflictos genéricas que pueden ser usadas en cualquier entorno de aplicación.

Conflictos semánticos

Los conflictos semánticos son conflictos específicos de la aplicación; por ejemplo, dadas dos políticas de obligación, si una requiere que el servicio

sea actualizado con una nueva versión y otra requiere que el servicio sea suspendido, tendremos un conflicto de este tipo. Aquí el conflicto no aparece por la modalidad de las políticas, sino surge porque ambas acciones a realizar son incompatibles entre sí. Por tanto, éste es un ejemplo específico que depende de la aplicación.

Existen otros conflictos semánticos reconocidos en la literatura; a continuación se detallan los conflictos semánticos más relevantes para nuestra investigación.

- *Conflicto de deberes.* Éste ocurre cuando la misma entidad realiza dos operaciones incompatibles sobre la misma entidad objetivo. Un ejemplo de este conflicto surge cuando existe en el sistema el principio de que un mismo servicio no debe ser permitido a iniciar un interfaz de red y a enviar información por el interfaz de red, pero existen dos políticas en el sistema que autorizan a un servicio concreto a ambas operaciones.

- *Conflicto de intereses.* Ocurre cuando la misma entidad realiza operaciones incompatibles sobre diferentes entidades objetivo. Un ejemplo de este conflicto aparece cuando un mismo dispositivo está autorizado a realizar operaciones de enrutamiento y a realizar operaciones autónomas que implican la desconexión de la red, ambas operaciones son incompatibles entre sí.

- *Gestor múltiple.* Éste ocurre cuando diferentes entidades realizan operaciones sobre la misma entidad objetivo y el resultado de cada operación es incongruente con el otro. Un ejemplo es cuando una operación de mantenimiento sobre un dispositivo implica dejar fuera de servicio al dispositivo, y la operación de monitorización requiere que el dispositivo esté en servicio, ambas políticas entran en conflicto cuando se realizan simultáneamente.

La detección de conflictos semánticos obliga a conocer detalles sobre el dominio administrativo. En este sentido, el reto reside en conseguir definir técnicas de detección de conflictos genéricas que pueden ser usadas en cualquier entorno de aplicación.

3.5.2. Técnica para la detección de conflictos

La técnica para la detección de conflictos parte de la definición de tres posibles estados para cualquier asociación existente en la ontología CIM-OWL entre un elemento del sistema gestionado y sus propiedades de seguridad, junto con las transiciones permitidas entre estos estados. Los estados son:

- **No especificado**. La asociación no existe. Ni ha sido creada ni tampoco inferida.
- **Requerido**. Una instancia de la asociación existe.
- **Prohibido**. Una instancia del complemento de la asociación existe. Usamos el constructor complementOf para representar el complemento de una asociación.

Figura 3.13: *Transiciones entre estados*

Los arcos entre estados, las transiciones, identifican cuales son los casos en los cuales el sistema permite inferencias desde el razonador. Las posibles inferencias, y transiciones entre estados, son:

- No especificado → Requerido. El proceso de razonamiento crea una nueva asociación que no fue explícitamente especificada por el administrador.
- Requerido → No especificado. El administrador elimina una asociación existente previamente.
- No especificado → Prohibido. El proceso de razonamiento crea una nueva asociación complemento que no fue previamente especificada.
- Prohibido → No especificado. El administrador elimina una asociación complemento.

Las transiciones que no aparecen explícitamente en el autómata de la figura 3.13 son consideradas como conflictos. Por ejemplo, las transiciones Prohibido → Requerido y Requerido → Prohibido no son posibles en una única transición. Un administrador debe hacer una transición de estado a No especificado (es decir, eliminar la asociación) y entonces establecer el nuevo estado (es decir, Requerido o Prohibido) mediante la definición de la política correspondiente. En este sentido, el administrador es forzado a eliminar una asociación entre un elemento del sistema y una propiedad de seguridad de manera explícita para evitar de esta manera la posible aparición de un conflicto en las reglas que ha especificado.

Por medio del proceso de inferencia del razonador, el sistema es capaz de identificar un conflicto de política cuando una asociación es requerida y prohibida simultáneamente. Si identificamos el estado requerido a una modalidad positiva y el estado prohibido a una modalidad negativa, podemos identificar un conflicto de modalidad como una inconsistencia en el modelo. Esto permite al razonador detectar cualquier tipo de conflicto de modalidad que pueda darse en las políticas.

Veamos un ejemplo de conflicto de política de modalidad para cada tipo de política de SWRL-CIM.

- *Política de autenticación.* Por ejemplo, un conflicto de modalidad dado por una transición Prohibido→Requerido aparece cuando una instancia de CIM_AssignedIdentity existe entre una instancia de CIM_Identity y otra instancia de una subclase CIM_Credential (esto significa que la identidad identificada por la instancia CIM_Identity no debería ser autenticada y asignada a la entidad con la credencial representada por la instancia de CIM_Credential), y entonces el razonador de reglas infiere una nueva instancia CIM_AssignedIdentity entre ellas (esto significa que la entidad es autenticada y asignada al credencial). La representación OWL de los datos inferidos por el razonador de reglas con una inconsistencia es la siguiente:

```
1 Individual(ex:aut1
2    type(ai:ComplementOfCIM_AssignedIdentity)
3    type(ai:CIM_AssignedIdentity)
4    value(ai:IdentityInfo ex:ident1)
5    value(ai:ManagedElement ex:credencial1))
```

- *Política de control de acceso.* Por ejemplo para una política de autorización, una inconsistencia dada por una transición Requerido→Prohibido aparece cuando una instancia de CIM_AuthorizedSubject existe entre una instancia de CIM_Role y otra instancia de CIM_AuthorizedPrivilege (esto significa que el role representado por la instancia de CIM_Role tiene el privilegio de autorización identificado por la instancia de CIM_AuthorizedPrivilege), y entonces el razonador de reglas infiere una nueva instancia ComplementOfCIM_AuthorizedSubject entre ellas (esto significa que el role tiene prohibido el privilegio). La representación OWL de los datos inferidos por el razonador de reglas con una inconsistencia es la siguiente:

```
1 Individual(ex:auth1
2    type(as:CIM_AuthorizedSubject)
```

```
3     type(as:ComplementOfCIM_AuthorizedSubject)
4     value(as:Privilege ex:authpriv)
5     value(as:PrivilegedElement ex:role1))
```

- *Política de obligación.* Por ejemplo, una inconsistencia dada por una transición Prohibido→Requerido aparece cuando una instancia de ComplementOfCIM_HostedService existe entre una instancia de CIM_ComputerSystem y otra instancia de CIM_Service (esto significa que el sistema identificado por la instancia CIM_ComputerSystem está obligado a no proporcionar el servicio representado por la instancia CIM_Service), y entonces el razonador de reglas infiere una nueva instancia CIM_HostedService entre ellas (esto significa que el sistema tiene que proporcionar el servicio). La representación OWL de los datos inferidos por el razonador de reglas con una inconsistencia es la siguiente:

```
1 Individual(ex:oblig1
2     type(hs:ComplementOfCIM_HostedService)
3     type(hs: CIM_HostedService)
4     value(hs:Antecedent ex:system1)
5     value(hs:Dependent ex:service1))
```

- *Política de red.* Por ejemplo para una política de filtrado, una inconsistencia dada por una transición Prohibido→Requerido aparece cuando una instancia de ComplementOfCIM_HostedFilterList existe entre una instancia de CIM_ComputerSystem y otra instancia de CIM_FilterList (esto significa que la lista de filtros identificada por la instancia CIM_FilterList no debería ser aplicada al sistema representado por la instancia CIM_ComputerSystem), y entonces el razonador de reglas infiere una nueva instancia CIM_HostedFilterList entre ellas (esto significa que tal lista de filtros tiene que ser aplicada al sistema). La representación OWL de los datos inferidos por el razonador de reglas con una inconsistencia es la siguiente:

```
1 Individual(ex:filtro1
2     type(hfl:ComplementOfCIM_HostedFilterList)
3     type(hfl:CIM_HostedFilterList)
4     value(hfl:Antecedent ex:system1)
5     value(hfl:Dependent ex:filterlist1))
```

Esta técnica de detección de conflictos de modalidad es genérica y puede ser usada en cualquier entorno de aplicación. Además esta técnica puede ser extendida para la detección de conflictos semánticos, definiendo asociaciones que nos representen deberes, intereses y gestores, pero actualmente el

esquema CIM no dispone de estos conceptos y la introducción de estos conceptos implica una remodelación importante de todo el esquema CIM.

Por otro lado, el proceso de inferencia seguido por el razonador de reglas sirve al administrador como información de entrada para la resolución del conflicto ya que esta información incorpora qué reglas han sido disparadas en el proceso de inferencia y que datos han generado. De esta forma el administrador conoce que regla o reglas, además de que instancias, están involucradas en el conflicto de políticas.

3.6. Conclusiones

Este capítulo ha descrito un lenguaje semántico para la representación de políticas de seguridad que hemos denominado SWRL-CIM. Este lenguaje está basado en el uso de una representación formal del esquema CIM usando OWL-DL (CIM-OWL) para la definición de los conceptos de las políticas y en la representación lógica mediante SWRL de las reglas de políticas. De esta manera SWRL-CIM combina las ventajas de ambos, SWRL y CIM-OWL, consiguiendo una semántica clara y bien definida que permite múltiples niveles de abstracción en la especificación de políticas de seguridad.

Previamente a la descripción de CIM-OWL, hemos presentado una representación semi-formal del esquema CIM usando XML (xCIM) que supone una versión mejorada de las propuestas del DMTF para la representación de CIM usando XML. xCIM fue el primer paso en nuestra investigación pero sus limitaciones para ser combinada arbitrariamente con otras especificaciones de manera flexible y para usarse en tareas cruciales en el análisis de políticas como la deducción lógica o la interpretación semántica, nos llevó en nuestra investigación a definir CIM-OWL.

El uso de CIM-OWL introduce grandes ventajas en la definición de los conceptos de políticas. CIM-OWL permite disponer de una ontología extensible que permite añadir nuevos conceptos de políticas en cualquier momento sin obligar al rediseño o a la redefinición del lenguaje de políticas. También CIM-OWL puede ser utilizado como ontología común para conseguir la interoperabilidad entre diferentes lenguajes semánticos. Además un motor de inferencia OWL puede hacer razonamiento sobre los conceptos CIM-OWL para validar la ontología y conseguir acceso a la información de las políticas.

En cuanto al uso de SWRL, éste permite la definición de reglas Horn y facilita la aplicación de técnicas de razonamiento basado en reglas sobre las políticas, lo que permite obtener conocimiento adicional que no ha sido explícitamente especificado durante la definición del sistema. Además SWRL es fácilmente integrable en los motores de inferencia actuales y no está

limitado a expresar ningún tipo de política.

Por ello SWRL-CIM permite definir diferentes tipos de políticas de seguridad. De manera que en este capítulo presentamos como SWRL-CIM permite especificar políticas de autenticación, políticas de autorización, políticas de delegación, políticas de obligación, políticas de filtrado y políticas de seguridad IP. Además SWRL-CIM permite introducir restricciones de tiempo en las políticas y la clasificación de políticas mediante niveles de seguridad.

Por último, dadas las facilidades de razonamiento que SWRL-CIM nos proporciona mediante el razonamiento sobre la ontología CIM-OWL y el razonamiento sobre las reglas de las políticas, este capítulo presenta una técnica de razonamiento para la detección de conflictos basada en la generación y detección de inconsistencias en la definición. Así identificamos un conflicto entre diferentes políticas de seguridad desde el punto de vista semántico cuando los datos inferidos de estas políticas por el razonador de reglas generan alguna inconsistencia en la base de conocimiento que es detectada por la validación del razonador de ontología.

Capítulo 4

Entornos de Aplicación

Este capítulo presenta cómo nuestra propuesta para la especificación de políticas de seguridad expuesta en el capítulo anterior se integra en una arquitectura de gestión de políticas y cómo esta arquitectura se utiliza en diferentes entornos de aplicación. Para ello, en primer lugar presentamos los detalles de implementación del lenguaje y después como éste se integra dentro del framework de gestión definido en el proyecto IST POSITIF (del inglés, Policy-based Security Tools and Framework) [81]. Un prototipo de este framework nos permitirá validar el lenguaje y sus características en los entornos de aplicación seleccionados.

4.1. Introducción

SWRL-CIM es un lenguaje de políticas de seguridad que necesita, como cualquier otro lenguaje de políticas, una arquitectura de gestión que dé el soporte para la distribución y aplicación de las políticas de seguridad. Por su parte, como veíamos en el capítulo anterior, SWRL-CIM proporciona técnicas para el análisis de conflictos y para la consulta de información relativa a las políticas que facilitan las funciones de la arquitectura de gestión en la toma de decisiones.

Si bien el lenguaje SWRL-CIM dadas sus características semánticas y funcionales puede ser integrado fácilmente en cualquier arquitectura de gestión de políticas, en nuestro trabajo de investigación hemos utilizado un framework de gestión desarrollado dentro del proyecto IST POSITIF. Este proyecto utilizó inicialmente un lenguaje de políticas basado en CIM Policy con una representación xCIM y en la actualidad existe un prototipo que integra el uso del lenguaje SWRL-CIM.

IST POSITIF define un framework de gestión donde SWRL-CIM puede

ser utilizado para representar diversos tipos de políticas. Así SWRL-CIM puede ser utilizado para la especificación de políticas de seguridad tanto en servicios como a nivel de red. Para mostrar como este framework de gestión de políticas puede usarse en ambos casos y así validar el framework, hemos utilizado la arquitectura para la gestión de políticas de autorización en servicios Grid y la gestión de políticas de filtrado en firewalls distribuidos.

Previamente a mostrar como SWRL-CIM se integra en la arquitectura de POSITIF y los entornos de aplicación en Grid y firewall distribuidos, dedicamos una sección a describir los detalles más importantes sobre la implementación de las técnicas de inferencia y la edición de políticas con SWRL-CIM.

4.2. Implementación

La implementación de SWRL-CIM involucra dos decisiones importantes: el motor de inferencia a utilizar y la edición de las políticas de seguridad. En ambas decisiones disponemos de diferentes alternativas ya que existen diversos motores de inferencia y del mismo modo también existen diversos editores de SWRL. Además es importante conseguir la interacción entre ambos como muestra la figura 4.1. Esta interacción se basa en conseguir que las entradas del motor de inferencia y del editor estén en el formato correcto. Para ello pueden ser necesarias un serie de transformaciones.

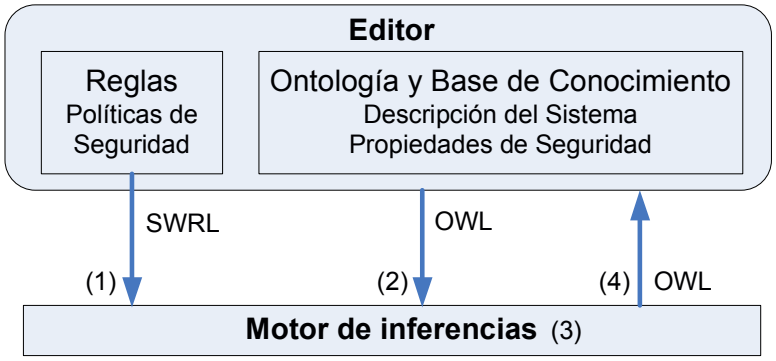

Figura 4.1: *Iteración entre editor de reglas y motor de inferencias*

Como muestra la figura 4.1, las reglas tienen que ser transformadas e introducidas en el motor de inferencia (1). Después, la ontología y la base de conocimiento tiene que ser transformada e introducida en el motor de inferencia (2). Después del razonamiento (3), los resultados del razonamiento deberían ser retornados en el formato del editor (4).

Nuestra implementación utiliza el motor de inferencia Jena-2 [85] que nos permite implementar las técnicas de razonamiento y, por tanto, las técnicas de detección de conflictos. Y el editor de políticas utilizado es ORE [15] que nos permite la definición de reglas SWRL tomando como base la ontología CIM-OWL de una manera guiada y fácil. ORE utiliza Jena-2 como motor de inferencia consiguiendo que no haya transformaciones en los pasos (2) y (4) de la figura 4.1. Además ORE incluye un módulo para transformar reglas SWRL a Jena-2, y viceversa, que facilita el paso (1) de la misma figura.

A continuación se detallan las principales características de Jena-2 y de ORE, además de compararlos con las principales alternativas que existen en la actualidad.

4.2.1. Motor de inferencia

Un requerimiento en nuestra búsqueda de un motor de inferencia para SWRL-CIM es que debemos encontrar un motor de reglas que soporte el razonamiento sobre ontologías OWL. Partiendo de este requerimiento hemos estudiado diferentes alternativas. Así, en la actualidad no existe ningún motor de reglas que soporte SWRL directamente, excepto Bossam [25] que recientemente ha incorporado soporte. Este motor que integra soporte para reglas SWRL es todavía una solución poco madura y aún no ha conseguido la robustez que tiene por ejemplo Jena-2. Otras alternativas que han sido estudiadas son por ejemplo Jess [24] y CommonRules [7] pero no fueron consideradas fundamentalmente por razones técnicas ya que ambas no soportan encadenamiento hacia atrás y CommonRules tampoco soporta cambios en las instancias en tiempo real, requerimientos ambos que se han considerado fundamentales en nuestra investigación. Además Jess y CommonRules disponen de limitaciones importantes en su licencia de uso. En nuestro estudio Jena-2 es identificado como el paquete que necesita menos esfuerzo para usar reglas SWRL a pesar de no disponer de soporte SWRL, ya que la transformación entre reglas SWRL y reglas Jena-2 es simple.

Las características que hacen a Jena-2 el motor de inferencia más adecuado en nuestro desarrollo son las siguientes:

- Dispone de un conjunto de paquetes Java que facilitan su integración en cualquier desarrollo de aplicaciones en Java. Por tanto, esto facilita en gran medida su integración en una arquitectura de gestión que esté desarrollada en este lenguaje como por ejemplo el framework de gestión del proyecto POSITIF.

- Soporta, además de los mecanismos internos del motor de reglas,

los dos métodos de razonamiento, encadenamiento hacia delante y encadenamiento hacia atrás.

- Dispone de un razonador propio para OWL. No depende de terceros en sus desarrollos, aunque permite integrar fácilmente otros razonadores con mayor soporte funcional como Pellet [48].

- Permite la definición de clases e instancias en OWL, además de soportar el lenguaje SPARQL [102] para realizar consultas sobre la ontología OWL. Los cambios en las clases, las instancias o las reglas pueden realizarse en tiempo real a través de la API, lo cual permite añadir nueva información a la base de conocimiento cuando se lanzan los procesos de razonamiento encaminados a detectar cualquier inconsistencia o conflicto existente en el conjunto de reglas.

- Se trata de código abierto con licencia tipo BSD. Este tipo de licencia de software libre está muy cercana al dominio público y permite el uso del código fuente en software no libre.

Aunque Jena-2 dispone de un formato propio para las reglas, las reglas SWRL pueden ser transformadas de forma sencilla a reglas Jena-2 desde su representación XML utilizando una transformación XSL. El Anexo D muestra un ejemplo de transformación XSL para transformar reglas SWRL a reglas Jena-2. Otras transformaciones pueden extraerse de otros paquetes y editores como por ejemplo ORE que dispone de la clase java *SWRLConverter* para transformar reglas SWRL a reglas Jena-2 y viceversa.

Por otro lado en Jena-2 es necesario el uso del razonador Pellet en tareas de razonamiento OWL que el propio razonador de Jena-2 no soporta. Es decir, Jena-2 dispone de soporte para OWL Lite pero no de un soporte completo para OWL DL. Para completar el razonamiento OWL DL usamos un razonador OWL DL externo como es Pellet. Esto se consigue gracias a que Jena-2 dispone de un interfaz DIG [41] que permite conectar cualquier razonador externo que soporte el estándar DIG como es Pellet (otras alternativas son Racer [89] o FaCT [16]).

4.2.2. Editor de reglas

Unos de los editores OWL más conocidos y utilizados por la comunidad investigadora son Protégé [87] y SWOOP [49]. Ambos editores disponen de soporte también para la especificación de reglas SWRL. Una de las primeras iniciativas fue la integración de Protégé y Jess mediante un plug-in que puede encontrarse en [77]. Posteriormente otra iniciativa [50] integró el uso

de SWRL en SWOOP utilizando Pellet. Otra alternativa es el editor gráfico Hoolet [17] con un razonador propio, aunque su uso no está tan extendido

Todas estas alternativas son válidas para la edición de reglas SWRL-CIM, si bien su uso no está orientado al objetivo de editar políticas de una manera guiada. Esto nos llevó plantearnos la creación de un editor que pudiera adaptarse a nuestras necesidades, más aún teniendo en cuenta que este editor debería integrarse en una arquitectura de gestión de políticas como la diseñada dentro del proyecto europeo de investigación POSITIF. Por esta razón, dentro de nuestro grupo de investigación, se ha desarrollado el editor ORE (del inglés, Ontology Rule Editor) [15].

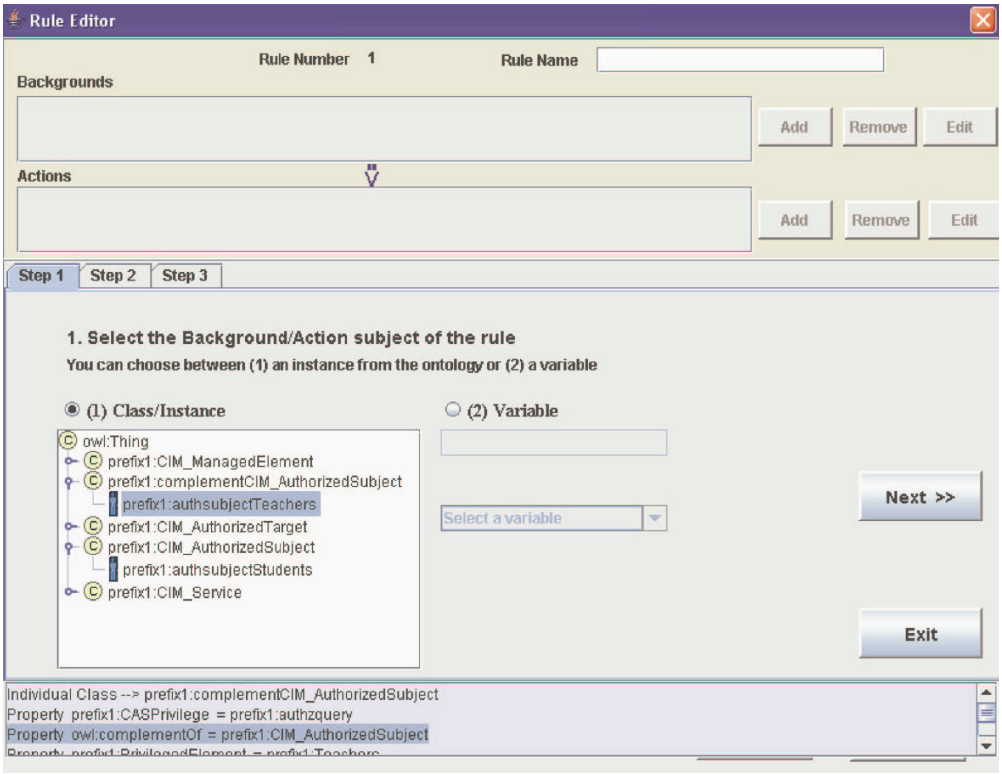

Figura 4.2: *Wizard del editor ORE*

ORE está orientado a la edición de reglas SWRL y ofrece varias funcionalidades:

- Carga la ontología y la base de conocimiento, es decir la descripción del sistema y las características de seguridad, bien mediante ficheros OWL o con la URI que identifica la ontología. Una vez que la ontología es cargada, ORE muestra información diversa sobre ésta

(versión, ontologías importadas, etc.) y genera un árbol jerárquico, que representa las clases e instancias contenidas en la ontología.

- Edición de reglas SWRL para la especificación de políticas de manera guiada gracias al Wizard que ORE incorpora. Los elementos de la regla tanto del lado izquierdo (antecedente) como del derecho (consecuente) son definidos de la misma manera, es decir, el Wizard guía al usuario en tres pasos: elegir el sujeto, el predicado y el objeto de cada elemento.

- Guarda las reglas SWRL de manera permanente en un repositorio. Por lo tanto, toda la carga semántica de las reglas es guardada para un uso posterior.

- Utiliza el motor de reglas de Jena-2 para la detección de conflictos entre las políticas. ORE envía las reglas SWRL al motor de inferencia, el motor las evalúa y los resultados son devueltos a ORE y mostrados al administrador. En este caso, el administrador puede decidir que los hechos y las acciones inferidas sean incluidas en la base de conocimiento.

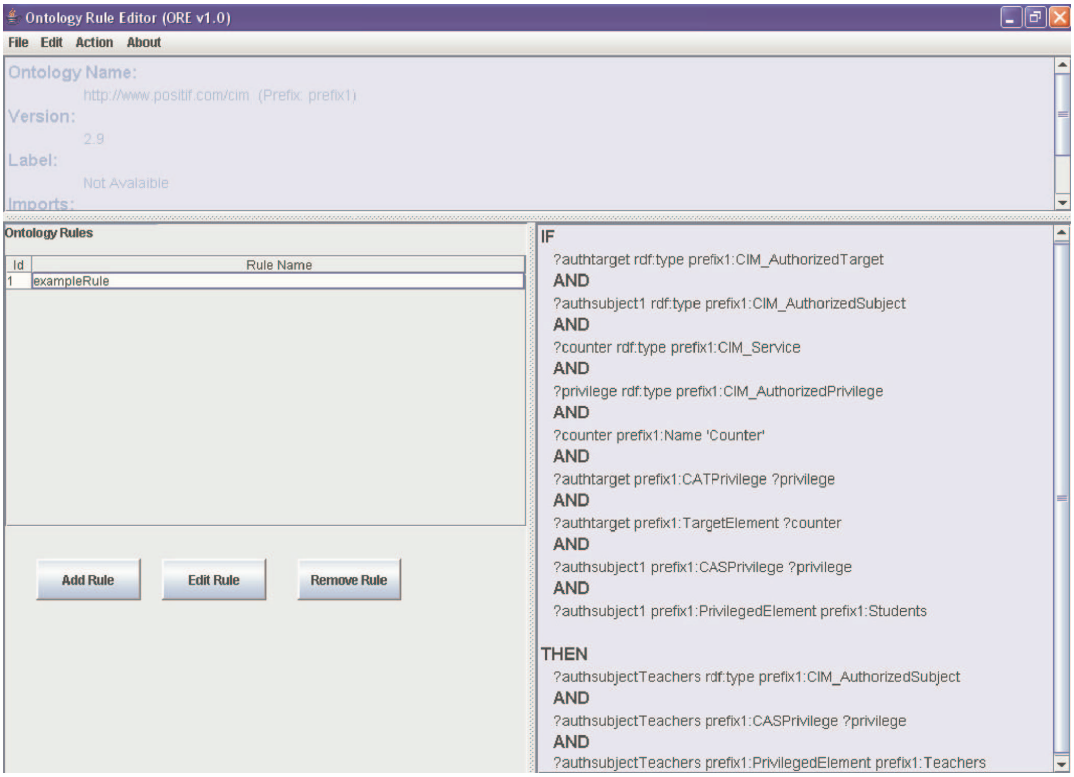

Figura 4.3: *Vista de las reglas en el editor ORE*

En cuanto a la especificación de políticas, la figura 4.2 muestra una captura de la ventana del Wizard del editor ORE. Éste está compuesto básicamente por un navegador a través de la ontología usada (en la parte inferior izquierda de la ventana) y dos listas (en la parte superior de la misma ventana) que reflejan los elementos del antecedente y el consecuente añadidos en la forma de tripletas RDF. El Wizard define tres fases que son usadas para crear cada tripleta: una para el sujeto, otra para la propiedad adjunta al sujeto y otra para el objeto. Una vez finalizada la especificación de la regla, ésta es guardada de manera permanente con el resto de reglas definidas. La figura 4.3 muestra la ventana que presenta una vista de todas las reglas editadas. En la parte derecha de la ventana se visualiza la regla en formato IF-THEN y los elementos en forma de tripletas RDF.

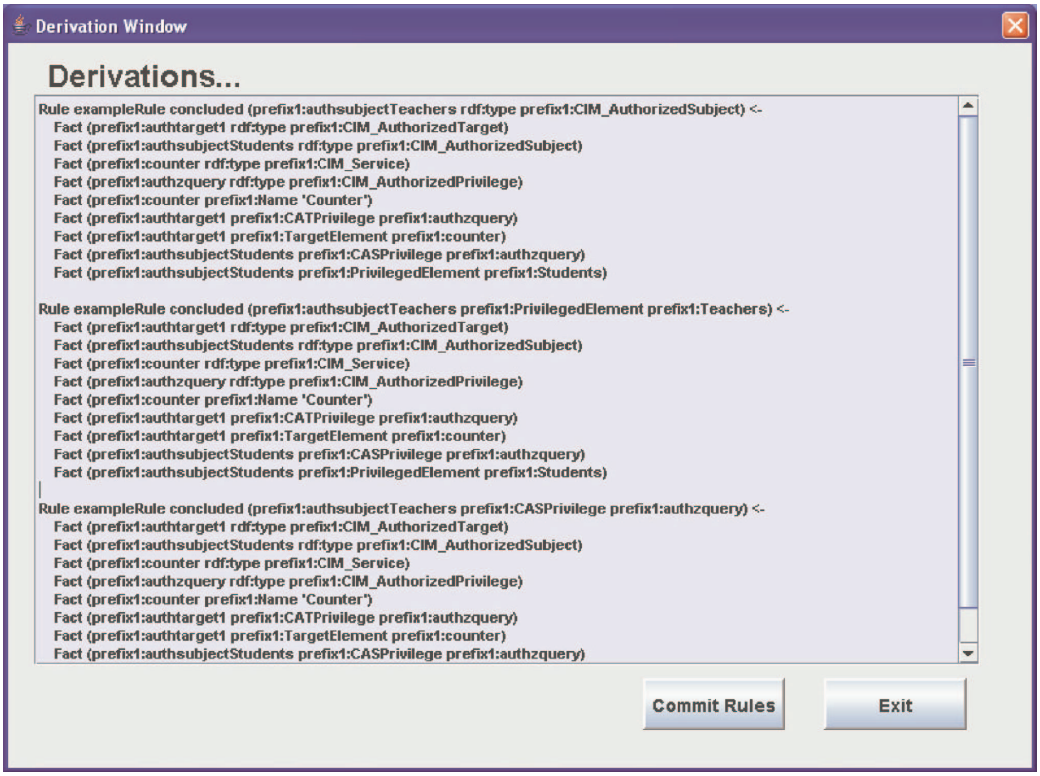

Figura 4.4: *Vista de los datos inferidos en el editor ORE*

En cuanto al proceso de razonamiento interno llevado a cabo en ORE éste es como sigue: primero, el razonador de reglas infiere datos del conjunto de reglas definidas por el administrador; el razonador de reglas comprueba los datos inferidos en la ontología, y si un conflicto es detectado el administrador es notificado; en otro caso los datos inferidos pueden ser añadidos a la

ontología si el administrador lo desea. Si no existe conflicto, se muestra la ventana de la figura 4.4 con los datos inferidos y dos botones: el botón Commit incluye las derivaciones en nuestra base de conocimiento y el botón Exit finaliza la edición de la regla sin incluirla en la base de conocimiento. Si por el contrario, se detecta un conflicto de política, se muestra el mensaje de alerta de la figura 4.5 con el texto 'Found conflict testing rules: Error (KB is inconsistent!)'.

De acuerdo con nuestra propuesta el razonador de reglas informa al administrador sobre el proceso de derivación mostrando las reglas que generan el conflicto, y el administrador puede modificar (o eliminar, si es necesario) algunas de las reglas para evitar el conflicto, aunque nuestros trabajos de investigación están siendo conducidos en la actualidad a proporcionar una recomendación al administrador sobre qué modificaciones pueden ser hechas en las reglas que provocan el conflicto para evitar tal situación.

Figura 4.5: *Mensaje de alerta de un conflicto en el editor ORE*

4.3. Arquitectura de Gestión

Establecidos los detalles de implementación de SWRL-CIM estamos en disposición de describir como debe realizarse su integración y su uso en una arquitectura de gestión. Para ello vamos a utilizar el framework de gestión de políticas de seguridad del proyecto IST POSITIF (del inglés, Policy-based Security Tools and Framework) [81]. Una de las principales metas del proyecto POSITIF es el diseño e implementación (como contribución a la comunidad de código abierto) de un framework para gestionar políticas de seguridad que pueden ser desarrolladas en diferentes tipos de dispositivos. Así, el objetivo del framework de POSITIF es proporcionar una vista unificada de la seguridad, lo cual por ejemplo es un requerimiento cuando tratamos con los servicios de seguridad a través de diferentes dominios de gestión.

A continuación se detallan los principales componentes del framework de gestión de POSITIF y como se integra el uso SWRL-CIM en el propio

framework.

4.3.1. Framework de POSITIF

El framework de POSITIF identifica un conjunto de bloques de seguridad (en inglés, security blocks) como dispositivos de red, servicios y/o aplicaciones que implementan alguna clase de capacidad de seguridad cuyos parámetros necesitan ser configurados y gestionados automáticamente. La figura 4.6 proporciona una vista funcional del framework de POSITIF incluyendo sus principales entradas y salidas, las cuales son:

- *Descripción del sistema.* Descripción formal del sistema que se pretende proteger usando las políticas de seguridad; tal definición será hecha principalmente en base a la topología de interconexión de la red y las características funcionales y de seguridad de los sistemas a proteger.

- *Política de seguridad.* Este elemento refleja la definición formal en uno o varios documentos de la política de seguridad que se pretende aplicar; para esto, se hará uso de un lenguaje de políticas multinivel, permitiendo la definición de requerimientos de alto nivel y/o controles detallados por parte del administrador de seguridad.

- *Amenazas y vulnerabilidades.* Base de datos externa que contiene información sobre patrones de ataques y vulnerabilidades de software y hardware bien conocidas en la comunidad Internet.

- *Mapa de seguridad de bloque.* Documento que describe como ciertos parámetros de seguridad genéricos que serán especificados como parte de la política de seguridad y la descripción del sistema pueden ser entonces configurados y aplicados en una versión específica de un sistema hardware o/y software.

- *Plug-ins.* Componentes software que implementan un interfaz particular y proporcionan las funcionalidades de un protocolo de comunicación (por ejemplo, SNMP o SOAP sobre HTTP) que puede ser usado para configurar automáticamente y monitorizar un bloque de seguridad específico.

Aunque estas entradas son consideradas como definidas externamente, el framework de POSITIF proporciona un interfaz para las bases de datos, y un formato para todos los documentos de entrada (es decir, descripción del sistema, política de seguridad y mapa de seguridad de bloque) y los componentes requeridos (es decir, los plug-ins).

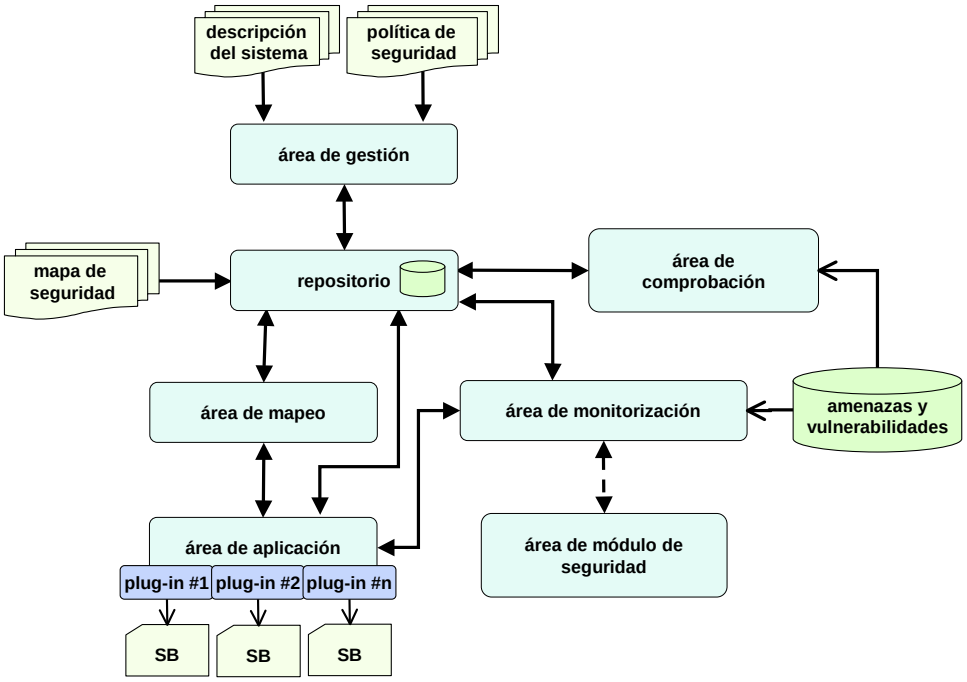

Figura 4.6: *Framework de gestión de POSITIF*

La figura 4.6 también presenta los bloques funcionales más relevantes del framework y que a continuación presentamos:

- *Área de gestión.* Esta área está a cargo de asistir a los administradores de la red y la seguridad en el proceso de definir (y gestionar) el comportamiento de seguridad deseado (especificado con la política de seguridad) y el sistema (definido con la descripción del sistema). Ésta también proporciona a los administradores la funcionalidad de configurar dinámicamente los componentes del framework y recuperar cierto estado e información de monitorización.

- *Repositorio del framework.* Este componente está a cargo de almacenar la descripción del sistema, las políticas de seguridad (en diferentes niveles de abstracción, desde especificaciones de políticas de alto nivel a configuraciones de bajo nivel), información de monitorización y parámetros de gestión del propio framework, y hacer toda esta información disponible a las correspondientes áreas del framework.

- *Área de comprobación.* La tarea principal de esta área es evaluar si el comportamiento deseado (es decir, la política de seguridad) es semánticamente coherente y puede ser implementada correctamente en

el sistema (definido con la descripción del sistema). Si ésta no puede ser implementada (porque hay un conflicto en las reglas o la política está referida a un servicio de seguridad no soportado en el sistema), será reportado a los administradores.

- *Área de mapeo.* Esta área está principalmente dirigida a producir las configuraciones particulares que serán más tarde aplicadas en los bloques de seguridad mediante el área de aplicación. Para hacer esto, un cierto número de mapas de seguridad de bloque deberían ser proporcionados al framework para permitir generar los parámetros de seguridad específicamente definidos en la política de seguridad y la descripción del sistema.

- *Área de aplicación.* Esta área está dirigida a aplicar las configuraciones particulares en los bloques de seguridad (dispositivos de red, servicios o aplicaciones). Para aplicar las configuraciones, algunos plug-ins pueden ser requeridos; ellos son definidos siguiendo un interfaz particular e implementando un protocolo para la configuración del dispositivo, tal como HTTP, SNMP [6], SSH [105], COPS [23] o COPS-PR [9]; algunos de ellos son proporcionados por el propio framework, aunque la intención principal es permitir que estos plug-ins sean desarrollados externamente según sean requeridos.

- *Área de módulo de seguridad.* Esta área está directamente relacionada con un conjunto de módulos de seguridad que pueden ser desarrollados como parte de la arquitectura. Estos módulos ligeros y de poco tamaño que pueden ser instalados en los dispositivos del sistema, añaden características de protección y capacidades de monitorización al framework. En este sentido actúan como sensores distribuidos en materia de seguridad.

- *Área de monitorización.* Esta área actúa como un monitor para la detección proactiva de intrusos además de la detección de intrusos estándar basada en aproximaciones reactivas (comprobando patrones de ataques). Este avance proactivo está basado en la política de seguridad definida por los administradores y en la información recopilada por el área de módulo de seguridad. Así, una intrusión se identifica cuando 'algo no cumple con la política de seguridad'.

La implementación del framework se basa en el desarrollo de Servicios Web y en el uso del lenguaje de programación Java. Los detalles particulares sobre los interfaces definidos para interactuar entre cada área y cierto

código que puede ser usado para integrar componentes externos dentro del framework propuesto (que en un futuro próximo será distribuido como código abierto) puede verse en detalle en [82].

4.3.2. SWRL-CIM en el framework de POSITIF

SWRL-CIM cumple con los requerimientos de un lenguaje de políticas de seguridad para poder utilizarse en el framework de gestión de POSITIF descrito en el apartado anterior. Además la implementación propuesta para SWRL-CIM con el uso de tecnologías web y Java facilita aún más su integración en el framework.

Todas las áreas del framework, a excepción del área de aplicación y el área de módulo de seguridad, se ven involucradas en la integración. A continuación describimos el papel jugado por SWRL-CIM en cada una de las áreas.

- *Área de gestión.* Los administradores utilizan el editor ORE para la edición de las políticas de seguridad. De esta manera se consigue una especificación SWRL-CIM de las políticas de seguridad, mientras para la descripción del sistema se realiza mediante CIM-OWL. Este proceso de edición se puede realizar bien de forma completa, bien de manera más sencilla e intuitiva (obviando algunos de los detalles de la ontología y el lenguaje de reglas subyacentes) mediante el Wizard de ORE presentado con anterioridad.

- *Repositorio del framework.* ORE guarda las reglas SWRL-CIM de manera permanente en el repositorio del framework. Para ello utiliza un servicio web específico definido por el framework para acceder al repositorio y almacenar (o recuperar) las políticas de seguridad. Todas las especificaciones habrán sido validadas sintácticamente frente al esquema XML definido a tal efecto.

- *Área de comprobación.* El motor de razonamiento Jena-2 se utiliza para evaluar las políticas de seguridad definidas con SWRL-CIM y para detectar la existencia de conflictos de políticas. El resultado de la evaluación será reportado a los administradores a través del editor ORE. Si existe un conflicto, el administrador puede utilizar ORE para modificar las políticas de seguridad y evitar el conflicto. En caso de no existir ningún conflicto, el administrador puede utilizar ORE para incluir el resultado de la evaluación de las políticas de seguridad en la descripción del sistema.

- *Área de mapeo.* Las configuraciones se consiguen desde los resultados de la evaluación de las políticas de seguridad, es decir transformando las características de seguridad definidas en la descripción del sistema en configuraciones para los bloques de seguridad. Esta transformación toma como base los mapas de seguridad de bloque que serán normalmente transformaciones XSL.

- *Área de monitorización.* Esta área utiliza la descripción del sistema para comprobar si existe alguna inconsistencia con la información recopilada por el módulo de manera proactiva sobre el sistema. Esta inconsistencia se consigue utilizando el razonador OWL de Jena-2 y como entradas al razonador la propia descripción del sistema en CIM-OWL y la información recopilada del sistema también en CIM-OWL. Si se detecta alguna inconsistencia, ésta será reportada a los administradores a través del editor ORE y quedará almacenada para propósitos de auditoría en el repositorio del framework. La funcionalidad que proporciona esta área permite una detección de conflictos dinámica frente a la detección estática que se consigue en el área de comprobación.

En la actualidad, el proyecto POSITIF está completando la implementación del framework de gestión con el uso del lenguaje SWRL-CIM. Aún así se ha definido un prototipo para los entornos de aplicación que se describen en la siguiente sección: gestión de políticas de autorización en Globus Toolkit 4 y gestión de políticas de filtrado en escenarios de firewall distribuidos.

4.4. Entornos de aplicación

Varios son los ejemplos y demostradores que han sido identificados como parte del proyecto POSITIF, aunque para validar la integración de SWRL-CIM en el framework de POSITIF se han considerado como los más representativos y útiles los siguientes entornos de aplicación: gestión de políticas de autorización en Globus Toolkit 4 y gestión de políticas de filtrado en escenarios de firewall distribuidos. Para cada entorno de aplicación se ha desarrollado un plug-in que permite aplicar las configuraciones como se mostrará a continuación.

4.4.1. Gestión de la seguridad en servicios: Globus Toolkit 4

Globus Toolkit 4 (GT4) [39] proporciona una gestión de recursos efectiva para el entorno de la computación grid. También incluye servicios de seguridad para proporcionar acceso seguro a estos recursos y, por tanto, necesita servicios de gestión de la seguridad para determinar cuando los recursos pueden ser accedidos (o no) desde alguna otra entidad que forme parte de la infraestructura grid. En este sentido, POSITIF representa un complemento adecuado al sistema GT4, proporcionando un gran rango de capacidades de gestión de políticas de seguridad que dependen de mecanismos de aplicación específicos de la plataforma, y ciertamente ayudan a los gestores a definir políticas que deberían ser distribuidas y aplicadas en los componentes del grid.

Para proporcionar un interfaz entre un entorno grid basado en GT4 y el framework de POSITIF, hacemos uso de los mecanismos de POSITIF para gestionar la Infraestructura de Seguridad del Grid (en inglés, Grid Security Infrastructure o con el acrónimo GSI) [40]. GSI (también llamado GT Security) es la base para la capa de seguridad de GT4 y está compuesta de un conjunto de herramientas para gestionar certificados y de un conjunto de clases Java para integrar fácilmente la seguridad dentro de los servicios web. GSI ofrece a los desarrolladores características tales como la seguridad a nivel de mensaje y a nivel de transporte, autenticación mediante certificados digitales, varios esquemas de autorización, delegación de credenciales y diferentes niveles de seguridad. GSI es el único componente de GT4 que es necesario para la integración con el framework de POSITIF. El interfaz es un plug-in del framework, que denominamos GT4 Security Plug-in. El plug-in es en sí mismo también un plug-in de GT4 que da a los clientes y servicios grid la habilidad de comprobar la acción a realizar en base a las políticas actuales definidas en el framework. La figura 4.7 muestra la arquitectura básica.

GT Security incluye un framework de autorización que permite una variedad de esquemas de autorización: una lista de control de acceso mediante fichero (grid-mapfile), una lista de control de acceso definida por un servicio, un gestor de autorización a medida, y acceso a un servicio de autorización mediante el protocolo SAML.

El plug-in GT Security se define como un gestor de autorización a medida, de manera que el plug-in GT4 Security es un sistema middleware de autorización que permite a cualquier servicio utilizar la lógica de autorización de POSITIF mediante interceptores. Esto significa que cualquier solicitud para que lleve asociada una decisión de autorización hecha por un servicio GT4 es enviada a un paquete independiente (es decir, al framework de

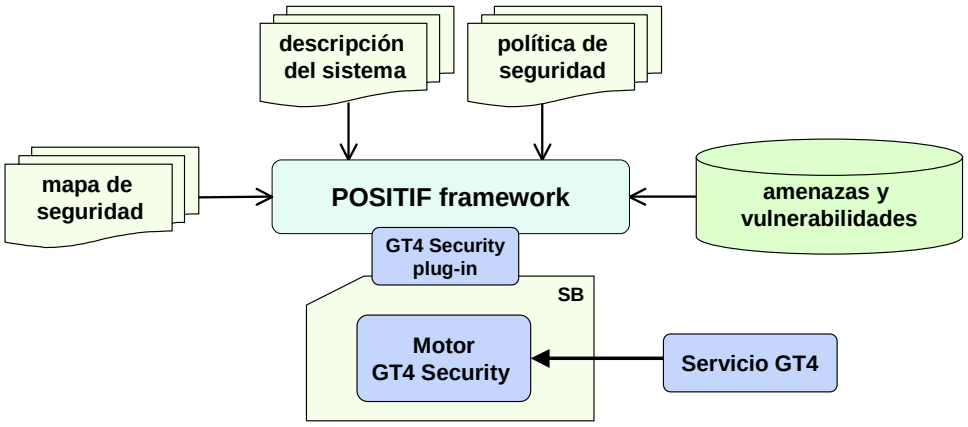

Figura 4.7: *Plug-in GT4 Security*

POSITIF vía el plug-in GT4 Security). Y, por tanto, el framework es responsable de gestionar la política asociada a un servicio GT4.

En cuanto a la configuración de la autorización en un servicio GT4, ésta involucra a una cadena de esquemas de autorización (también conocidos como Policy Decisions Points (PDPs)). Cuando una decisión de autorización necesita realizarse, los PDPs en la cadena son evaluados en orden para llegar una decisión de permiso o denegación. Todos los PDPs deben devolver una evaluación positiva para que la cadena finalmente evalúe el acceso como permitido. Para incorporar nuestro framework a la cadena de decisiones, el GT4 Security plug-in incluye dos interceptores: POSITIFAuthzPDP y POSITIFAuthzPIP (véase la figura 4.8). POSITIFAuthzPDP crea los métodos para implementar el interfaz GT4 para un PDP [8] que debe ser implementado por todos los PDPs en la cadena de interceptores. El sujeto puede contener credenciales públicas o privadas poseyendo atributos que en nuestro caso serán coleccionados y verificados por el interceptor POSITIFAuthzPIP.

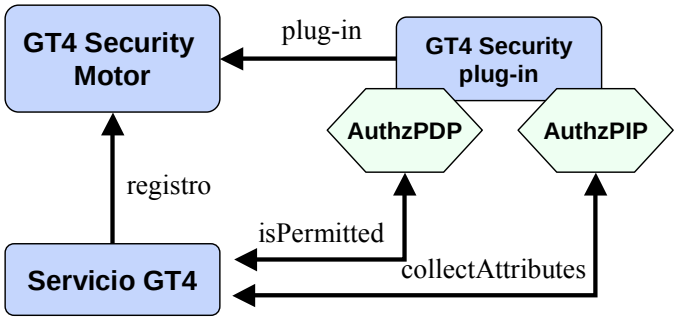

Figura 4.8: *Interceptores de autorización para un servicio GT4*

El siguiente ejemplo muestra un descriptor de seguridad para un servicio GT4 con la cadena de autorización configurada. Los valores elegidos para el alcance de los interceptores son pdpscope y pipscope.

```
<authz value='pipscope:org.globus.positif.authorization.POSITIFAuthzPIP
    pdpscope:org.globus.positif.authorization.POSITIFAuthzPDP'/>
```

Actualmente el plug-in GT4 Security ha sido desarrollado como un interceptor de autorización (en inglés, authorization interceptor); para otros tipos de políticas (por ejemplo, autenticación) el plug-in puede añadir esta nueva funcionalidad fácilmente gracias a la extensibilidad del framework de POSITIF.

4.4.2. Gestión de la seguridad en la red: Firewall distribuido

Los firewalls distribuidos [4] fueron propuestos como un mecanismo para reducir algunas de las limitaciones de los firewalls convencionales, especialmente las relativas a la dependencia de la topología (con conceptos tradicionales como perímetro de seguridad o puntos de entrada bien definidos para aplicar filtrado de tráfico), la existencia de tráfico cifrado extremo-a-extremo pasando a través del firewall sin inspección, y la necesidad de incrementar la protección contra personas o dispositivos de la propia organización. Nuestra investigación en esta área está dirigida a hacer uso del modelo de firewall distribuido, donde las políticas de firewall son definidas por un administrador y aplicadas a cada elemento individual de la red y no sólo a un elemento central, como ocurre tradicionalmente; en este sentido todos los dispositivos de la red (servidores, equipos finales, etc.) colaboran en la protección de la misma.

El motor de firewall es el único componente de cada elemento de la red que usamos en la integración. El interfaz es un plug-in del framework de POSITIF, que denominamos Plug-in Firewall (véase figura 4.9). En contraste con el Plug-in GT Security que delega todas las decisiones al framework (outsourcing), Plug-in Firewall está basado en la provisión de su configuración por adelantado desde el framework (provisioning). Actualmente Plug-in Firewall soporta iptables [100].

La aplicación de las configuración en los bloques de seguridad está basada en el uso de nuestra implementación [35] del protocolo COPS-PR [9], de manera que el bloque de seguridad inicia la comunicación COPS-PR con el plug-in enviándole cierta información interna (por ejemplo, su role y sus

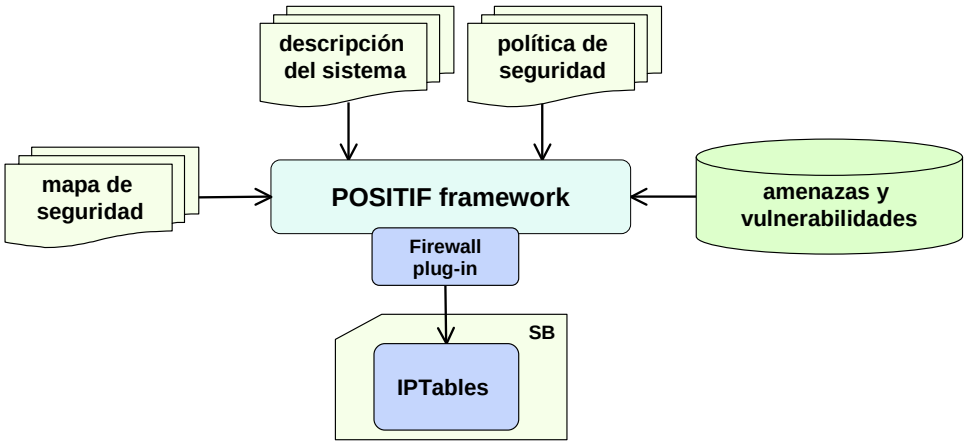

Figura 4.9: *Plug-in Firewall*

identificadores de red) y solicitando su configuración actual. La comunicación COPS-PR se mantiene indefinidamente; así si ocurre algún evento que implique un cambio en la configuración (por ejemplo, un cambio en los identificadores de red o un cambio en las políticas de filtrado), se inicia un intercambio de mensajes entre el bloque de seguridad y el plug-in para establecer la nueva configuración.

La configuración la determina el área de mapeo aplicando un mapa de seguridad de bloque que en este caso es una transformación XSL con la cual se consiguen a partir de las políticas de filtrado las reglas iptables a aplicar en el bloque de seguridad.

En el futuro, el plug-in o nuevos plug-ins añadirán nuevos motores de filtrado fácilmente gracias a la extensibilidad del framework de POSITIF.

4.5. Validación del framework

La figura 4.10 muestra el escenario utilizado para la validación del prototipo donde se combinan los componentes de los entornos de aplicación y los componentes del framework de gestión. El escenario presenta tres servidores y un conjunto de ordenadores personales (PC). Los servidores son un sistema con un servicio de conexión remota segura (SSH), otro con los servicios de Globus Toolkit 4 (GT4) y uno más con la plataforma POSITIF. El conjunto de ordenadores personales toman el role de investigador, jefe de laboratorio y administrador. Además identificamos la red interna donde están todos los servidores y la red externa que realiza el acceso a la red interna a través de un router.

En base a este escenario, se define un proceso de validación donde se

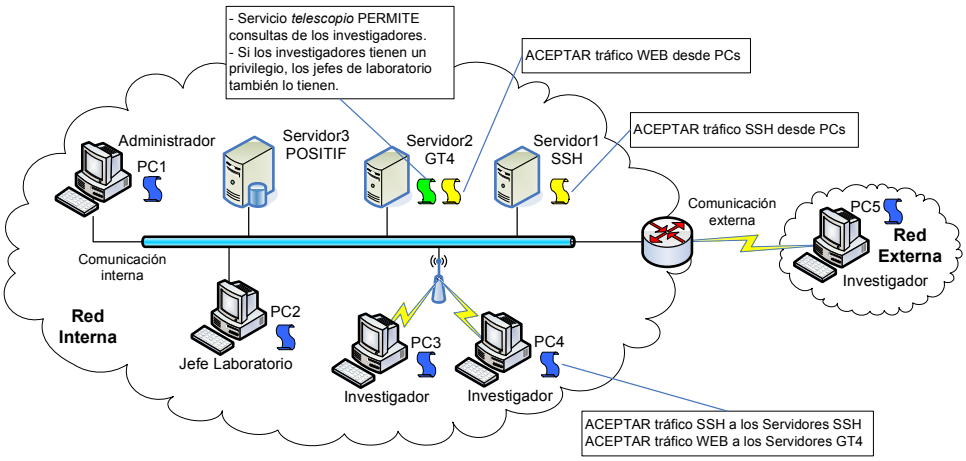

Figura 4.10: *Escenario para la validación del prototipo*

llevan a cabo una serie de pasos que muestran como un conjunto de políticas pueden especificarse y después aplicarse en el sistema. El proceso también permite validar la detección de conflictos. Estos son los pasos del proceso de validación:

1. *Descripción del sistema.* La descripción del sistema debe realizarse mediante la ontología CIM-OWL generando instancias para cada uno de los componentes del escenario. La edición de esta ontología debe realizarse con un editor OWL que valide la sintaxis. En nuestro caso hemos usado SWOOP, aunque otros editores OWL pueden ser utilizados como por ejemplo Protégé. Las principales clases CIM involucradas en la descripción del sistema son CIM_ComputerSystem para describir los servidores y los ordenadores personales, CIM_Service para describir los servicios GT4, CIM_Role para definir los roles, CIM_Privilege para representar los privilegios de autorización y CIM_FilterLists para representar los filtros de red que se aplicarán de manera distribuida para seguir el planteamiento de firewall distribuidos.

   El siguiente listado muestra un fragmento de la descripción del sistema usando CIM-OWL (véase el apartado 3.2.3.) que incluye los roles que aparecen en el escenario (administrador, investigador, jefe de laboratorio, servidor SSH, servidor GT4 y PC), dos ordenadores personales (pc1 y pc5), dos servidores (servidor 1 y servidor 2), el servicio *telescopio*, el privilegio para realizar consultas al servicio *telescopio* y las asociaciones de los sistemas con sus roles, las listas de filtros SSH y web para los servidores y clientes.

```
1 <rdf:RDF>
```

```
2         <CIM_Role rdf:about="#Administrador">
3                 <Name>Administrador</Name>
4         </CIM_Role>
5         <CIM_Role rdf:about="#Investigador">
6                 <Name>Investigador</Name>
7         </CIM_Role>
8         <CIM_Role rdf:about="#Jefelab">
9                 <Name>Jefe Laboratorio</Name>
10        </CIM_Role>
11        <CIM_Role rdf:about="#ServidorSSH">
12                <Name>Servidor SSH</Name>
13        </CIM_Role>
14        <CIM_Role rdf:about="#ServidorGT4">
15                <Name>Servidor GT4</Name>
16        </CIM_Role>
17        <CIM_Role rdf:about="#PC">
18                <Name>Ordenador personal</Name>
19        </CIM_Role>
20        <CIM_ComputerSystem rdf:about="#PC1">
21                <Name>pc1.positif.org</Name>
22        </CIM_ComputerSystem>
23        <CIM_ComputerSystem rdf:about="#PC5">
24                <Name>pc5.positif.org</Name>
25        </CIM_ComputerSystem>
26        <CIM_ComputerSystem rdf:about="#Servidor1">
27                <Name>servidor1.positif.org</Name>
28        </CIM_ComputerSystem>
29        <CIM_ComputerSystem rdf:about="#Servidor2">
30                <Name>servidor2.positif.org</Name>
31        </CIM_ComputerSystem>
32        <CIM_MemberOfCollection rdf:about="#PC1esPC">
33                <Member rdf:resource="#PC1" />
34                <Collection rdf:resource="#PC" />
35        </CIM_MemberOfCollection>
36        <CIM_MemberOfCollection rdf:about="#PC5esPC">
37                <Member rdf:resource="#PC5" />
38                <Collection rdf:resource="#PC" />
39        </CIM_MemberOfCollection>
40        <CIM_MemberOfCollection rdf:about="#Servidor1esSSH">
41                <Member rdf:resource="#Servidor1" />
42                <Collection rdf:resource="#ServidorSSH" />
43        </CIM_MemberOfCollection>
44        <CIM_MemberOfCollection rdf:about="#Servidor2esGT4">
45                <Member rdf:resource="#Servidor2" />
46                <Collection rdf:resource="#ServidorGT4" />
47        </CIM_MemberOfCollection>
48        <CIM_Service rdf:about="#telescopio">
49                <Name>Telescopio</Name>
50        </CIM_Service>
```

```
51          <CIM_AuthorizedPrivilege rdf:about="#authzquery">
52                  <InstanceID>positif:authzquery</InstanceID>
53                  <PrivilegeGranted>true</PrivilegeGranted>
54                  <Activities>Query</Activities>
55          </CIM_AuthorizedPrivilege>
56          <CIM_AuthorizedTarget rdf:about="#authtarget1">
57                  <CATPrivilege rdf:resource="#authzquery"/>
58                  <TargetElement rdf:resource="#telescopio"/>
59          </CIM_AuthorizedTarget>
60          <CIM_FilterList rdf:about="#sshClienteFilterList">
61                  <Name>SSH_cliente</Name>
62          </CIM_FilterList>
63          <CIM_FilterList rdf:about="#sshServidorFilterList">
64                  <Name>SSH_servidor</Name>
65          </CIM_FilterList>
66          <CIM_FilterList rdf:about="#webClienteFilterList">
67                  <Name>Web_cliente</Name>
68          </CIM_FilterList>
69          <CIM_FilterList rdf:about="#webServidorFilterList">
70                  <Name>Web_servidor</Name>
71          </CIM_FilterList>
72  </rdf:RDF>
```

2. *Definición de las políticas de seguridad.* En este paso utilizamos SWRL-CIM (véase el apartado 3.3.) para crear las políticas de seguridad en base a la descripción del sistema realizada en el paso anterior. La edición de las políticas se realiza con el editor ORE tal y como se ha descrito anteriormente. Las políticas que hemos definido para cada entorno de aplicación son:

   - Una política de autorización básica para un servicio GT4, 'El servicio telescopio permite consultas de los investigadores', y una política de delegación, 'Si los investigadores tienen un privilegio, los jefes de laboratorio también lo tienen'.
   - Políticas de filtrado, 'los servidores SSH aceptan tráfico SSH desde los PCs', 'los servidores GT4 aceptan tráfico web desde los PCs' y 'los PCs aceptan tráfico SSH a los servidores SSH y tráfico web a los servidores GT4'.

   La figura 4.10 muestra las políticas y también a qué sistemas del escenario afecta cada política de seguridad.

   La política de autorización que acabamos de comentar consiste en asociar el privilegio de consulta del servicio *telescopio* al role Investigador. Su representación lógica es la siguiente:

$$CIM\_Role(R1?) \wedge Name(R1?, "Investigador") \Rightarrow CIM\_AuthorizedSubject(AS1?) \wedge Privilege(AS1?, \#authzquery) \wedge PrivilegedElement(AS1?, R1?)$$

La política de delegación indica que cualquier privilegio asociado al role Investigador debe estar también asociado al role Jefe de laboratorio. Su representación lógica es la siguiente:

$$CIM\_AuthorizedPrivilege(AP1?) \wedge CIM\_Role(R1?) \wedge Name(R1?, "Investigador") \wedge CIM\_AuthorizedSubject(AS1?) \wedge Privilege(AS1?, AP1?) \wedge PrivilegedElement(AS1?, R1?) \wedge CIM\_Role(R2?) \wedge Name(R2?, "JefeLaboratorio") \Rightarrow CIM\_AuthorizedSubject(AS2?) \wedge Privilege(AS2?, AP1?) \wedge PrivilegedElement(AS2?, R2?)$$

La política de filtrado para los servidores SSH consiste en asignar la lista de filtrado correspondiente a los sistemas con el role Servidor SSH. Su representación lógica es la siguiente:

$$CIM\_Role(R1?) \wedge Name(R1?, "ServidorSSH") \wedge CIM\_ComputerSystem(CS1?) \wedge CIM\_MemberOfCollection(MOC1?) \wedge Collection(MOC1?, R1?) \wedge Member(MOC1?, CS1?) \Rightarrow CIM\_HostedFilterList(HFL1?) \wedge Antecedent(HFL1?, CS1?) \wedge Dependent(HFL1?, \#sshServidorFilterList)$$

Del mismo modo, la política de filtrado para los servidores web consiste en asignar la lista de filtrado correspondiente a los sistemas con el role Servidor Web. Su representación lógica es la siguiente:

$$CIM\_Role(R1?) \wedge Name(R1?, "ServidorGT4") \wedge$$
$$CIM\_ComputerSystem(CS1?) \wedge$$
$$CIM\_MemberOfCollection(MOC1?) \wedge$$
$$Collection(MOC1?, R1?) \wedge Member(MOC1?, CS1?)$$
$$\Rightarrow$$
$$CIM\_HostedFilterList(HFL2?) \wedge$$
$$Antecedent(HFL2?, CS1?) \wedge$$
$$Dependent(HFL2?, \#webServidorFilterList)$$

Y por último, la política de filtrado para los ordenadores personales consiste en asignar la lista de filtrado correspondiente a los sistemas con el role PC. Su representación lógica es la siguiente:

$$CIM\_Role(R1?) \wedge Name(R1?, "PC") \wedge$$
$$CIM\_ComputerSystem(CS1?) \wedge$$
$$CIM\_MemberOfCollection(MOC1?) \wedge$$
$$Collection(MOC1?, R1?) \wedge Member(MOC1?, CS1?)$$
$$\Rightarrow$$
$$CIM\_HostedFilterList(HFL3?) \wedge$$
$$Antecedent(HFL3?, CS1?) \wedge$$
$$Dependent(HFL3?, \#sshClienteFilterList)$$
$$CIM\_HostedFilterList(HFL4?) \wedge$$
$$Antecedent(HFL4?, CS1?) \wedge$$
$$Dependent(HFL4?, \#webClienteFilterList)$$

3. *Comprobación de las políticas de seguridad y almacenamiento permanentemente de los datos inferidos en el repositorio.* Una vez definidas las políticas, el framework de gestión utiliza el razonador de reglas Jena-2 para realizar las tareas de comprobación. Así el razonador nos permitirá chequear la validez de la especificación de las políticas dentro de la ontología y el conjunto de instancias. En nuestro caso, el comprobador valida las políticas y no detecta ningún conflicto. Para que el razonador Jena-2 interprete las reglas SWRL-CIM, éstas deben ser transformadas a formato Jena. Por ejemplo, el siguiente listado muestra la representación Jena-2 de la política de delegación.

```
1 #exampleRule:
2 ( ?authsubject1  rdf:type cimas:CIM_AuthorizedSubject )
3 ( ?privilege  rdf:type cimap:CIM_AuthorizedPrivilege )
4 ( ?role1  rdf:type cimrole:CIM_Role )
5 ( ?role2  rdf:type cimrole:CIM_Role )
6
```

```
( ?role1 cimrole:Name 'Investigador' )
( ?role2 cimrole:Name 'Jefe_Laboratorio' )
( ?authsubject1 cimas:Privilege ?privilege )
( ?authsubject1 cimas:PrivilegedElement ?role1)
->
( ?authsubject2  rdf:type cimas:CIM_AuthorizedSubject )
( ?authsubject2 cimas:Privilege ?privilege )
( ?authsubject2 cimas:PrivilegedElement ?role2)
```

Además el razonador de reglas realiza la inferencia sobre las reglas SWRL derivando nuevos individuos (es decir, nuevas instancias de clases o asociaciones CIM). De la política de autorización infiere la asociación entre el privilegio de consulta al servicio *telescopio* y el role Investigador. El siguiente listado muestra los datos inferidos.

```
<CIM_AuthorizedSubject rdf:about="#authsubject1">
        <CASPrivilege rdf:resource="#authzquery"/>
        <PrivilegedElement rdf:resource="#Investigador"/>
</CIM_AuthorizedSubject>
```

De la política de delegación deriva la asociación entre el privilegio de consulta al servicio *telescopio* y el role Jefe Laboratorio.

```
<CIM_AuthorizedSubject rdf:about="#authsubject2">
        <CASPrivilege rdf:resource="#authzquery"/>
        <PrivilegedElement rdf:resource="#Jefelab"/>
</CIM_AuthorizedSubject>
```

Las políticas de filtrado infieren la asociación de las diferentes listas de filtros a los sistemas. El siguiente listado muestra un fragmento de los datos inferidos.

```
<CIM_HostedFilterList rdf:about="#hostedFilterList1">
        <Antecedent rdf:resource="#servidor1" />
        <Dependent rdf:resource="#sshServidorFilterList" />
</CIM_HostedFilterList>
<CIM_HostedFilterList rdf:about="#hostedFilterList2">
        <Antecedent rdf:resource="#servidor2" />
        <Dependent rdf:resource="#webServidorFilterList" />
</CIM_HostedFilterList>
<CIM_HostedFilterList rdf:about="#hostedFilterList3">
        <Antecedent rdf:resource="#PC1" />
        <Dependent rdf:resource="#sshClienteFilterList" />
</CIM_HostedFilterList>
<CIM_HostedFilterList rdf:about="#hostedFilterList4">
        <Antecedent rdf:resource="#PC1" />
        <Dependent rdf:resource="#webClienteFilterList" />
</CIM_HostedFilterList>
```

Los datos inferidos son mostrados en el editor ORE y el administrador puede decidir si los incluye (o no) en la base de conocimiento. En nuestro caso, todos estos datos son guardados permanentemente en el repositorio del framework.

4. *Aplicación de las políticas de seguridad.* La aplicación de las políticas de seguridad implica su transformación en configuraciones particulares generadas a partir de los mapas de seguridad. En nuestro caso, la aplicación de las políticas se lleva tal y como se ha descrito en el apartado anterior tanto para Globus Toolkit como para firewall distribuidos haciendo uso para ello de los mecanismos que nos provee el framework diseñado e implementado como parte del proyecto europeo POSITIF.

5. *Definición de una nueva política de seguridad.* En este paso volvemos a especificar una política de seguridad pero con el objetivo de que entre en conflicto con alguna de las políticas definidas anteriormente. Así la política de autorización, 'El servicio telescopio no permite consultas de los jefes de laboratorio' entra en conflicto con la política de delegación especificada anteriormente. Su representación lógica es la siguiente:

$$CIM\_Role(R1?) \wedge Name(R1?, "JefeLaboratorio") \Rightarrow CIM\_ComplementOfAuthorizedSubject(CAS1?) \wedge Privilege(CAS1?, \#authzquery) \wedge PrivilegedElement(CAS1?, R1?)$$

6. *Detección y resolución de conflictos.* Una vez definida la nueva política el razonador de reglas realiza las tareas de comprobación y detecta la existencia de una incosistencia entre los datos inferidos (mostrados en el siguiente listado) y la base de conocimiento que describe el sistema. La existencia de un individuo que es instancia de la asociación CIM_AuthorizedSubject y de su complemento simultáneamente genera la inconsistencia.

```
<CIM_ComplementOfAuthorizedSubject rdf:about="#authsubject2">
        <CASPrivilege rdf:resource="#authzquery"/>
        <PrivilegedElement rdf:resource="#Jefelab"/>
</CIM_ComplementOfAuthorizedSubject>
```

El aviso de incosistencia se muestra el editor ORE junto con las reglas concretas que dan lugar al conflicto y el administrador debe resolver el conflicto.

4.6. Conclusiones

Este capítulo ha descrito como se ha conseguido una implementación completa del lenguaje semántico SWRL-CIM. En este sentido, se ha determinado el motor de inferencia a utilizar, Jena-2, y el editor de reglas, ORE, y su interacción para aplicar las técnicas de razonamiento necesarias para la detección de conflictos y la derivación de datos a partir de las reglas. Una vez determinada la implementación se ha descrito como se integra SWRL-CIM dentro del framework de gestión definido en el proyecto IST POSITIF. Para mostrar esta integración en el proyecto POSITIF se ha realizado un prototipo para dos entornos de aplicación: la gestión de políticas de autorización en Globus Toolkit 4 y la gestión de políticas de filtrado en escenarios de firewall distribuidos.

Además, este capítulo ha presentado un escenario final donde se combinan los componentes de los entornos de aplicación y los componentes del framework de gestión para la validación del prototipo. Hemos definido un proceso de validación donde se llevan a cabo una serie de pasos que muestran como un conjunto de políticas pueden especificarse y después aplicarse en el sistema utilizando el framework. Además en el proceso de validación se muestra cómo SWRL-CIM facilita la detección de conflictos.

Capítulo 5

Conclusiones y Vías Futuras

Una vez descrita nuestra propuesta y validada en un framework de gestión concreto, nos centramos en destacar los resultados más relevantes que se han conseguido como consecuencia de la labor de investigación que se ha llevado a cabo a lo largo del desarrollo de esta tesis doctoral. Para ello presentamos las conclusiones relativas al lenguaje semántico de políticas de seguridad propuesto, sin olvidar los entornos de aplicación que se han puesto en marcha en forma de prototipo dentro del framework de gestión del proyecto POSITIF para validar nuestra propuesta. También es objetivo de este capítulo el definir algunas de las vías futuras que se vislumbran como de mayor interés a partir del trabajo de investigación que se ha llevado a cabo.

La metodología que se va a seguir es la de hacer una revisión de los problemas más importantes que se han planteado tras el análisis y la comparativa de los lenguajes de política de seguridad que existen en la actualidad y las distintas soluciones que nuestra propuesta define para resolver dichos problemas, dejando en todo momento patentes las principales aportaciones que se han realizado como parte de esta tesis doctoral. De manera adicional, se va a aportar una visión de futuro que permita dejar constancia de que los resultados obtenidos representan un avance en una área que se vislumbra como de gran interés para el futuro de la gestión automática de la seguridad.

5.1. Conclusiones

A pesar del esfuerzo que está siendo realizado por un gran número grupos de investigación y organismos de estandarización, la realidad es que la gestión de la seguridad basada en políticas es un campo de investigación donde deben resolverse aún ciertos problemas. No en vano, es claro que la

actividad investigadora en esta línea está generando una gran producción científica en todo tipo de entornos de aplicación como son sistemas de computación grid, sistemas pervasivos, sistemas basados en Servicios Web, sistemas multi-agente y sistemas peer-to-peer.

Tomando como base esta realidad, muchas organizaciones llevan dedicando desde hace varios años importantes recursos, tanto técnicos como humanos, para la búsqueda de mecanismos efectivos que permitan proporcionar una gestión de la seguridad que sea automática en todos sus aspectos (desde la distribución de políticas a la resolución de conflictos). Como consecuencia de esta labor de investigación muchas han sido las propuestas que han surgido, como Ponder, KAoS o RuleML, por poner algunos ejemplos de las propuestas estudiadas.

A pesar de que ninguna de estas propuestas representa una solución final al problema de la provisión automática de seguridad, si que podemos afirmar que las soluciones semánticas son las propuestas de mayor interés para la gestión de la seguridad en los actuales y futuros sistemas y servicios de comunicaciones en red. Los argumentos para ello son, en primer lugar, su facilidad para el análisis de conflictos entre políticas. Esta tarea, sin lugar a dudas, es identificada por la comunidad investigadora como la tarea más compleja que debe resolver un framework de gestión. De manera adicional, destacar que los lenguajes semánticos facilitan procesos tales como la definición de los elementos que componen la política y el acceso a su información; la interoperabilidad entre diferentes lenguajes; la extensibilidad del lenguaje; y la integración en frameworks distribuidos de gestión. Es por ello que hemos considerado a los lenguajes semánticos como el punto de partida para el desarrollo del objetivo principal de esta tesis doctoral, a saber, la definición de un lenguaje de especificación de políticas seguridad.

Pero no basta sólo con tener una propuesta para la representación de políticas de seguridad. De hecho, y al igual que ocurre con las demás soluciones y propuestas semánticas, es necesario disponer de una técnica para el análisis de conflictos, en nuestro caso concreto, una técnica de razonamiento para la detección de conflictos basada en la generación y detección de inconsistencias en la definición. Esta técnica representa el otro punto de partida, junto con el uso de una representación semántica, del que se ha hecho uso en esta tesis para definir un lenguaje de políticas de seguridad.

Sobre esta base, es decir representación semántica y técnicas para la detección de conflictos, se ha diseñado un lenguaje de políticas de seguridad capaz de definir políticas de autenticación, autorización, delegación, obligación, filtrado y seguridad IP. Este lenguaje está basado en tres componentes principales, a saber, un modelo de información estandarizado e independiente de cualquier representación, denominado CIM,

que proporciona una semántica clara y bien definida a los elementos del lenguaje; una representación OWL de los conceptos e instancias del modelo de información CIM (Common Information Model), denominada ontología CIM-OWL, que permite la extensibilidad y facilita la interoperabilidad con el resto de lenguajes semánticos de políticas; y una representación lógica mediante SWRL de las reglas de políticas, denominada SWLR-CIM, que define propiamente al lenguaje de políticas y permite usar motores de inferencia para aplicar los mecanismos de razonamiento sobre las políticas.

Las conclusiones principales que se han obtenido en relación con el lenguaje de políticas de seguridad que se ha propuesto, son las que se presentan a continuación.

- Nuestra propuesta define una semántica clara y sin ambigüedad. En este sentido SWRL-CIM define la especificación de reglas de políticas, mientras CIM-OWL define la especificación de los conceptos relativos a las políticas.

- Nuestro lenguaje permite diferentes niveles de abstracción de la políticas ya que permite diferente grado de detalle en la definición de las reglas de políticas. Así CIM-OWL permite especificar detalles concretos de los recursos como son su dirección IP o el algoritmo de cifrado utilizado, y también, por ejemplo, conceptos de mayor nivel de abstracción como son roles o privilegios.

- Se trata de un lenguaje extensible. La ontología CIM-OWL permite añadir nuevos conceptos de políticas en tiempo de ejecución. En este sentido la ontología puede ser extendida en cualquier momento y esto no involucra el rediseño o la redefinición del lenguaje de políticas.

- El lenguaje permite la interoperabilidad entre lenguajes semánticos. Esto se consigue fácilmente compartiendo una ontología común. En este caso, CIM-OWL puede ser utilizada como ontología común entre diferentes lenguajes aunque la definición de reglas de políticas sigan metodologías diferentes.

- Nuestra propuesta facilita la detección de conflictos entre políticas. SWRL-CIM no está integrado directamente con ningún motor de razonamiento, pero SWRL es fácilmente integrable en motores ya existentes que permitan razonamiento basado en reglas y así aplicar las técnicas propuestas para conseguir la detección de conflictos de políticas en base a las inconsistencias generadas por el razonamiento en la ontología CIM-OWL. Además el uso de representaciones semánticas

permiten el uso de técnicas de razonamiento sobre las políticas que facilitan el acceso a la información de la política.

El procedimiento definido para la detección de conflictos está basado en un autómata donde los estados definen tres posibles modos para cualquier asociación en la ontología CIM-OWL entre un elemento del sistema gestionado y sus propiedades de seguridad. Las transiciones entre estados ocurren cuando nueva información es generada por el razonador de reglas cuando una nueva política de seguridad es definida. De manera que las transiciones que no aparecen explícitamente en el autómata generan la aparición de una inconsistencia en la ontología CIM-OWL. Estas inconsistencias son detectadas por el razonador de la ontología y las identificamos como conflictos.

Gracias a las características de nuestra propuesta cualquier framework de gestión podría integrar su uso para la especificación de políticas. En este sentido, hemos descrito como conseguir una implementación completa de nuestra propuesta. Se ha determinado el motor de inferencia a utilizar, Jena-2, y el editor de reglas, ORE (Ontology Rule Editor), y su interacción para aplicar las técnicas de razonamiento necesarias para la detección de conflictos y la derivación de datos a partir de las reglas. Una vez determinada la implementación se ha descrito como se integra SWRL-CIM dentro del framework de gestión definido en el proyecto europeo de investigación POSITIF (Policy-based Security Tools and Framework). Para mostrar esta integración en el proyecto POSITIF se han realizado dos prototipos que tratan de la gestión de políticas de autorización en Globus Toolkit 4 y gestión de políticas de filtrado en escenarios de firewall distribuidos.

Las principales conclusiones en relación con la implementación y su integración en el framework de gestión de POSITIF son las que ahora se presentan.

- Existen diferentes soluciones a la implementación de nuestra propuesta. Aunque nuestra elección ha sido Jena-2 como motor de inferencia, existe la posibilidad de utilizar otros motores de inferencia. Además la edición de reglas puede también realizarse con otros editores como Protégé o SWOOP.

- El framework de gestión de POSITIF es un framework basado en Servicios Web con un gran número de interfaces que facilitan en gran medida la integración del editor ORE para la especificación de políticas y del motor de inferencia Jena-2 en las diferentes áreas que componen el framework.

- La realización de dos prototipos para la gestión de la seguridad, uno en gestión de políticas de autorización en GT4 y otro en gestión de políticas de firewall, nos han permitido validar la integración entre nuestra propuesta y el framework, además de validar en entornos de aplicación nuestra propuesta.

- En base a un escenario concreto, hemos definido un proceso de validación que muestra como un conjunto de políticas pueden especificarse y después aplicarse en el sistema. También este proceso nos ha permitido comprobar como la técnica propuesta para la detección de conflictos funciona correctamente integrada con el framework de gestión.

Como conclusión final, es posible afirmar que el trabajo de investigación aquí presentado constituye un paso importante para permitir la gestión automática de políticas de seguridad proporcionando un camino a la extensibilidad, interoperabilidad, homogeneidad y escalabilidad de los servicios de seguridad en las redes y las aplicaciones existentes en los sistemas de comunicaciones actuales, y en aquellos que están por venir.

5.2. Vías Futuras

Surgen diversas vías futuras del trabajo de investigación presentado en esta tesis. En este sentido, y en lo referente al lenguaje de políticas SWRL-CIM, varias son las líneas de trabajo futuro que creemos de interés. La primera de ellas está en relación con el soporte de un mayor número de tipos de políticas de seguridad y la capacidad de describir un mayor número de conceptos. Para ello es necesario extender el modelo de información CIM y así introducir la definición de nuevos conceptos en la ontología CIM-OWL. Estos conceptos serían los necesarios para describir otros tipos de políticas de seguridad como políticas de privacidad o políticas de enrutamiento. Además estos nuevos conceptos pueden ser utilizados para describir entornos pervasivos, sistemas móviles o sistemas multi-agente. Claro está que estas definiciones deben hacerse colaborando con los grupos de trabajo que define el DMTF donde existen diversas propuestas para ampliar el modelo CIM.

Otras de las vías futuras de interés es la de extender los mecanismos de análisis de conflictos para poder detectar cualquier tipo de conflicto (por ejemplo el conflicto de intereses) y además facilitar la resolución de forma automática del conflicto. En este sentido, nuestra labor de investigación va en la dirección de definir un framework formal para la resolución automática de conflictos basada en agentes inteligentes y técnicas de argumentación

[60, 80, 79]. Así modelamos el problema como un sistema multi-agente en el cual los agentes tienen su propia base de conocimiento, donde un conflicto aparece cuando dos agentes tienen políticas inconsistentes entre sí y el proceso de argumentación busca la armonización de las políticas, es decir la desaparición de conflictos. De esta manera el propio framework puede resolver automáticamente el conflicto, aunque también este proceso puede ser utilizado para proponer al administrador una resolución del conflicto y que sea el propio administrador quien decida que acción tomar.

Por su parte el framework de gestión de políticas definido en el proyecto POSITIF debe alcanzar una integración completa con el lenguaje SWRL-CIM y con las técnicas de razonamiento propuestas. En la actualidad los entornos de aplicación presentados están basados en dos prototipos y, por tanto, se necesita un mayor esfuerzo en la implementación del framework para llegar a un desarrollo completo y extensible al resto de escenarios planteados dentro del propio proyecto POSITIF. Para ello fundamentalmente hay que integrar el motor de inferencia Jena-2 en todas las áreas de gestión en las que haya que razonar o inferir sobre las políticas o la ontología, además de integrar el editor ORE para la especificación de políticas con el editor para la especificación del dominio de gestión mediante la ontología CIM-OWL. Incluso una integración avanzada debería involucrar el uso de interfaces web semánticas mediante OWL-S (OWL Services) [10] y la representación de cualquier tipo de información (mapas de seguridad de bloque, amenazas y vulnerabilidades) mediante OWL.

Otra vía futura viene dada por el hecho de conseguir la interoperatividad entre el framework de gestión de SWRL-CIM y los framework de gestión del resto de lenguajes semánticos. Si bien los lenguajes lo facilitan en gran medida, es necesario un esfuerzo en el desarrollo e implementación de esta interoperatividad que además es imprescindible en escenarios multidominio donde sistemas gestionados individualmente y con diferentes frameworks de gestión deben operar conjuntamente e intercambiar información. En este sentido, un primer objetivo debería ser obtener la interoperabilidad entre los lenguajes KAoS y SWRL-CIM, y posteriormente entre los frameworks de gestión de ambos que podría conseguirse mediante la sincronización de sus repositorios.

Anexo A

Clasificadores (qualifiers) CIM

Los clasificadores en el esquema CIM son valores que proporcionan información adicional sobre clases, asociaciones, indicaciones, métodos, parámetros de métodos, instancias, propiedades o referencias. Todos los clasificadores tienen nombre, tipo, valor, alcance y un valor por defecto. Los clasificadores no pueden ser duplicados, de manera que una clase, instancia o propiedad no pueden tener más de un clasificador con el mismo nombre.

A continuación presentamos los diferentes clasificadores que define el modelo CIM [21]. En la definición de los clasificadores se define un valor por defecto que es el valor asumido para el clasificador cuando éste no es explícitamente especificado para elementos del modelo.

A.1. Meta-clasificadores

Estos clasificadores son usados para refinar el uso de una clase. Estos clasificadores son mutuamente excluyentes.

Tabla A.1: *Meta-clasificadores de CIM*

Clasificador	**Valor por defecto**	**Tipo**	**Significado**
ASSOCIATION	Falso	Boleano	El objeto de clase está definiendo una asociación.
INDICATION	Falso	Boleano	El objeto de clase está definiendo una indicación.

A.2. Clasificadores estándar

La siguiente tabla es una lista de los clasificadores estándar que cualquier implementación debe poseer. No todos los clasificadores pueden ser aplicados a todos los constructores del meta-modelo este hecho es indicado en la tabla con la columna 'Aplicado'.

Tabla A.2: Clasificadores estándar de CIM

Clasificador	**Valor por defecto**	**Aplicado**	**Tipo**	**Significado**
ABSTRACT	Falso	Clase, Asociación, Indicación	Boleano	Indica que la clase es abstracta y sirve sólo como base para nuevas clases. No es posible crear instancias de tal clase.
AGGREGATE	Falso	Referencia	Boleano	Define el componente padre de una asociación de agregación.
AGGREGATION	Falso	Asociación	Boleano	Indica que la asociación es una agregación.
ALIAS	NULL	Propiedad, Referencia, Método	Cadena	Establece un nombre alternativo para una propiedad o método en el esquema.
ARRAYTYPE	'Bag'	Propiedad, Parámetro	Cadena	Indica el tipo de una array. Sus valores válidos son 'Bag', 'Indexed' y 'Ordered'.
BITMAP	NULL	Propiedad, Método, Parámetro	Array de cadenas	Indica que posiciones de bit son significativos en un mapa de bits. La posición de un valor específico en el array define un índice que es usado para seleccionar una cadena en el array BitValues. Ver BitValues.

BITVALUES	NULL	Propiedad, Método, Parámetro	Array de cadenas	Proporciona la traducción entre un valor de posición de bit y una cadena asociada. Ver BitMap.
COUNTER	Falso	Propiedad, Método, Parámetro	Boleano	Representa a un entero negativo que se incrementa monótonamente.
DESCRIPTION	NULL	Todos	Cadena	Proporciona una descripción del elemento.
DEPRECATED	NULL	Clase, Asociación, Propiedad, Método	Array de cadenas	Indica que el elemento es tolerado pero no recomendado y podría ser sustituido por otro.
DISPLAYNAME	NULL	Todos	Cadena	Define un nombre que será mostrado en un interfaz de usuario en cambio del nombre actual del elemento.
DN	Falso	Propiedad, Parámetro	Boleano	Es usado para especificar que la propiedad o el parámetro deben ser un nombre distinguido (en inglés Distinguished name).
EMBEDDED OBJECT	Falso	Propiedad, Parámetro	Boleano	Es usado para incluir los datos de una instancia específica en una indicación o para capturar el contenido de una instancia en un instante concreto.
EXPERIMENTAL	Falso	Todos	Boleano	Indica que el elemento es experimental y podría sufrir modificaciones en versiones posteriores.
GAUGE	Falso	Propiedad, Método, Parámetro	Boleano	Representa un entero que puede incrementarse o decrementarse.

IN	Verdad	Parámetro	Boleano	Indica que el parámetro es usado para pasar valores a un método.
KEY	Falso	Propiedad, Referencia	Boleano	Indica que la propiedad es la clave o forma parte de la clave.
MAPPING STRINGS	NULL	Clase, Propiedad, Asociación, Indicación, Referencia	Array de cadenas	Indica la correspondencia del objeto dentro de otros proveedores de datos de gestión o agentes.
MAX	NULL	Referencia	Entero	Indica la máxima cardinalidad de la referencia.
MAXLEN	NULL	Propiedad, Método, Parámetro	Entero	Indica la máxima longitud, en caracteres, de una cadena para un elemento.
MAXVALUE	NULL	Propiedad, Método, Parámetro	Entero	Valor máximo de este elemento.
MIN	0	Referencia	Entero	Indica la mínima cardinalidad de la referencia.
MINVALUE	NULL	Propiedad, Método, Parámetro	Entero	Valor mínimo de este elemento.
MODELCOR RESPONDENCE	NULL	Propiedad	Array de cadenas	Indica una correspondencia entre una propiedad y otras propiedades del esquema.
NONLOCAL	NULL	Referencia	Cadena	Indica la localización de una instancia.
NON LOCALTYPE	NULL	Referencia	Cadena	Indica el tipo de localización de una instancia.
NULLVALUE	NULL	Propiedad	Cadena	Define el valor que indica que la propiedad es NULL.

OCTETSTRING	Falso	Propiedad, Parámetro	Boleano	Es usado para identificar a la propiedad o el parámetro como una cadena de octetos.
OUT	Falso	Parámetro	Boleano	Indica que el parámetro es usado para recuperar valores de un método.
OVERRIDE	NULL	Propiedad, Método, Referencia	Cadena	Indica que la propiedad, método o referencia en la clase sobrescribe al constructor con el mismo nombre en la clase padre en el árbol de herencia o en la clase padre especificada.
PROPAGATED	NULL	Propiedad	Cadena	Contiene el nombre de la clave que es propagada en una asociación weak.
READ	Verdad	Propiedad	Boleano	Indica si el acceso de lectura es permitido para el elemento.
REQUIRED	Falso	Propiedad	Boleano	Indica que un valor diferente de NULL es requerido para la propiedad.
REVISION	NULL	Clase, Asociación, Indicación, Esquema	Cadena	Proporciona el número menor de revisión del elemento.
SCHEMA	NULL	Propiedad, Método	Cadena	El nombre del esquema en el cual el elemento es definido.
SOURCE	NULL	Clase, Asociación, Indicación, Referencia	Cadena	Indica la localización de una instancia.
SOURCETYPE	NULL	Clase, Asociación, Indicación, Referencia	Cadena	Indica el tipo de localización de una instancia.

STATIC	Falso	Propiedad, Método	Boleano	Para métodos indica que es un método de clase. Para propiedades indica que es una variable de clase.
TERMINAL	Falso	Clase	Boleano	Indica que la clase no tiene subclases.
UNITS	NULL	Propiedad, Método, Parámetro	Array de cadenas	Proporciona las unidades en la que los datos del elemento son expresados.
VALUEMAP	NULL	Propiedad, Método, Parámetro	Array de cadenas	Define el conjunto de valores permitidos para esta propiedad, retorno de un método o parámetro. Puede utilizarse solo o en combinación con Values. Cuando es usado en combinación, la posición en ambos arrays asocia sus valores.
VALUES	NULL	Propiedad, Método, Parámetro	Array de cadenas	Proporciona la traducción entre un valor entero y una cadena asociada. Si ValueMap no está presente, Values es indexado desde el valor 0.
VERSION	NULL	Clase, Asociación, Indicación	Cadena	Proporciona el número mayor de revisión del elemento.
WEAK	Falso	Referencia	Boleano	Indica que las claves de la clase referenciada incluye las claves de los otros participantes en la asociación.
WRITE	Falso	Propiedad	Boleano	Indica si el acceso de escritura es permitido para el elemento.

De todos estos clasificadores que se definen en el meta-modelo, hay varios que no son utilizados en la versión actual del esquema CIM (versión 2.12). Éstos son Alias, DisplayName, NonLocal, NonLocalType, NullValue, Revision, Schema, Source, SourceType, Static y Terminal. Los clasificadores BitMap y BitValues son utilizados sólo en las propiedades LinkWidthActive y LinkSpeedActive de la clase CIM_IBPort. También es poco utilizado el clasificador DN que sólo aparece en la propiedad CADistinguishedName de la clase CIM_CertificateAuthority, y el clasificador READ se utiliza sólo con su valor por defecto y, por tanto, nunca toma el valor Falso.

A.3. Clasificadores opcionales y Clasificadores de usuario

Los clasificadores opcionales listados en esta tabla están pensados para situaciones que no son comunes en todas las implementaciones. Por tanto, las implementaciones pueden ignorar los clasificadores opcionales dado que no están obligados a interpretar o entender estos clasificadores.

Tabla A.3: Clasificadores opcionales de CIM

Clasificador	**Valor por defecto**	**Aplicado**	**Tipo**	**Significado**
DELETE	Falso	Asociación, Referencia	Boleano	Utilizado para indicar que la asociación o referencia debe ser borrada si una clase o asociación son borradas, respectivamente. Ver IfDeleted.
EXPENSIVE	Falso	Propiedad, Referencia, Clase, Asociación, Método	Boleano	Indica que es costoso de implementar.

IFDELETED	Falso	Referencia, Asociación	Boleano	Indica que todos los objetos clasificados con Delete dentro de la asociación deben ser borrados si el objeto referenciado o la asociación, respectivamente, son borrados.
INVISIBLE	Falso	Asociación, Propiedad, Método, Referencia, Clase	Boleano	Indica que el elemento sólo ha sido definido para propósito interno.
LARGE	Falso	Propiedad, Clase	Boleano	Indica que la propiedad o la clase necesitan una gran cantidad de espacio en memoria.
PROVIDER	NULL	Todos	Cadena	Indica el proveedor que obtiene o proporciona los datos para el elemento.
SYNTAX	NULL	Propiedad, Referencia, Método, Parámetro	Cadena	Tipo específico asignado a los datos del elemento.
SYNTAXTYPE	NULL	Propiedad, Referencia, Método, Parámetro	Cadena	Define el formato del clasificador Syntax.
TRIGGERTYPE	NULL	Clase, Propiedad, Método, Asociación, Indicación, Parámetro	Cadena	Indica bajo que circunstancias un trigger es disparado.
UNKNOWN VALUES	NULL	Propiedad	Array de Cadenas	Utilizado para indicar un conjunto de valores desconocidos para la propiedad.

UNSUPPORTED VALUES	NULL	Propiedad	Array de Cadenas	Utilizado para indicar un conjunto de valores no soportados para la propiedad.

De todos estos clasificadores opcionales que se definen, ninguno es utilizado en la versión actual del esquema CIM.

Por otro lado, el usuario puede definir cualquier clasificador adicional. De cualquier modo, DMTF recomienda que esto sea excepcional y que esté seguro que ningún otro clasificador (incluidos los opcionales) no cubren sus necesidades.

Anexo B

Representación de los clasificadores en CIM-OWL

Este anexo presenta de manera detallada la representación en CIM-OWL de los meta-clasificadores y los clasificadores estándar de CIM. Para ello describimos la representación en tres secciones agrupando los clasificadores aplicados a cada constructor del meta-modelo: Clases/Asociaciones, Propiedades y Referencias.

B.1. Clases/Asociaciones

ABSTRACT

Se define la clase QrAbstract para representar este clasificador: <owl:Class rdf:ID='QrAbstract'/>. La clase con el clasificador Abstract se define como subclase de QrAbstract. Esto se consigue utilizando la característica <rdfs:subClassOf rdf:resource='#QrAbstract'/>.

AGGREGATION

Se define la clase QrAggregation para representar este clasificador: <owl:Class rdf:ID='QrAggregation'/>. La asociación con el clasificador Aggregation se define como subclase de QrAggregation. Esto se hace utilizando la característica <rdfs:subClassOf rdf:resource='#QrAggregation' />.

ASSOCIATION

Se define la clase QrAssociation para representar este clasificador: <owl:Class rdf:ID='QrAssociation'/>. La clase con el clasificador Association, es decir una asociación, se define como subclase de QrAggregation. Esto se consigue utilizando la característica <rdfs:subClassOf rdf:resource='#QrAssociation' />.

DESCRIPTION

Esta es representada mediante <rdfs:comment>, de manera que el texto que forma la descripción de la clase se introduce en esta característica.

DEPRECATED

Esto se consigue con la característica <owl:deprecatedClass>e indicando con la anotación deprecated la clase propuesta para usarse en su lugar. Por ejemplo:

```
<owl:AnnotationProperty rdf:about='\&qr;deprecated'/>
<owl:deprecatedClass rdf:ID='CIM_CollectionInSystem'>
    <qr:deprecated>CIM_OwningCollectionElement</qr:deprecated>
</owl:deprecatedClass>
```

Este ejemplo indica que la clase CIM_CollectionInSystem no es recomendada y que la clase CIM_OwningCollectionElement debería usarse en su lugar.

DISPLAYNAME

Ésta es representada mediante <rdfs:label>. El texto contenido por este clasificador es introducido en esta característica.

EXPERIMENTAL

Se define la clase QrExperimental para representar este clasificador: <owl:Class rdf:ID='QrExperimental'/>. La clase con el clasificador Experimental se define como subclase de QrExperimental. Esto se consigue utilizando la característica <rdfs:subClassOf rdf:resource='#QrExperimental' />.

INDICATION

Se define la clase QrIndication para representar este clasificador: <owl:Class rdf:ID='QrIndication'/>. La clase con el clasificador Indication, es decir una indicación, se define como subclase de QrIndication. Esto se consigue utilizando la característica <rdfs:subClassOf rdf:resource='#QrIndication' />.

MAPPINGSTRINGS

Ésta es representada mediante <rdfs:seeAlso>. El texto contenido por este clasificador es introducido en esta característica.

REVISION

Ésta se consigue con la anotación revision. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;revision'/>

SOURCE

Ésta se consigue con la anotación source. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;source'/>

SOURCETYPE

Ésta se consigue con la anotación sourcetype. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;sourcetype'/>

TERMINAL

Se define la clase QrIndication para representar este clasificador: <owl:Class rdf:ID='QrTerminal'/>. La clase con el clasificador Terminal se define como subclase de QrTerminal. Esto se consigue utilizando la característica <rdfs:subClassOf rdf:resource='#QrTerminal' />.

VERSION

Ésta es representada mediante <owl:versionInfo>. El texto contenido por este clasificador es introducido en esta característica.

B.2. Propiedades

ALIAS

Ésta es representada mediante <owl:equivalentProperty>. De manera que se crea una propiedad equivalente para la propiedad.

ARRAYTYPE

Se define una propiedad por cada tipo de array, es decir se define QrBagArray, QrIndexedArray y QrOrderedArray. Así este clasificador se representada mediante una subpropiedad de alguna de estas propiedades, es decir mediante las característica <rdfs:subPropertyOf rdf:resource='#QrBagArray'/>, <rdfs:subPropertyOf rdf:resource='#QrIndexedArray'/>o <rdfs:subPropertyOf rdf:resource='#QrOrderedArray'/>.

BITMAP

Ésta se consigue con la anotación bitmap. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;bitmap'/>

BITVALUES

Ésta se consigue con la anotación bitvalues. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;bitvalues'/>

COUNTER

Se define la propiedad QrCounter para representar este clasificador: <rdf:Property rdf:ID='QrCounter'/>. La propiedad con el clasificador Counter se define como subpropiedad de QrCounter. Esto se consigue utilizando la característica <rdfs:subPropertyOf rdf:resource='#QrCounter'/>

DESCRIPTION

Al igual que el caso de las clases, esta es representada mediante <rdfs:comment>. De manera que el texto que forma la descripción de la propiedad se introduce en esta característica.

DEPRECATED

Ésta se consigue con las características <owl:deprecatedClass>o <owl:deprecatedProperty>.

DISPLAYNAME

Ésta es representada mediante <rdfs:label>. El texto contenido por este clasificador es introducido en esta característica.

DN

Se define la propiedad QrDN para representar este clasificador: <rdf:Property rdf:ID='QrDN'/>. La propiedad con el clasificador DN se define como subpropiedad de QrDN. Esto se consigue utilizando la característica <rdfs:subPropertyOf rdf:resource='#QrDN'/>

EMBEDDEDOBJECT

Se define la propiedad QrEmbeddedObject para representar este clasificador: <rdf:Property rdf:ID='QrEmbeddedObject'/>. La propiedad con el clasificador EmbeddedObject se define como subpropiedad de QrEmbeddedObject. Esto se consigue utilizando la característica <rdfs:subPropertyOf rdf:resource='#QrEmbeddedObject'/>

EXPERIMENTAL

Se define la propiedad QrExp para representar este clasificador: <rdf:Property rdf:ID='QrExp'/>. La propiedad con el clasificador Experimental se define como subpropiedad de QrExp. Esto se consigue utilizando la característica <rdfs:subPropertyOf rdf:resource='#QrExp'/>.

GAUGE

Se define la propiedad QrGauge para representar este clasificador: <rdf:Property rdf:ID='QrGauge'/>. La propiedad con el clasificador Gauge se define como subpropiedad de QrGauge. Esto se consigue utilizando la característica <rdfs:subPropertyOf rdf:resource='#QrGauge'/>

KEY

Se define la propiedad QrKey para representar este clasificador: <rdf:Property rdf:ID='QrKey'/>. La propiedad con el clasificador Key se define como subpropiedad de QrKey. Esto se consigue utilizando la característica <rdfs:subPropertyOf rdf:resource='#QrKey'/>.

MAPPINGSTRINGS

Al igual que el caso de las clases, ésta es representada mediante <rdfs:seeAlso>.

MAXLEN

El valor de este clasificador se utiliza para definir un nuevo tipo de dato que se le asignará a la propiedad. El nuevo tipo de dato será una cadena de texto restringida al tamaño indicado por el clasificador. Por ejemplo, para un valor de 255:

```
<xs:simpleType>
   <xs:restriction base='xs:string'>
      <xs:maxLength value='255'/>
   </xs:restriction>
</xs:simpleType>
```

MAXVALUE

El valor de este clasificador se utiliza para definir un nuevo tipo de dato que se le asignará a la propiedad. El nuevo tipo de dato será un entero con el valor máximo indicado por el clasificador. Por ejemplo, para un valor de 255:

```
<xs:simpleType>
   <xs:restriction base='xs:integer'>
      <xs:maxInclusive value='255'/>
   </xs:restriction>
</xs:simpleType>
```

MINVALUE

Del mismo modo que el clasificador anterior, el valor de este clasificador se utiliza para definir un nuevo tipo de dato que se le asignará a la propiedad. El nuevo tipo de dato será un entero con el valor mínimo indicado por el clasificador. Por ejemplo, para un valor de 5:

```
<xs:simpleType>
   <xs:restriction base='xs:integer'>
      <xs:minInclusive value='5'/>
   </xs:restriction>
</xs:simpleType>
```

MODELCORRESPONDENCE

Ésta se consigue con la anotación modelcorrespondence. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;modelcorrespondence'/>

NULLVALUE

Ésta se consigue con la anotación nullvalue. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;nullvalue'/>

OCTETSTRING

Se define la propiedad QrOctetString para representar este clasificador: <rdf:Property rdf:ID='QrOctetString'/>. La propiedad con el clasificador OctetString se define como subpropiedad de QrOctetString. Esto se consigue utilizando la característica <rdfs:subPropertyOf rdf:resource='#QrOctetString'/>

OVERRIDE

Ésta se consigue con la anotación override. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;override'/>

PROPAGATED

Ésta se consigue con la anotación propagated. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;propagated'/>

READ

Se define la propiedad QrRead para representar este clasificador: <rdf:Property rdf:ID='QrRead'/>. La propiedad con el clasificador Read se define como subpropiedad de QrRead. Esto se consigue utilizando la característica <rdfs:subPropertyOf rdf:resource='#QrRead'/>

REQUIRED

Se define la propiedad QrRequired para representar este clasificador: <rdf:Property rdf:ID='QrRequired'/>. La propiedad con el clasificador Required se define como subpropiedad de QrRequired. Esto se consigue utilizando la característica <rdfs:subPropertyOf rdf:resource='#QrRequired'/>

SCHEMA

Ésta se consigue con la anotación scheme. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;scheme'/>

STATIC

Se define la propiedad QrStatic para representar este clasificador: <rdf:Property rdf:ID='QrStatic'/>. La propiedad con el clasificador Static se define como subpropiedad de QrStatic. Esto se consigue utilizando la característica <rdfs:subPropertyOf rdf:resource='#QrStatic'/>

UNITS

Ésta se consigue con la anotación units. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;units'/>

VALUEMAP

Se define un nuevo tipo con los valores del clasificador. El nuevo tipo será indicado por la anotación valuemap. Esta anotación se define <owl:AnnotationProperty rdf:about='&qr;valuemap'/>. Un ejemplo es el siguiente:

```
<xs:simpleType name='CIM_Privilege.Activities.ValueMap'>
<xs:restriction base='xs:string'>
   <xs:enumeration value='1'/>
   <xs:enumeration value='2'/>
   <xs:enumeration value='3'/>
   <xs:enumeration value='4'/>
   <xs:enumeration value='5'/>
   <xs:enumeration value='6'/>
   <xs:enumeration value='7'/>
   <xs:enumeration value='..15999'/>
   <xs:enumeration value='16000..'/>
</xs:restriction>
</xs:simpleType>
```

VALUES

Se define un nuevo tipo con los valores del clasificador. El nuevo tipo será indicado por la anotación values. Esta anotación se define <owl:AnnotationProperty rdf:about='&qr;values'/>.Un ejemplo es el siguiente:

```
<xs:simpleType name='CIM_Privilege.Activities.Values'>
<xs:restriction base='xs:string'>
   <xs:enumeration value='Other'/>
   <xs:enumeration value='Create'/>
   <xs:enumeration value='Delete'/>
   <xs:enumeration value='Detect'/>
   <xs:enumeration value='Read'/>
   <xs:enumeration value='Write'/>
   <xs:enumeration value='Execute'/>
   <xs:enumeration value='DMTF_Reserved'/>
   <xs:enumeration value='Vendor_Reserved'/>
</xs:restriction>
</xs:simpleType>
```

WRITE

Se define la propiedad QrWrite para representar este clasificador: <rdf:Property rdf:ID='QrWrite'/>. La propiedad con el clasificador Write se define como subpropiedad de QrWrite. Esto se consigue utilizando la característica <rdfs:subPropertyOf rdf:resource='#QrWrite'/>

B.3. Referencias

Para las referencias hay clasificadores comunes con las propiedades. Éstos clasificadores comunes tendrán la misma representación, de manera que la siguiente tabla sólo indica la representación de los clasificadores no comunes.

AGGREGATE

Se define la propiedad QrAggregate para representar este clasificador: <rdf:Property rdf:ID='QrAggregate'/>. La propiedad con el clasificador Aggregate se define como subpropiedad de QrAggregate. Esto se consigue utilizando la característica <rdfs:subPropertyOf rdf:resource='#QrAggregate'/>

MAX

Ésta se consigue con la anotación max. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;max'/>

MIN

Ésta se consigue con la anotación min. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;min'/>

NONLOCAL

Ésta se consigue con la anotación nonlocal. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;nonlocal'/>

NONLOCALTYPE

Ésta se consigue con la anotación nonlocaltype. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;nonlocaltype'/>

SOURCE

Ésta se consigue con la anotación source. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;override'/>

SOURCETYPE

Ésta se consigue con la anotación source. Esta anotación se define como <owl:AnnotationProperty rdf:about='&qr;sourcetype'/>

WEAK

Se define la propiedad QrWeak para representar este clasificador: <rdf:Property rdf:ID='QrWeak'/>. La propiedad con el clasificador Aggregate se define como subpropiedad de QrWeak. Esto se consigue utilizando la característica <rdfs:subPropertyOf rdf:resource='#QrWeak'/>

Anexo C

Extensiones del esquema CIM

En este anexo detallamos las extensiones del esquema CIM que han sido necesarias para permitir la especificación de conceptos adicionales en la definición de políticas de seguridad con SWRL-CIM. Concretamente estas extensiones han sido dos que a continuación detallamos:

- Modificación de la definición de las asociaciones CIM_HostedFilterList y CIM_HostedFilterEntryBase del esquema CIM quitando la restricción de cardinalidad que no permite asociar un filtro o lista de filtros a más de una instancia de ComputerSystem. Así las asociaciones CIM_HostedFilterList y CIM_HostedFilterEntryBase se convierten en asociaciones muchos-a-muchos.

```
1  //===========================================
2  //  CIM_HostedFilterList
3  //===========================================
4  [Association, Version ( "2.7.0" ), Description (
5  "FilterLists are defined in the context of a ComputerSystem/ "
6  "network device, where the list is used and administered.")]
7  class CIM_HostedFilterList : CIM_HostedDependency {
8
9  [Override ( "Antecedent" ), Description (
10 "The ComputerSystem/network device that scopes the "
11 "FilterList.")]
12 CIM_ComputerSystem REF Antecedent;
13
14 [Override ( "Dependent" ), Description (
15 "The FilterList on the System.")]
16 CIM_FilterList REF Dependent;
17 };
18
19 //===========================================
20 //  CIM_HostedFilterEntryBase
21 //===========================================
```

```
[Association, Version ( "2.7.0" ), Description (
"All filter entries (FilterEntryBase and its subclasses) are "
"defined in the context of a ComputerSystem/network device, "
"where the filter entries are used and administered.")]
class CIM_HostedFilterEntryBase : CIM_HostedDependency {

[Override ( "Antecedent" ), Description (
"The ComputerSystem/network device that scopes the "
"FilterEntryBase.")]
CIM_ComputerSystem REF Antecedent;

[Override ( "Dependent" ), Description (
"The FilterEntryBase on the System.")]
CIM_FilterEntryBase REF Dependent;
};
```

- Extensión del esquema CIM para incluir la clase CIM_ClockDevice. Esta clase representa a un reloj que nos permite conocer la fecha y hora actuales mediante la propiedad DateTime de solo lectura, y también el día de la semana actual con la propiedad Day de solo lectura.

```
//===========================================
//  CIM_ClockDevice
//===========================================
[Version ( "2.12.1" )]
class CIM_ClockDevice : CIM_LogicalDevice {

[Read]
datetime DateTime;

[Read]
string Day;
};
```

Anexo D

Transformación XSL de reglas SWRL a reglas Jena-2

Este anexo presenta la transformación XSL de reglas SWRL representadas en XML a reglas Jena-2. Esta transformación permite que reglas definidas en SWRL puedan representarse en reglas Jena-2 y de esta manera utilizar el motor de razonamiento de Jena-2 sobre estas reglas.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0" xmlns:swrlx="http://www.w3.org/2003/11/swrlx"
xmlns:ruleml="http://www.w3.org/2003/11/ruleml"
xmlns:owlx="http://www.w3.org/2003/05/owl-xml">
   <xsl:output method="text"/>
   <xsl:template match="ruleml:imp">
    [<xsl:choose>
         <xsl:when test="ruleml:_rlab/@ruleml:href">
            <xsl:value-of select="ruleml:_rlab/@ruleml:href"/>
         </xsl:when>
         <xsl:otherwise>rule-<xsl:number/>
         </xsl:otherwise>
      </xsl:choose>: <xsl:text xml:space="preserve"> </xsl:text>
      <xsl:apply-templates select="ruleml:_body"/> ->
      <xsl:apply-templates select="ruleml:_head"/>]
  </xsl:template>
   <xsl:template match="ruleml:var">
      <xsl:text xml:space="preserve">
      </xsl:text>?<xsl:value-of select="."/>
      <xsl:text xml:space="preserve">
      </xsl:text>
   </xsl:template>
   <xsl:template match="owlx:Individual">
      <xsl:text xml:space="preserve"> </xsl:text>
      <xsl:value-of select="@owlx:name"/>
```

```
27        <xsl:text xml:space="preserve"> </xsl:text>
28     </xsl:template>
29     <xsl:template match="owlx:DataValue">
30        <xsl:text xml:space="preserve"> </xsl:text>
31        <xsl:choose>
32           <xsl:when test="@owlx:datatype=
33 ____'http://www.w3.org/2001/XMLSchema#decimal'
34 ____or_@owlx:datatype=
35 ____'http://www.w3.org/2001/XMLSchema#integer'
36 ____or_@owlx:datatype=
37 ____'http://www.w3.org/2001/XMLSchema#nonPositiveInteger'
38 ____or_@owlx:datatype=
39 ____'http://www.w3.org/2001/XMLSchema#negativeInteger'
40 ____or_@owlx:datatype=
41 ____'http://www.w3.org/2001/XMLSchema#long'
42 ____or_@owlx:datatype=
43 ____'http://www.w3.org/2001/XMLSchema#int'
44 ____or_@owlx:datatype=
45 ____'http://www.w3.org/2001/XMLSchema#short'
46 ____or_@owlx:datatype=
47 ____'http://www.w3.org/2001/XMLSchema#byte'
48 ____or_@owlx:datatype=
49 ____'http://www.w3.org/2001/XMLSchema#nonNegativeInteger'
50 ____or_@owlx:datatype=
51 ____'http://www.w3.org/2001/XMLSchema#unsignedLong'
52 ____or_@owlx:datatype=
53 ____'http://www.w3.org/2001/XMLSchema#unsignedInt'
54 ____or_@owlx:datatype=
55 ____'http://www.w3.org/2001/XMLSchema#unsignedShort'
56 ____or_@owlx:datatype=
57 ____'http://www.w3.org/2001/XMLSchema#unsignedByte'
58 ____or_@owlx:datatype=
59 ____'http://www.w3.org/2001/XMLSchema#positiveInteger'
60 ____or_@owlx:datatype=
61 ____'http://www.w3.org/2001/XMLSchema#float'
62 ____or_@owlx:datatype=
63 ____'http://www.w3.org/2001/XMLSchema#double'
64 _________">
65              <xsl:value-of select="."/>
66           </xsl:when>
67           <xsl:otherwise>
68           '<xsl:value-of_select="."/>'
69           </xsl:otherwise>
70        </xsl:choose>
71        <xsl:text xml:space="preserve"> </xsl:text>
72     </xsl:template>
73     <xsl:template match="swrlx:classAtom">(
74     <xsl:apply-templates select="*[2]"/>
75        http://www.w3.org/1999/02/22-rdf-syntax-ns#type
```

```
76        <xsl:value-of select="owlx:Class/@owlx:name"/> )
77    </xsl:template>
78    <xsl:template match="swrlx:datarangeAtom">isDType(
79    <xsl:apply-templates select="*[2]"/>
80        <xsl:value-of select="owlx:Datatype/@owlx:name"/> )
81    </xsl:template>
82    <xsl:template match="swrlx:individualPropertyAtom">(
83    <xsl:apply-templates select="*[1]"/>
84        <xsl:value-of select="@swrlx:property"/>
85        <xsl:apply-templates select="*[2]"/>)
86    </xsl:template>
87    <xsl:template match="swrlx:datavaluedPropertyAtom">(
88    <xsl:apply-templates select="*[1]"/>
89        <xsl:value-of select="@swrlx:property"/>
90        <xsl:apply-templates select="*[2]"/>)
91    </xsl:template>
92    <xsl:template match="swrlx:sameIndividualAtom">(
93    <xsl:apply-templates select="*[1]"/> owl:sameAs
94    <xsl:apply-templates select="*[2]"/>)
95    </xsl:template>
96    <xsl:template match="swrlx:differentIndividualsAtom">(
97    <xsl:apply-templates select="*[1]"/> owl:differentFrom
98    <xsl:apply-templates select="*[2]"/>)
99    </xsl:template>
100   <xsl:template match="swrlx:Ontology">
101       <xsl:apply-templates select="ruleml:imp"/>
102   </xsl:template>
103 </xsl:stylesheet>
```

Bibliografía

[1] F. Baader, D. Calvanese, D.L. McGuinness, D.Nardi, y P.F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Application.* Cambridge University Press, 2002.

[2] A. K. Bandara, E. C. Lupu, y A. Russo. Using Event Calculus to Formalise Policy Specification and Analysis. En *IEEE Policy 2003*, Lake Como, Italia, Junio 2003.

[3] A.K. Bandara. *A Formal Approach to Analysis and Refinement of Policies.* PhD thesis, Departament of Computing, Imperial College London, University of London, Julio 2005.

[4] S.M. Bellovin. Distributed firewalls. En *;login: Magazine*, páginas 37–39, Noviembre 1999.

[5] D. Box, F. Curbera, y M. Hondo. Web services policy framework (ws-policy). Disponible de forma on-line en la URL http://www-128.ibm.com/developerworks/library/specification/ws-polfram/, Marzo 2006. versión 1.1.

[6] J. Case, M. Fedor, M. Schoffstall, y J. Davin. *A Simple Network Management Protocol (SNMP).* IETF, Mayo 1990. Request For Comments (RFC) 1157.

[7] IBM Research Center. Commonrules rule-based framework. Disponible de forma on-line en la URL http://www.alphaworks.ibm.com/tech/commonrules, 2002.

[8] IBM Research Center. Authorization processing for globus toolkit java web services. Disponible de forma on-line en la URL http://www-128.ibm.com/developerworks/grid/library/gr-gt4auth/, 2005.

[9] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, y A. Smith. *COPS Usage for Policy Provisioning (COPS-PR).* IETF, Marzo 2001. Request For Comments (RFC) 3084.

[10] DAML-S Coalition. Owl-s: Semantic markup for web services. Disponible de forma on-line en la URL http://www.daml.org/services/owl-s/1.0/, 2003.

[11] N. Damianou, A. Bandara, M. Sloman, y E. Lupu. A Survey of Policy Specification Approaches. Technical report, Department of Computing, Imperial College of Science Technology and Medicine, 2002.

[12] N. Damianou, N. Dulay, E.C. Lupu, y M.S. Sloman. The Ponder Policy Specification Language. En *IEEE Policy 2001*, Bristol, UK, Enero 2001.

[13] N. Damianou, T. Tonouchi, N. Dulay, E.C. Lupu, y M.S. Sloman. Tools for Domain-based Policy Management of Distributed Systems. En *IEEE/IFIP NOMS 2002*, Florencia, Italia, Abril 2002.

[14] J.E. Lopez de Vergara, V.A. Villagra, J.I. Asensio, y J. Berrocal. Ontologies: Giving semantics to network management models. *IEEE Network*, 17(3):15–21, Junio 2003.

[15] Universidad de Murcia Departamento de Ingeniería de la Información y las Comunicaciones. Ore (ontology rule editor). Disponible de forma on-line en la URL http://sourceforge.net/projects/ore, 2006.

[16] University of Manchester Department of Computer Science. Fact owl-dl reasoner. Disponible de forma on-line en la URL http://owl.man.ac.uk/factplusplus/, 2006.

[17] University of Manchester Department of Computer Science. Hoolet owl-dl reasoner. Disponible de forma on-line en la URL http://owl.man.ac.uk/hoolet/, 2006.

[18] DMTF. Common information model (cim) standards. Disponible de forma on-line en la URL http://www.dmtf.org/standards/cim/.

[19] DMTF. Wbem initiative. Disponible de forma on-line en la URL http://www.dmtf.org/wbem/.

[20] DMTF. Ws-cim work group. Disponible sólo para miembros del DMTF de forma on-line en la URL http://www.dmtf.org/apps/org/workgroup/interop/ws-cim/.

[21] DMTF. Cim specification. Disponible de forma on-line en la URL http://www.dmtf.org/standards/documents/CIM/DSP0004.pdf, 1999.

[22] N. Dulay, E. Lupu, M. Sloman, y N. Damianou. A Policy Deployment Model for the Ponder Language. En *IEEE/IFIP IM 2001*, Washington, USA, Mayo 2001.

[23] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, y A. Sastry. *The COPS (Common Open Policy Service) Protocol.* IETF, Enero 2000. Request For Comments (RFC) 2748.

[24] Sandia National Laboratories E. Friedman-Hill. Jess, the rule engine for the javatm platform. Disponible de forma on-line en la URL http://herzberg.ca.sandia.gov/jess, 2006.

[25] ETRI. Bossam rule/owl reasoner. Disponible de forma on-line en la URL http://mknows.etri.re.kr/bossam/, 2006.

[26] D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, 2004.

[27] Jena Semantic Web Framework. Disponible de forma on-line en la URL http://jena.sourceforge.net/, 2006.

[28] F.J. García, O. Cánovas, G. Martínez, y A.F. Gómez Skarmeta. Self-configuration of grid nodes using a policy-based management architecture. En *APGAC Workshop, ICCS 2004*, Cracovia, Polonia, Junio 2004.

[29] F.J. García, J.D. Jiménez, G. Martínez, y A.F. Gómez Skarmeta. Policy-Driven Routing Management Using CIM. En *MMM-ACNS 2005*, San Petersburgo, Rusia, Septiembre 2005.

[30] F.J. García, G. López, J.D. Jiménez, G. Martínez, y A.F. Gómez Skarmeta. Deployment of a Policy-based Management System for the Dynamic Provision of IPsec-based VPNs in IPv6 Networks. En *IEEE SAINT 2005*, Trento, Italia, Febrero 2005.

[31] F.J. García, G. Martinez, J.A. Botía, y A.F. Gómez Skarmeta. On the Application of the Semantic Web Rule Language in the Definition of Policies for System Security Management. En *AWeSOMe Workshop, OnTheMove Workshops 2005*, Larnaca, Chipre, Octubre 2005.

[32] F.J. García, G. Martinez, J.A. Botía, y A.F. Gómez Skarmeta. Representing Security Policies in Web Information Systems. En *PM4W Workshop, WWW 2005*, Chiba, Japón, Mayo 2005.

[33] F.J. García, G. Martinez, J.A. Botía, y A.F. Gómez Skarmeta. *Web Semantics and Ontology*, capítulo Description of Policies Enriched by Semantics for Security Management. Idea Group Inc., 2006.

[34] F.J. García, G. Martinez, y A.F. Gómez Skarmeta. A Semantically-Rich Management System based on CIM for the OGSA Security Services. En *KDMG Workshop, AWIC 2005*, Lodz, Polonia, Junio 2005.

[35] F.J. García, G. Martinez, y A.F. Gómez Skarmeta. An XML-Seamless Policy Based Management Framework. En *MMM-ACNS 2005*, San Petersburgo, Rusia, Septiembre 2005.

[36] F.J. García, G. Martínez, O. Cánovas, y A.F. Gómez Skarmeta. A Proposal of a CIM-based Policy Management Model for the OGSA Security Architecture. En *GADA Workshop, OnTheMove Workshops 2004*, Larnaca, Chipre, Octubre 2004.

[37] F.J. García, G. Martínez, A. Muñoz, J.A. Botía, y A.F. Gómez Skarmeta. Distributed Provision and Management of Security Services in Globus Toolkit 4. En *GADA Workshop, OnTheMove Workshops 2006*, Montpellier, Francia, Noviembre 2006.

[38] F.J. García, A.F. Gómez Skameta, G. López, y G. Martínez. Secure VPNs over IPv6 Networks: An Evaluation and its Integration in a Policy Management Framework. *Journal of Internet Technology*, 5(2), 2004.

[39] Globus. Globus toolkit. Disponible de forma on-line en la URL http://www.globus.org/toolkit/, 2006.

[40] Globus. Gt security (gsi). Disponible de forma on-line en la URL http://www.globus.org/toolkit/security/, 2006.

[41] DL Implementation Group. Dig interface. Disponible de forma on-line en la URL http://dl.kr.org/dig/, 2006.

[42] T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[43] A. Guerrero, V.A. Villagrá, y J.E. López de Vergara. Ontology-based integration of management behaviour and information definitions using SWRL and OWL. En *IFIP/IEEE DSOM 2005*, Barcelona, España, Octubre 2005.

[44] D. Harrington, R. Presuhn, y B. Wijnen. *An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks.* IETF, Diciembre 2002. Request For Comments (RFC) 3411.

[45] IBM. Proposal for a cim mapping to wsdm. Disponible de forma on-line en la URL ftp://www6.software.ibm.com/software/developer/library/ws-wsdm.pdf, 2005.

[46] Microsoft IBM. Security in a web services world: A proposed architecture and roadmap. Disponible de forma on-line en la URL http://www-128.ibm.com/developerworks/library/specification/ws-secmap/, Abril 2002.

[47] IETF. The ietf network configuration working group (netconf). Disponible de forma on-line en la URL http://www.ietf.org/html.charters/netconfcharter.html.

[48] Maryland Information y Network Dynamics Lab Semantic Web Agents Project. Pellet. Disponible de forma on-line en la URL http://www.mindswap.org/2003/pellet/, 2006.

[49] Maryland Information y Network Dynamics Lab Semantic Web Agents Project. Swoop - a hypermedia-based featherweight owl ontology editor. Disponible de forma on-line en la URL http://www.mindswap.org/2004/SWOOP/, 2006.

[50] Maryland Information y Network Dynamics Lab Semantic Web Agents Project. Swoop+rules. Disponible de forma on-line en la URL http://www.mindswap.org/ edna/swoopRules/, 2006.

[51] The Rule Markup Initiative. Ruleml-powered policy specification and interchange. Disponible de forma on-line en la URL http://policy.ruleml.org, Marzo 2004.

[52] The Rule Markup Initiative. Swrl: A semantic web rule language combining owl and ruleml. Disponible de forma on-line en la URL http://www.ruleml.org/swrl/, Diciembre 2004. versión 0.6.

[53] The Rule Markup Initiative. Disponible de forma on-line en la URL http://www.ruleml.org/, 2006.

[54] ITU-T. *Information Technology — Open Systems Interconnection, Systems Management Overview*, 1992. ITU-T Rec. X.701.

[55] ITU-T. *Principles for a Telecommunications Management Network (TMN)*, 1996. ITU-T Rec. M.3010.

[56] M. Johnson, P. Chang, R. Jeffers, J.M. Bradshaw, V.W. Soo, M.R. Breedy, L. Bunch, S. Kulkarni, J. Lott, N. Suri, y A. Uszok. KAoS semantic policy and domain services: An application of DAML to Web services-based grid architectures. En *AAMAS 2003*, Melbourne, Australia, Julio 2003.

[57] L. Kagal. Rei : A Policy Language for the Me-Centric Project. Technical report, HP Labs, Septiembre 2002.

[58] L. Kagal, T. Finin, y A. Johshi. A Policy Language for Pervasive Computing Environment. En *IEEE Policy 2003*, Lake Como, Italia, Junio 2003.

[59] T. Klie y F. Strauss. Integrating SNMP Agents with XML-based Management Systems. *IEEE Communications Magazine*, Julio 2004.

[60] S. Kraus, M.Ñirkhe, y K. P. Sycara. Reaching agreements through argumentation: a logical model (preliminary report). En *12th International Workshop on Distributed Artificial Intelligence*, página 233–247, Hidden Valley, Pennsylvania, 1993.

[61] G. López, F.J. García, M. Gil, G. Martínez, y A.F. Gómez Skarmeta. Deploying Secure Cryptographic Services in Multi-domain IPv6 Networks. En *IEEE AINA 2005*, Taiwan, Marzo 2005.

[62] G. López, F.J. García, M. Gil, G. Martínez, y A.F. Gómez Skarmeta. Providing advanced authentication services in IPv6 multi-domain scenarios. *International Journal Internet Protocol Technology*, 1(2), 2005.

[63] E. Lupu y M. Sloman. Conflicts in Policy-Based Distributed Systems Management. *IEEE Transactions on Software Engineering*, Noviembre 1999.

[64] E.C. Lupu y M. Sloman. Conflict Analysis for Management Policies. En *IFIP/IEEE IM 1997*, San Diego, USA, 1997.

[65] E.C. Lupu y M. Sloman. Conflicts in policy-based distributed system management. *IEEE Transaction on Software Engineering*, 25(6), 2003.

[66] J.P. Martin-Flatin. *Web-Based Management of IP Networks and Systems.* Wiley, 2002.

[67] G. Martinez, F.J. García, J.A. Botía, y A.F. Gómez Skarmeta. Extending the Common Information Model for Incorporating Semantics and Ontology-based Reasoning in the Specification of Security Policies. En *HPOVUA'2005*, Oporto, Portugal, Julio 2005.

[68] G. Martinez, F.J. García, A. Muñoz, J.A. Botía, y A.F. Gómez Skarmeta. Enabling Conflict Detection using Ontology and Rule-Based Reasoning in the Specification of Security Policies. En *HPOVUA'2006*, Niza, Francia, Mayo 2006.

[69] G. Martínez, F.J. García, y A.F. Gómez Skarmeta. *Web and Information Security*, capítulo Policy-based Management of Web and Information Systems Security: an Emerging Technology. E. Ferrari and B. Thuraisingham (eds), Idea Group Inc., 2005.

[70] G. Martínez, G. López, F.J. García, y A.F. Gómez Skarmeta. Dynamic and Secure Management of VPNs in IPv6 Multi-Domain Scenarios. *Elsevier Computer Communications*, In Press., 2006.

[71] J. Moffett y M.S. Sloman. Policy Conflict Analysis in Distributed System Management. *Journal of Organisational Computing*, 4(1):1–22, 1994.

[72] OASIS. Extensible access control markup language (xacml). Disponible de forma on-line en la URL http://www.oasis-open.org/committees/xacml/.

[73] OASIS. Web services distributed management: Management using web services. Disponible de forma on-line en la URL http://www.oasis-open.org/committees/wsdm/.

[74] OASIS. Extensible access control markup language (xacml). Disponible de forma on-line en la URL http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, Diciembre 2004. versión 2.0.

[75] OASIS. Security assertion markup language (saml). Disponible de forma on-line en la URL http://www.oasis-open.org/committees/security/, Marzo 2005. versión 2.0.

[76] Object Management Group. *The Common Object Request Broker: Architecture and Specification (CORBA)*, Diciembre 1995. Versión 2.0.

[77] M. O'connor, H. Knublauch, S. Tu, y M. Musen. Writing rules for the semantic web using swrl and jess. En *8th International Protege Conference, Protege with Rules Workshop*, 2005.

[78] OSI. *Information Processing Systems—Open Systems Interconnection—Systems Management Overview*, 1991. ISO 10040.

[79] S. Parsons, C. Sierra, y N. R. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.

[80] S. D. Parsons y N. R. Jennings. Negotiation through argumentation-A preliminary report. En *Second International Conference Multi-Agent Systems (ICMAS'96)*, página 267–274, Kyoto, Japón, 1996.

[81] POSITIF. Policy-based security tools and framework. Disponible de forma on-line en la URL http://www.positif.org/, 2006.

[82] POSITIF. Positif project framework. Disponible de forma on-line en la URL http://positif.dif.um.es/, 2006.

[83] Polyander Research Project. Disponible de forma on-line en la URL http://www-dse.doc.ic.ac.uk/Projects/polyander/, 2006.

[84] S. Quirolgico, P. Assis, A. Westerinen, M. Baskey, y E. Stokes. Toward a Formal Common Information Model Ontology. En *WISE 2004 Workshops*, Philadelphia, USA, Octubre 2004.

[85] HP Labs Semantic Web Research. Jena: A semantic web framework for java. Disponible de forma on-line en la URL http://jena.sourceforge.net/, 2006.

[86] S. Ross-Talbot, S. Tabet, S. Chakravarthy, y G. Brown. A generalized RuleML-based Declarative Policy specification language for Web Services. En *W3C Workshop on Constraints and Capabilities for Web Services*, California, USA, Octubre 2004.

[87] Stanford University School of Medicine Stanford Medical Informatics. Protégé ontology editor. Disponible de forma on-line en la URL http://protege.stanford.edu/, 2006.

[88] J. Strassner. *Policy-Based Network Management: Solutions for the Next Generation*. Morgan Kaufmann, 2003.

[89] Racer Systems. Renamed abox and concept expression reasoner (racer). Disponible de forma on-line en la URL http://www.racer-systems.com/, 2006.

[90] G. Tonti, J. Bradshaw, R. Jeffers, R. Montanari, N.S. Suri, y A. Uszok. Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder. En *ISWC 2003*, Florida, USA, 2003.

[91] Stanford University. Jtp- java theorem prover. Disponible de forma on-line en la URL http://www.ksl.stanford.edu/software/JTP/.

[92] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P.Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, y J. Lott. KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction and Enforcement. En *IEEE Policy 2003*, Lake Como, Italia, Junio 2003.

[93] D. Verma. *Policy-Based Networking: Architecture and Algorithms*. Sams, 2000.

[94] D. Verma. Simplifying Network Administration using Policy based Management. *IEEE Network*, Marzo 2002.

[95] W3C. Xml path language (xpath). Disponible de forma on-line en la URL http://www.w3.org/TR/xpath, 1999.

[96] W3C. Xsl transformations (xslt). Disponible de forma on-line en la URL http://www.w3.org/TR/xslt, 1999.

[97] W3C. The ontology web language. Disponible de forma on-line en la URL http://www.w3.org/TR/owl-features/, 2004.

[98] W3C. Owl web ontology language semantics and abstract syntax. Disponible de forma on-line en la URL http://www.w3.org/TR/owl-semantics/, 2004.

[99] W3C. Rdql - a query language for rdf. Disponible de forma on-line en la URL http://www.w3.org/Submission/RDQL/, 2004.

[100] W3C. Writing rules for the semantic web using swrl and jess. Disponible de forma on-line en la URL http://www.mindswap.org/ edna/swoopRules/, 2004.

[101] W3C. Resource description framework (rdf). Disponible de forma on-line en la URL http://www.w3.org/RDF/, 2006.

[102] W3C. Sparql query language for rdf. Disponible de forma on-line en la URL http://www.w3.org/TR/rdf-sparql-query/, 2006.

[103] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, y S. Waldbusser. *Terminology for Policy-Based Management*. IETF, Noviembre 2001. Request For Comments (RFC) 3198.

[104] R. Yavatkar, D. Pendarakis, y R. Guerin. *A Framework for Policy-based Admission Control*. IETF, Enero 2000. Request For Comments (RFC) 2753.

[105] T. Ylonen. *The Secure Shell (SSH) Protocol Architecture*. IETF, Enero 2006. Request For Comments (RFC) 4251.

Listado de Acrónimos

CIM	Common Information Model
COPS	Common Open Policy Service
DL	Descriptive Logic
DMTF	Distributed Management Task Force
DTD	Document Type Definition
FOL	First Order Logic
IETF	Internet Engineering Task Force
ISO	Internacional Organization for Standarization
ITU	International Telecommunications Union
KPAT	KAoS Policy Admin Tool
MIB	Management Information Base
MOF	Manager Object Format
NMRG	Network Management Research Group
OMG	Object Management Group
OSI-SM	Open Systems Interconnection - Systems Management
OWL	Web Ontology Language
PBM	Policy-Based Management
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PMT	Policy Management Tool
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RuleML	Rule Markup Language
SAML	Security Assertion Markup Language
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SWRL	Semantic Web Rule Language
TMN	Telecommunications Management Network
WBEM	Web-Based Enterprise Management
WSDL	Web Service Definition Language
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language
XSD	XML Schema Definition
XSL	eXtensible Stylesheet Language