# ARQUITECTURA DE SERVICIOS DE GESTION DE POLÍTICAS EMPLEANDO PROTOCOLO COPS

Autor

Ing. Matteoda Ramiro

**Director**: Dr. Daniel Riesco
**Co-Director**: Dr. Gustavo Rossi

*Presentada a la Facultad de Informática de la Universidad Nacional de La Plata como parte de los requisitos para la obtención del título de Magíster en Ingeniería de Software*

Facultad de Informática
Universidad Nacional de La Plata

**Marzo de 2015**

Dedicatoria:

A mi Amor Carla, por su paciencia, apoyo y amor haciéndome feliz.

A mi Padre, hermano, sobrinas y familia,

A mi amigo Martin, de alguna manera especial esta presente en mis logros,

y dedicada en especial a mi Madre siempre presente en mi corazón.

## AGRADECIMIENTOS

Quiero agradecer especialmente a mi director de tesis, Dr. Daniel Riesco, por su predisposición, apoyo y contribuciones a lo largo del desarrollo de mi tesis. También agradezco al Dr. Gustavo Rossi por haber aceptado ser codirector y ayudar en su corrección.

Agradezco al Dr. Fernando Magnago por su constante ayuda y aliento para que realice la maestría e impulsarme a terminarla.

TABLA DE CONTENIDOS

RESUMEN ..... 8
ABSTRACT ..... 9
1. INTRODUCCION A LA TESIS ..... 10
1.1 Introducción ..... 10
1.2 Contribución del Trabajo ..... 11
1.3 Organización de la Tesis ..... 11
2. GESTION BASADA EN POLITICAS ..... 13
2.1 Introducción ..... 13
2.2 Políticas: definición ..... 13
2.2.1 Lenguajes de Definición de Políticas ..... 14
2.2.2 ¿Dónde se utilizan Políticas? ..... 18
2.3 Arquitectura de Gestión basada en políticas ..... 19
2.4 WS-Policy ..... 23
2.4.1 Aserciones de Política ..... 24
2.4.2 Expresión de Política ..... 25
2.5 Estado del Arte ..... 26
3. PROTOCOLO COPS (COMMON OPEN POLICY SERVICE) ..... 30
3.1 Introducción: ..... 30
3.2 Modelo básico ..... 31
3.3 El Protocolo: ..... 32
3.3.1 Objetos ..... 32
3.3.2 Mensajes: ..... 34
3.4 Esquema de Comunicación: ..... 35
3.4.1 Intercambio de Mensajes ..... 35
3.4.2 Seguridad: ..... 36
3.4.3 PEP Inicialización: ..... 37
3.4.4 Modo de Operación: ..... 37
3.4.5 Keep-Alive ..... 37
3.4.6 PEP/PDP Close ..... 37
3.4.7 Sincronización ..... 38
4. ARQUITECTURA ORIENTADA A SERVICIOS ..... 39
4.1 Introducción: ..... 39
4.2 ¿Qué es Arquitectura Orientada a Servicios? ..... 40
4.3 Elementos de una Arquitectura Orientada a Servicios: ..... 41
4.3.1 Servicios: ..... 41
4.3.2 Contrato de Servicios: ..... 42
4.3.2 Repositorio de Servicios: ..... 43
4.3.3 Proveedor de Servicios ..... 43
4.3.4 Consumidor de Servicios ..... 43
4.4 Principios de Arquitectura. ..... 44
4.4.1 Bajo Acoplamiento: ..... 44
4.4.2 Exposición funcional gruesa: ..... 45
4.4.3 Transparencia de Locación: ..... 46
4.4.4 Comunicación: ..... 46
4.4.5 Contrato de servicio estandarizado ..... 46
4.4.6 Orquestación y Coreografía de Servicios: ..... 46
4.4.7 Reuso de Servicios: ..... 46
4.4.8 Composición de Servicios: ..... 47

4.5 Bus de Servicios ..... 47
4.6 Web Services ..... 49
5. ARQUITECTURA PROPUESTA DE SERVICIOS GESTION DE POLÍTICAS ...53
5.1 Introducción ..... 53
5.2 Políticas en SOA ..... 53
5.2.1 Punto Final ..... 55
5.2.2 Políticas y Contratos ..... 55
5.2.3 Contexto de ejecución ..... 56
5.3 Modelo básico de gestión de políticas del IETF ..... 56
5.3.1 Extensión del modelo utilizando servicios. ..... 57
5.3.2 Inclusión de protocolo COPS en la arquitectura. ..... 60
5.4 Diseño orientado a objetos para implementar COPS ..... 63
5.4.1 Objetos ..... 63
5.4.2 Mensajes ..... 66
5.5 Implementación de Políticas de Calidad de Servicio sobre Web Services.....68
5.5.1 Modelo de calidad ..... 69
5.5.2 Implementación de la arquitectura propuesta ..... 70
6. CASO DE ESTUDIO: GESTION DE LICENCIAS DE SOFTWARE.....74
6.1 Introducción ..... 74
6.2 Contexto general de los sistemas de licencia ..... 74
6.3 Arquitectura de un sistema de gestión de licencia ..... 75
6.3.1 Proceso de obtención de licencia ..... 76
6.4 Modelo de gestión de licencias utilizando COPS sobre Web Services ..... 77
6.4.1 Escenario de obtención y validación de licencia ..... 78
7. TRABAJOS RELACIONADOS ..... 82
7.1. Introducción ..... 82
7.2 Utilización de protocolo COPS ..... 82
7.3 Gestión basada en Políticas ..... 83
7.4 Aplicaciones de gestión de licencias ..... 84
8. CONCLUSIONES Y FUTUROS TRABAJOS ..... 85
REFERENCIAS ..... 87
GLOSARIO ..... 91
ANEXO A: PROTOCOLO COPS ..... 92
A.1: El Protocolo ..... 92
A.1.1 Objetos ..... 92
A.1.2 Mensajes ..... 100
ANEXO B: Requerimientos de Calidad ..... 105
B.1: Requerimientos de Calidad ..... 105

INDICE DE FIGURAS

Figura 2.1 Arquitectura de gestión de políticas …… 21
Figura 3.1 Arquitectura COPS ..... 31
Figura 3.2 Conexión TCP con tres sesiones COPS con distintos tipos de clientes….. 35
Figura 3.3 Intercambio de mensajes entre el PEP y el PDP ……. 36
Figura 4.1 Componentes de una Arquitectura Orientada a Servicios …….. 44
Figura 4.2 Bus de Servicios …..…….. 48
Figura 4.3 Modelo básico de Web Service …….. 50
Figura 5.1 Modelos SOA ……. 54
Figura 5.2 componentes de una arquitectura orientada a servicio ……... 55
Figura 5.3 Modelo IETF extendido ……. 57
Figura 5.4 Arquitectura de Servicio de Gestión de Políticas …… 58
Figura 5.5 Enlace de servicios de políticas ……... 59
Figura 5.6 Gestión de políticas en bus de servicios …….. 60
Figura 5.7 Secuencia de mensajes COPS ……. 61
Figura 5.8 Estructura de objetos COPS …… 64
Figura 5.9. Encabezado (Common Hearder) de mensajes COPS …… 66
Figura 5.10. Calidad de uso ……. 69
Figura 5.11 Secuencia de mensajes QoS ……. 71
Figura 6.1 Arquitectura de gestion de licencias ..……. 77
Figura 6.2 Mensajes para intercambio de licencia……..……. 79

INDICE DE TABLAS

Tabla 3.1 Resumen objetos COPS ……. 33
Tabla 3.2 Resumen mensajes COPS …….. 34

RESUMEN

La calidad de servicio y la seguridad son aspectos fundamentales a tener en cuenta dentro de la gestión de recursos en de una red; automatizar la gestión es un objetivo buscado en sistemas distribuidos de gran escala a fin de optimizar el manejo de estos recursos. Una solución ampliamente aplicada es la gestión basada en políticas (PBN, Policy-Based Networking), la cual centra su atención en los usuarios y los servicios ofrecidos.

Se define políticas como la combinación de reglas y servicios donde las reglas definen los criterios para acceso y uso de los recursos [PCIM01]. El modelo de gestión PBN proporciona un modo específico de asignar recursos de red, calidad de servicio y seguridad, y generalmente cuenta con un servidor (Punto de Decisión de Políticas PDP) y uno o varios clientes (Punto de Ejecución de Políticas PEP).

Existen muchos beneficios para sistemas de gestión de redes basado en políticas, como por ejemplo, manejo de la seguridad independientemente del sistema o plataforma, gestionar recursos complejos en tiempo real, contar con diferentes configuraciones o asignación de acuerdo al cliente.

Durante los recientes años, PBN ha sido objeto de investigación como una solución al proceso de asignación de recursos en sistemas distribuidos. Han surgido muchas publicaciones en el área, la mayoría definen un modelo para la negociación de políticas utilizando un lenguaje de políticas propio no estandarizado que son aplicables a gestionar un único recurso, como por ejemplo calidad de servicio. Teniendo en cuenta un sistema distribuido, se han hallado pocos trabajos que planteen el uso de políticas al nivel de servicios de Web o arquitectura orientada a servicios.

El Internet Engineering Task Force [IETF] se ha encargado de desarrollar COPS [COPS01], un protocolo estándar para la distribución de políticas entre un PDP y un PEP, y a través de este define un modelo de referencia planteando una arquitectura basada en políticas con el objetivo de manejar recursos en una red.

Dado la necesidad de definir un modelo genérico de gestión de recursos en sistemas distribuidos mediante políticas, la solución que se describe en esta tesis propone una arquitectura de servicios de gestión de políticas utilizando COPS. El modelo planteado tiene como objetivo definir un esquema de negociación de políticas independientemente de la información que estas regulen, tomando como base la arquitectura definida por el IETF y extendiendo su protocolo de comunicación de políticas sobre una arquitectura orientada a servicios que permite lograr un bajo acoplamiento entre el servidor y el cliente. Aprovechando las ventajas del estándar COPS el diseño de políticas presentado es simple, extensible, permite manejar diferentes tipos de políticas sobre diferentes tipos de clientes y agregar nuevas políticas sin redefinir el modelo.

## ABSTRACT

The quality of service and security are key issues to consider in the management of resources within a network, automate the management is an aim sought by large-scale distributed systems to optimize the management of these resources. A widely applied solution is policy-based management (PBN, Policy-Based Networking), which focuses on users and services.

Policy is defined as the combination of rules and services where rules define the guidelines for access and use of resources [PCIM01]. PBN management model provides a way to assign specific network resources, service quality and security, and generally have a server (Policy Decision Point PDP) and one or more clients (Policy Enforcement Point PEP).

There are many benefits to network management systems based on policies, for example, management of security regardless of system or platform, manage complex resources in real time, have different configurations or resource allocation for each custumer.

During recent years, PBN has been investigated as a solution to the process of resource allocation in distributed systems. Have been many publications in the area, most defined a model for the negotiation of policies using a own policy language to manage a single resource, such as quality of service. Given a distributed system, few studies have been found to raise the political inside of Web service level or service-oriented architecture

The Internet Engineering Task Force [IETF] has been responsible for developing COPS [COPS01], a standard protocol for the distribution of policies between a PDP and PEP, and through that define a reference model to pose a policy-based architecture with objective to manage resources on a network.

Given the need to define a generic model of resource management in distributed systems through policies, the solution described in this thesis proposes a service architecture for policy management using COPS. The proposed model aims to define a policy framework for negotiation regardless of the information that they regulate, based on the architecture defined by the IETF and communication protocol extending its policy on service-oriented architecture that achieves low coupling between server and client. Taking advantage of the standard COPS, the policy design presented is simple, extensible, can handle different types of policies on different types of customers and add new policies without redefining the model.

# 1. INTRODUCCION A LA TESIS

## 1.1 Introducción

Actualmente existe una gran cantidad de dispositivos conectados a Internet, lo cual ha incrementado el uso de ancho de banda; asimismo cada día surgen nuevos servicios que se ofrecen a los usuarios para mejorar la experiencia y aprovechamiento en el uso de Internet, podemos citar por ejemplo las redes sociales, servicios de video conferencias, servicios de transmisión música o video en donde la calidad del servicio puede variar de acuerdo a lo que el usuario contrate, es por ello poder administrar de forma eficiente recursos o sistemas con una gran demanda es fundamental para su éxito. Similarmente se va incrementando diariamente la demanda de servicios de red, los cuales requieren de mayores recursos de los que actualmente proporcionan las redes. Esto ha marcado la pauta para el desarrollo de nuevas soluciones para el control de recursos y la gestión de sistemas distribuidos mediante tareas de monitoreo y control sobre los recursos y servicios que se ofrecen.

Un método prometedor para gestionar de manera eficiente la demanda de recursos o servicios de una gran cantidad de usuarios es utilizar políticas de acceso a los mismos mediante servicios de negociación de políticas. Se han elaborado algunas especificaciones donde se implementa la arquitectura de redes basadas en políticas (PBN – Policy Based Networks) para control de calidad de servicios QoS (Quality of Service).

El Internet Engineering Task Force [IETF] ha realizado esfuerzos por estandarizar esta arquitectura utilizando un servidor de políticas definido como Punto de Decisión de Políticas (PDP) y un cliente de aplicación de políticas definido como Punto de Ejecución de Políticas (PEP), centralizando el control de políticas en el servidor PDP. Dentro del modelo del IETF se propone [COPS01] como protocolo de negociación de políticas

COPS es un protocolo estándar para intercambiar información de políticas y decisiones, definido como un protocolo de consulta y respuesta. COPS fue diseñado para operar fiablemente y en tiempo real, con mínima sobrecarga, para proveer un control dedicado de políticas por el PEP. COPS fue originalmente diseñado para proveer control de políticas al protocolo de reservación de recursos RSVP, sin embargo, su diseño genérico posibilita el intercambio de otras decisiones de políticas.

Una de las falencias del modelo definido por el IETF para el intercambio de políticas es que el mismo esta centrado en una arquitectura cliente-servidor en donde el PEP tiene un fuerte acoplamiento a su servidor PDP, para mejorar esto se podría utilizar una Arquitectura Orientada a Servicios que tiene mucha aceptación en los sistemas actuales, donde el sistema entero puede ser considerado como una colección de servicios, cada uno teniendo su funcionalidad disponible a través de su interfaz, esos servicios pueden ser combinados para servir tareas dentro de una PBN y administrar de manera mas independiente los recursos o servicios mediante políticas.

Una de las áreas en las que más se está investigando y que se está destacando como la alternativa a los métodos tradicionales de gestión de recursos es la gestión basada en políticas que permita tomar decisión de gestión sobre los recursos del sistema dentro de una arquitectura orientada a servicios.

Durante los últimos años la gestión basada en políticas ha sido un área de investigación. Hay una variedad de publicaciones y presentaciones que, si bien son aplicaciones óptimas para el uso de políticas, proponen lenguajes propios de políticas que no abordan un modelo general, que propongan algún protocolo que soporte varios tipos de políticas como COPS. Muchos trabajos más afines con lo desarrollado de esta tesis, centran el uso de políticas en un recurso particular, sin la posibilidad de definir políticas genéricas independientemente de la aplicación que éstas regulen. Pocos se han encontrado orientados a una solución dentro de un sistema distribuido, la mayoría define un modelo básico cliente servidor.

El motivo de la investigación de esta tesis es analizar los estándares y modelos actuales de gestión basada en políticas y llegar a proponer una arquitectura de servicios de gestión de políticas, que sirva para negociar políticas de una manera independiente del tipo de recurso o servicio que se esté administrando.

Tomando como base la arquitectura de gestión de recursos mediante políticas definidas por el grupo de trabajo del IETF, que cuenta con un punto de aplicación de políticas y un punto de decisión de políticas como componentes principales de la arquitectura, donde la comunicación entre los puntos está basada en un modelo cliente/servidor, el objetivo principal de esta tesis es extender el modelo propuesto por el IETF a una arquitectura orientada a servicios para definir una arquitectura de servicios de gestión y negociación de políticas, empleando el protocolo COPS como medio de comunicación de políticas. Esta propuesta está basada en los estándares definidos por Internet Engineering Task Force [IETF].

A fin de mostrar un escenario de aplicación de la arquitectura propuesta, se prevé un caso de estudio que muestre la utilización de la solución propuesta.

## 1.2 Contribución del Trabajo

El siguiente artículo publicado es el resultado obtenido respecto del tema de la tesis:

- “POLICIES NEGOTIATION SERVICES BY THE COPS PROTOCOL ON WEB SERVICES”. Presentado en ICISTM08 (Second International Conference on Information System, Technology and Management), Marzo del 2008, Dubai.
N. Debnath, A. Arsaute, R. Matteoda, D. Riesco, G. Montejano.

## *1.3* Organización de la Tesis

La tesis se divide en cuatro grandes partes: Introducción, Estado del Arte, Contexto, Arquitectura propuesta. A continuación se detalla los capítulos que componen las restantes partes de lo que sigue a esta tesis.

*Capítulo 1 Introducción*: Breve introducción global, presentando los objetivos y motivación de la tesis.

*Capítulo 2 Gestión Basada en Políticas*: Se da una introducción a la gestión de políticas, la importancia del uso de políticas y se analizan algunos trabajos relacionados al tema de la tesis como análisis del estado del arte.

*Capítulo 3 Protocolo COPS:* Se da una descripción del protocolo COPS, explicando el modelo de comunicación, los mensajes y objetos que utiliza el protocolo. También se incluyen aspectos como seguridad, inicialización y secuencia de mensajes.

*Capítulo 4 Arquitectura Orientada a Servicios*: Presenta una reseña de la Arquitectura Orientada a Servicios, explicando los componentes y aspectos principales; además incluye una breve explicación de Web Services.

*Capítulo 5 Arquitectura Propuesta de Servicios de Gestión de Políticas*: Este es el capítulo principal de la tesis, se define un modelo para gestión de políticas sobre arquitecturas orientadas a servicios, utilizando protocolo COPS. Al final se explica la Implementación de Políticas de Calidad de Servicio sobre Web.

*Capítulo 6 Caso de Estudio: Gestión de Licencias de software* A fin de evaluar la arquitectura propuesta, se presenta un modelo para gestionar licencias utilizando servicios de políticas sobre Web Services

*Capítulo 7 Trabajos Relacionados* Revisar en este capítulo los trabajos mas influyentes que fueron de fundamental aporte para lograr el objetivo de esta tesis.

*Capítulo 8 Conclusiones y Futuros Trabajos:* Aquí se presentan las conclusiones finales y los futuros trabajos relacionados a la misma.

*Anexo A: Protocolo COPS – detalle:* En este anexo se describe en detalle el formato y contenido de cada uno de los mensajes y objetos que componen el protocolo COPS según la norma [COPS01].

*Anexo B: Requerimientos de Calidad – detalle*: En este anexo se describe los requerimientos de calidad de software tales como performance, seguridad, disponibilidad tienen gran influencia en la arquitectura de software de un sistema.

# 2. GESTION BASADA EN POLITICAS

## *2.1* Introducción

La diversidad de sistemas y servicios ofrecidos a través de una red y varios tipos de clientes accediendo a usar los recursos disponibles lleva a buscar soluciones optimas que permitan gestionar los mismos, como por ejemplo el ancho de banda de Internet ofrecido para cada cliente particular o posibilidad de limitar las funcionalidades de un servicio ofrecido. El nodo de control o servidor de acceso debe aplicar las reglas definidas como manera de gestionar los recursos de la red y negociar con los sistemas o aplicaciones que acceden a ellos. Un método prometedor para resolver este dilema es utilizar políticas de control o gestión.

Actualmente se ha observado un incremento en la demanda de servicios de red, los cuales requieren de mayores recursos de los que actualmente proporcionan las redes. Esto ha marcado la pauta para el desarrollo de nuevas soluciones para el control de recursos, como es el caso de los servicios de negociación de políticas. Se han elaborado algunas especificaciones donde implementa la arquitectura de redes basadas en políticas (PBN – Policy Based Networks) para control de QoS (Quality of Service) de servicios, esto con el fin de ofrecer calidad de servicio extremo a extremo.

La Gestión basada en Políticas se centra en los usuarios y en los servicios y no en la interfaz y los dispositivos, permitiendo por ejemplo la creación de usuarios asociados a cierto rol de modo que puedan acceder o no a un determinado recurso.

Separar las políticas de la implementación de un sistema permite a las políticas ser modificadas dinámicamente a fin de cambiar la estrategia para manejar el sistema y así modificar el comportamiento del sistema sin cambiar su implementación [PDM94].

## *2.2* Políticas*:* definición

*Política*: "la combinación de reglas y servicios donde las reglas definen los criterios para acceso y uso de los recursos" [PCIM01].

En un sentido general, una política es una o más reglas que describen la acción que se produce cuando se da una determinada condición, pudiendo existir reglas y políticas concatenadas.

También se define como un método de acción para determinar presentes y futuras decisiones. Las políticas son implementadas o ejecutadas dentro de un contexto particular (tal como políticas definidas dentro de una unidad de negocio). [TPBM01].

Dentro de las definiciones que se encuentran de políticas se pueden encontrar los siguientes puntos en común:

- Es una regla que gobierna el comportamiento de una entidad

- El conjunto de políticas esta bien definido mediante un modelo común
- Cada regla define un alcance, un mecanismo y acciones.

Un ejemplo de una política de red podría ser: "si los recursos de la red están por debajo de un cierto nivel aceptable, entonces restringir el trafico WWW".

El alcance de esta política es la red, y mas precisamente el trafico WWW. El mecanismo es la asignación de ancho de banda, y la acción es limitar los recursos de la red usados por el trafico WWW.

Una política es representada en un repositorio como un conjunto de reglas. La sintaxis y semántica de la representación de la política en el repositorio necesita estar estandarizada para permitir interoperabilidad entre diferentes vendedores.

Para almacenamiento de políticas se puede usar un repositorio que implemente el Protocolo Ligero de Acceso a Directorios (LDAP), la descripción de políticas a almacenar en el directorio esta en el esquema del repositorio. El esquema define el tipo de entrada que puede ser almacenado en el repositorio, así como el número de atributos en cada tipo de entrada, y las relaciones entre los diferentes tipos de entradas.

El IETF en [PCLD04] define la implementación de LDAP como protocolo de acceso al repositorio de políticas. El esquema usado para las políticas es propuesto por el IETF en [PCELD05]. Este consiste de tres tipos de entradas:

- Políticas: esta clase representa la semántica asociada con una política "si condición entonces acción"
- Condiciones: determinan si se debe o no aplicar una política.
- Acciones: ejecutan una o mas operaciones que afectaran el servicio

El [IETF] cuenta con un grupo especializado (Policy Framework Working Group) que trabaja continuamente en la definición de lenguaje de definición de políticas, una arquitectura de políticas, terminología de políticas y un modelo de política. Como por ejemplo la definición de un esquema especial para políticas de calidad de servicio.

Un ejemplo de uso políticas es el acuerdo de nivel de servicio (ANS) que define un contrato entre un proveedor de servicio y un cliente a fin de fijar la calidad de dicho servicio. El ANS incluye los objetivos de nivel de servicio (ONS), los cuales se utilizan para especificar los servicios que la red proporcionara a sus clientes. Algunos ejemplos de ONS incluyen performance del servicio, disponibilidad, tiempo de respuesta de la aplicación, etc. Por lo tanto, las políticas del servicio expresan como será utilizada la red por parte de los usuarios, las aplicaciones.

Las políticas también pueden definir las reglas de limitación del ancho de banda para acceso a Internet por parte de los usuarios según el servicio.

### 2.2.1 Lenguajes de Definición de Políticas

Los lenguajes de políticas permiten especificar reglas que son independientes de las entidades que implementan las políticas.

Los lenguajes para políticas definen y aplican las políticas en la red, para ello utilizan un gestor que controla los recursos de la red mediante un conjunto de políticas predefinidas. Especifican reglas que son independientes de las entidades que las implementan. Estos lenguajes se definen en términos de métricas de ejecución y permiten la definición de eventos y condiciones de red.

Varios lenguajes han sido concebidos para permitir la especificación de reglas de políticas. Existen diversos lenguajes para la gestión basada en políticas, algunos de ellos dan mayor énfasis a la seguridad, otros a la gestión de recursos, etc.

- Ponder del Imperial College of Science en Inglaterra. [PNDR00]
- Policy Description Language (PDL) [PLB99]
- Security Policy Language (SPL) del Institute Mageland en Portugal. [SPL99]
- El eXtensible Access Control Language (XACML) [OASIS05]
- Enterprise Privacy Authorization Language (EPAL) de IBM [EIBM03]
- Platform for Privacy Preferences (P3P) de W3C [P3P007]

A continuación se da una breve descripción de cada uno, para obtener información mas detallada se puede ver las referencias correspondiente.

**Ponder** [PNDR00] del Imperial College of Science en Inglaterra es uno de los lenguajes de políticas de libre distribución más antiguos y avanzados .Es un lenguaje declarativo orientado a objetos para especificar políticas de gestión y de seguridad para sistemas de objetos distribuidos. El lenguaje permite cubrir el amplio rango de los requerimientos actuales que presentan los paradigmas de los sistemas distribuidos.

Ponder ser describe como un lenguaje declarativo, orientado a objetos para especificar políticas de seguridad y gestión de sistemas distribuidos. Las políticas que Ponder soporta incluyen: Políticas básicas, compuestas y meta-políticas.

Las políticas básicas pueden ser:

- Políticas de autorización: Definen control de acceso y uso de privilegios sobre los objetos destino
- Políticas de obligación: especifican políticas que deben ser ejecutadas sobre determinados objetos cuando ocurren eventos específicos.
- Políticas por delegación. Especifican acciones que un usuario puede delegar a otros

Las políticas compuestas constan de:

- Grupos. Son políticas y restricciones que tienen relaciones semánticas similares y deben utilizarse conjuntamente
- Roles. Especifican derechos y obligaciones.
- Relaciones. Entre políticas que definen roles.
- Tipo de especialización. Permiten la extensión a nuevas definiciones de políticas utilizando la herencia

Las Meta políticas especifican políticas acerca de las políticas que son parte de una política compuesta y son usadas para definir restricciones específicas de la aplicación.

La desventaja de Ponder es que el nivel de complejidad es muy alto debido a que tiene un gran número de constructores. Además el compilador del Ponder solamente genera las reglas de implementación, es decir, dichas reglas aún necesitan ser escritas manualmente en código de aplicación específico.

**PDL** Policy Description Language [PLB99] es un lenguaje de políticas que se basa en un paradigma de evento-condición-acción para definir las políticas como reglas que son generadas cuando un evento ocurre y si la condición es verdadera la acción es ejecutada.

Las políticas en PDL son descriptas por una colección de proposiciones de dos tipos, proposiciones de reglas de políticas y proposiciones de eventos de políticas. Las proposiciones de reglas de políticas son expresiones de la forma: "*evento causa acción si condición* (si el evento ocurre en una situación donde la condición es verdadera entonces la acción será ejecutada".

Este lenguaje se utiliza para políticas operacionales. PDL no distingue entre políticas de autorización y políticas de obligación. Las políticas hechas en PDL son por una parte políticas de obligación, debido a que especifican explícitamente las acciones ejecutadas después de la recepción de un evento.

Una de las desventajas del lenguaje PDL es que no permite la composición de políticas dentro de roles, o otros grupos de estructuras. Tampoco permite definir nuevos eventos debido a que los eventos se reciben a partir de los agentes previamente definidos. PDL no contempla un lenguaje de descripción de eventos.

**SPL** [SDL99] Security Policy Language es un lenguaje de seguridad diseñado para expresar políticas que ayuden a decidir acerca de la aceptabilidad de eventos. La aceptabilidad de cada uno de los eventos depende de las propiedades de los eventos (por ejemplo: autor, destino, acción, etc.), en el contexto específico en el que suceden los eventos y depende también de las propiedades de los eventos pasados. Este lenguaje puede ser implementado por medio de un monitor de eventos.

SPL esta compuesto de cuatro bloques básicos: entidades, conjuntos, reglas y políticas. Las reglas son el componente principal del lenguaje. Las reglas expresan restricciones en términos de relaciones entre entidades y conjuntos. Una política SPL se compone de conjuntos y reglas. Los conjuntos contienen las entidades que utilizan las políticas para decidir acerca de la aceptabilidad de los eventos. Las reglas son funciones de los eventos que pueden tener tres tipos de valores: negar, permitir y no aplicar. Cada política tiene una regla especial llamada "regla de consulta" (identificada por el signo interrogación) la cual es una composición de otras reglas y define el comportamiento de la política. La delegación se logra cuando se hace una instancia a una política y se utiliza como una regla en la composición de otras reglas.

Las desventajas del SPL son que a pesar de que SPL define un modelo de restricciones, dicho modelo fue diseñado para que lo implementara un monitor de eventos y no es adecuado para utilizarse en todas las plataformas de gestión.

**XACML [**OASIS05**]** es un lenguaje de políticas de control de acceso de propósito general usado para describir políticas y decisiones de control de acceso en un modelo solicitud/response. Es un estándar OASIS escrito en XML diseñado para soportar gestión de políticas tanto centralizada como descentralizada. describe tanto un lenguaje de políticas como un protocolo petición/respuesta para el control de las decisiones de autorización. Ambos, lenguaje y protocolo, están definidos en XML. El lenguaje de políticas XACML se utiliza para describir requisitos de control de acceso generales, y tiene unos puntos de extensión estándar que permiten la definición de nuevas funciones, tipos de datos, combinaciones lógicas, etc. El protocolo petición/respuesta nos permite componer una petición para solicitar si cierta acción debería, o no, ser permitida, además de ofrecernos formas de interpretar las decisiones de control. La respuesta siempre incluye una contestación que refleja si la petición debería ser o no permitida utilizando uno de los siguientes cuatro valores: Permitir, Denegar, Indeterminado (no se pudo tomar una decisión porque ocurrió un error o fue omitido algún valor requerido) o No Aplicable (la petición no puede ser contestada por este servicio). Estos valores se encuentran predefinidos por la propia especificación

XACML define tres elementos básicos: reglas, políticas y conjunto de políticas. Las reglas contienen una expresión lógica; las políticas contienen un conjunto de reglas y un procedimiento para combinar el resultado de su evaluación; son la unidad básica de política usada por el punto de decisión de políticas (PDP) y forman la base para la decisión de autorización. Los conjuntos de políticas están compuestos por políticas y algún otro conjunto de políticas y poseen procedimientos para combinar los resultados de su evaluación; son el estándar para combinar políticas separadas dentro de una sola política.

Dentro de la arquitectura XACML se definen:

- Punto de Administración de Políticas (PAP): punto donde se manejan las políticas, se administra la información utilizada para definirlas.
- Punto de Decisión de Políticas (PDP): punto que evalúa y hace publica las decisiones de autorización.
- Punto de Aplicación de Políticas (PEP): punto que intercepta el requerimiento de acceso a un servicio por parte del usuario y aplica la decisión del PDP. El PEP puede ser parte de un servidor Web, un punto de acceso a la red, un agente de mail.
- Punto de Información de Políticas (PIP): punto que puede proveer información externa al PDP.

XACML posee un conjunto de algoritmos de combinación que definen un procedimiento para arribar a una decisión de autorización dado el resultado de evaluar un conjunto de reglas o políticas. Como ejemplo se puede definir: “si una regla o política dentro de un conjunto es evaluada como *permitir acceso*, entonces decidir permitir acceso”.

Existen muchos lenguajes específicos de aplicación o propietarios que ya realizan las mismas acciones, aunque XACML posee varios puntos a su favor. Es un estándar, ya que utiliza lenguaje XML; es genérico, permite proporcionar control de acceso dentro de cualquier entorno; es distribuido, esto significa que una política puede ser escrita de forma que puede referenciar a otras políticas ubicadas en localizaciones arbitrarias.

**EPAL** Enterprise Privacy Authorization Language [EIBM03] introducido por IBM para describir políticas formalmente, es un lenguaje para intercambio de políticas de privacidad entre aplicaciones o empresas. EPAL contiene políticas de privacidad y reglas de privacidad, provee los elementos necesarios que son usados para construir la política.

Una política en EPAL es esencialmente una lista de reglas de privacidad que son ordenadas en precedencia descendiente (si una regla se aplica, la subsiguiente es ignorada). La definición de una regla incluye decisión, una categoría de usuario, una acción, una categoría de datos, un propósito y también pueden contener condiciones y obligaciones. Las reglas son usadas para determinar si una solicitud es permitida o denegada.

Un lenguaje de políticas debe ser cerrado en cuanto a su combinación de políticas, es decir, que este puede expresar la lógica de combinación de dos cualquiera de sus políticas y EPAL como muchos otros no es cerrado en cuanto a su combinación, lo cual representa una desventaja.

Hay ciertas situaciones para las cuales EPAL no es un lenguaje apropiado, como por ejemplo la limitación de características en EPAL para conformar reglas de políticas sobre un producto preexistente o herramienta.

**P3P** Platform for Privacy Preferentes [P3P07] posibilita a los sitios Web expresar sus políticas de privacidad en un formato estándar que puede ser recuperado automáticamente e interpretado fácilmente por los agentes de usuarios.

En P3P los agentes de usuarios informaran a los usuarios de practicas del sitio y tomaran decisiones automáticas en base a esas practicas cuando sea necesario, así los usuarios no necesitan leer las políticas de privacidad cada vez que ingresan a cada sitio.

Idealmente un lenguaje de políticas de privacidad debería forzar consistencia, garantizar seguridad, permitir razonamiento local, permitir combinación de políticas y ser extensible. Sin embargo, esas metas no pueden estar presentes simultáneamente en todos, diseñar lenguajes que provean todas esas condiciones es difícil. Uno de los problemas abiertos que intentan resolver los lenguajes de políticas es la detección y resolución de conflictos entre políticas ya definidas.

Los lenguajes no están ligados a una base de datos en particular, se sugiere la utilización de un directorio LDAP para almacenar las políticas. Es fundamental al elegir un lenguaje tener en cuenta la aplicación que se le quiere dar, los requerimientos fundamentales, la posibilidad que provee de definir una amplia variada de políticas en forma clara y extensible a cualquier tipo.

### 2.2.2 ¿Dónde se utilizan Políticas?

Aunque actualmente las políticas suelen implementarse en routers y conmutadores, es posible hacer que las políticas residan en módulos de software o en servidores dedicados específicamente a esta tarea. Estos elementos se encargan de recibir las peticiones de tráfico solicitadas por los conmutadores u otros nodos, recabar información de políticas de bases de datos y directorios, y, en consecuencia, configurar

los dispositivos de red de acuerdo con las peticiones recibidas y los derechos establecidos en la especificación de nivel de servicio acordada entre el usuario y el proveedor de servicio

El control de calidad de servicio se puede ver como el acto de manejar asignación de recursos basado en políticas administrativas y recursos disponibles. Cuando se planea el control de calidad de servicio a través de políticas es importante revisar, clasificar y entonces priorizar los servicios que provee la red, a partir de esto se definen las políticas para cada servicio. Las políticas administrativas son cargadas en una base de datos de políticas y controladas por el servidor de políticas

Dentro de una red, las políticas, proporcionan las guías para especificar como los diferentes elementos de red, como por ejemplo routers, servidores, switches, deberían manejar el tráfico generado por los diferentes usuarios y aplicaciones.

Conforme las redes inalámbricas se convierten más complejas, nuevos métodos de control jerárquico serán necesarios. Un método para resolver esto es utilizar políticas de control o gestión en lugar de configuración de parámetros de servicios y dispositivos concretos.

El 3GPP ha elaborado algunas especificaciones donde implementa la arquitectura de gestión de redes basada en políticas (PBN) para control de calidad de servicio a servicios UMTS IMS [3GPP05], esto con el fin de ofrecer calidad de servicio extremo a extremo en el dominio de UMTS.

La gestión automática ha sido aceptada ampliamente como una solución en sistemas distribuidos de gran escala y esta siendo implementado en muchas áreas de computación. De muchas de las técnicas empleadas en gestión automática, la basada en políticas es la más popular porque esta es conceptualmente simple, fácil de implementar y ha sido ampliamente aplicado [PDPI06].

Sistemas actuales basados en políticas pueden exhibir gestión automática conteniendo políticas que son estáticas en tiempo de ejecución, por ejemplo, esta justo puede cargar un conjunto de parámetros vinculados al comportamiento. En tales aproximaciones se presentan problemas cuando se cambian las políticas.

## 2.3 Arquitectura de Gestión basada en políticas

La gestión basada en políticas (PBN, Policy-Based Networking) es un cambio importante en la forma en la que se ha venido concibiendo la gestión de red. En lugar de concentrarse en los dispositivos e interfaz de red o en los sistemas distribuidos; centra su atención en los usuarios y los servicios ofrecidos. Durante los últimos años ha sido objeto de estudio como alternativa para el proceso de gestión de redes.

El modelo de gestión PBN proporciona un modo específico de asignar recursos de red, calidad de servicio y seguridad, considerando un conjunto de políticas previamente definido. Estos sistemas proporcionan escalabilidad y facilidad para adaptarse a los cambios de las redes en tiempo real. PBN se propone capturar y definir de manera formal las reglas de negocio en forma de requisitos al más alto nivel y establecer un proceso determinista de traducción de estos requisitos a políticas específicas que liguen

las necesidades del negocio con un comportamiento coherente de la red y de los sistemas distribuidos, definido a alto nivel.

El sistema de gestión basado en políticas se encargará entonces del proceso de transformación de éstas políticas en representaciones operacionales aptas para ser interpretadas por los dispositivos de red y cuyos modelos de agentes de gestión que operan en información se pueden corresponder a cualquiera de las arquitecturas de gestión tradicionales. La gestión basada en políticas hace referencia a la agrupación de recursos basada en las políticas que determinan cómo interactuarán entre sí.

La configuración basada en políticas es una técnica usada porque esta posibilita que la configuración del sistema sea especificada externamente del mecanismo de implementación. Las políticas pueden ser examinadas por expertos y modificadas, sin cambiar funcionalidades del sistema.

Un sistema de gestión basado en políticas debe cumplir con algunos requisitos, a continuación se comentan los principales:

- Un modelo de información extensible tanto para todos los elementos de red, como para los servicios, redes y clientes.
- Un lenguaje de especificación de políticas capaz de representar los requisitos de negocios de manera formal e independiente tanto de dispositivos como de marcas.
- Medios para detectar y resolver conflictos entre políticas.
- Un modelo escalable de administración, gestión y distribución de políticas.

El Internet Engineering Task Force ha realizado esfuerzos por estandarizar esta arquitectura y ha propuesto un sistema donde define una serie de entidades y las funciones que deben llevar a cabo [YAV00]. A su vez también desarrollo un protocolo [COPS01] para la distribución de políticas y un modelo orientado a objetos para su representación [PCIM01].

El Internet Engineering Task Force está trabajando en cómo representar, administrar y compartir políticas y en cómo representar las reglas de políticas, para que sean independientes de los fabricantes y permitan una eficiente gestión.

El grupo de trabajo del IETF define un modelo de referencia [YAV00] de una arquitectura basada en políticas dirigida a la gestión de la red, especialmente la calidad de servicio y el modelo de información básico para las políticas. La **Figura 2.1** muestra la arquitectura de políticas definida por el IETF.

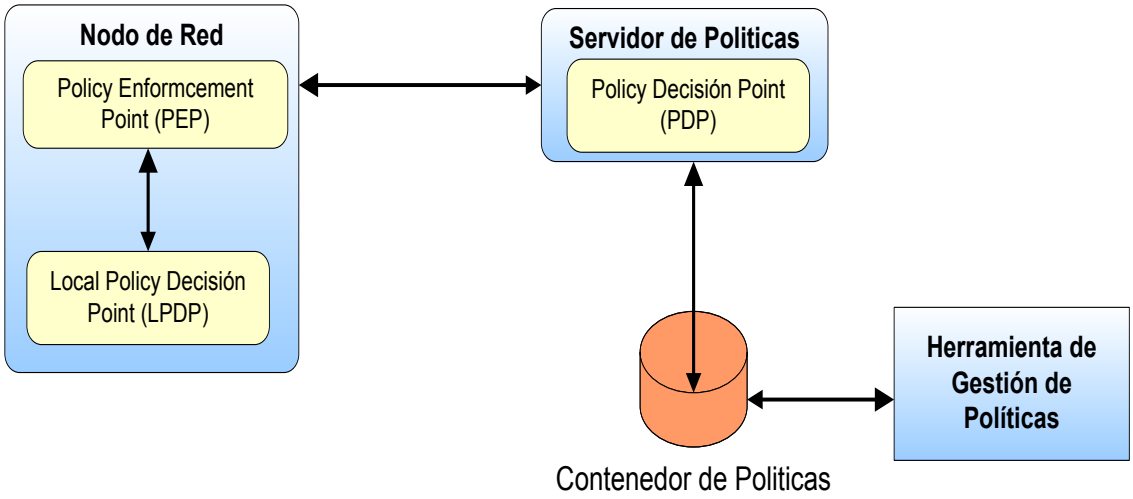



Figura 2.1: *Arquitectura de gestión de políticas*.

Los dos principales elementos para el control de políticas son el Punto de Ejecución de Políticas (PEP) y el Punto de Decisión de Políticas (PDP).

*Punto de Ejecución de Políticas (PEP):* Representa la entidad en la cual las políticas son aplicadas o ejecutadas. Este componente puede estar embebido en un dispositivo o ser implementado en un servidor Web.

Cuando ocurre un evento en la red, el cual requiere la implicación de una política. El PEP envía la petición de este evento al PDP donde se procesa y se obtiene una respuesta que será enviada al PEP que es el que actuará.

El PEP se encarga de ejecutar las políticas de acuerdo a la solicitud del usuario, las decisiones obtenidas del PDP y la información almacenada localmente. La ejecución de una política va a permitir gestionar los recursos de red, logrando por ejemplo cambiar su configuración para conseguir calidad de servicio (QoS), seguridad, etc.

El modulo PEP debe usar los siguientes pasos para cada política de decisión:

1. Cuando un evento local o mensaje invoca al PEP para una política de decisión, este debe crear un requerimiento que incluye información desde el mensaje que describa la solicitud de control de admisión.
2. El PEP puede consultar a una base de datos local para identificar un conjunto de elementos de políticas (llamémoslo A) que podrían ser evaluadas localmente. La configuración local especifica el tipo de políticas que son evaluadas localmente. El PEP pasa el requerimiento con el conjunto A al punto de decisión local LPDP y agrupa el resultado del LPDP.
3. El PEP pasa entonces el requerimiento con todos los elementos de políticas al PDP. El PDP aplica las políticas basado en toda la información recibida y genera una decisión.
4. El PDP retorna la decisión final al PEP.
5. El PEP ejecuta la acción correspondiente de acuerdo a la información recibida del PDP.

El PDP también puede, en cualquier momento, enviar una notificación asincrónica al PEP para cambiar una decisión previa o para generar mensajes de error.

*Punto de Decisión de Políticas (PDP):* es la entidad responsable de generar decisiones de políticas por si mismo o de acuerdo a los requerimientos enviados por el PEP y los elementos de política asociados. Realiza la toma de decisiones comparando el estado actual de la red con el estado deseado (establecido entre el cliente y el proveedor) y estudiar la forma de cómo conseguirlo. El PDP, teniendo en cuenta el origen de la petición, obtiene todas las políticas del contenedor de políticas y genera una decisión que será comunicada al PEP. El PDP puede enviar en cualquier momento una decisión asíncrona cambiando una política. Básicamente, la funcionalidad de crear mensajes de error o alertas y poder propagarlos usando algún protocolo nativo es necesaria. El PDP también puede enviar decisión o mensajes en forma masiva a todos los PEPs conectados a su red.

El PDP puede implementarse en diversos lugares; puede estar junto al servidor de políticas, ser implementado en un elemento independiente que solo hiciese esta función o implementarlo junto al PEP.

*Punto de Decisión de Políticas Local (LPDP):* Cuando el cliente PEP pierde conexión con el servidor de políticas remoto (PDP), este puede seguir operando mediante un servidor local (LPDP – Punto de Decisión de Políticas Local). Cuando vuelve a recobrar la conexión con el servidor remoto, el cliente le informa de todas las actualizaciones que se realizaron en el intervalo de tiempo en que la conexión fue perdida.

Éste estaría implementado junto con el PEP y guardaría durante un tiempo las decisiones de las políticas aplicadas a las peticiones de tal manera que, una petición igual y poco separada en el tiempo podría ser administrada sin necesidad de tener que enviar una petición al PDP.

*Herramienta de gestión de políticas:* es una interfaz utilizada por el administrador para crear o modificar políticas. Recibe como entrada políticas de alto nivel introducidas por el usuario o administrador. Las políticas generadas por la herramienta son transformadas a políticas de bajo nivel para ser cargadas sobre modelos de información estándar en el repositorio de políticas.

*Contenedor de Políticas:* es usado para almacenar las políticas generadas por la herramienta de gestión. Almacenar toda la información concerniente a las políticas, reglas y condiciones y acciones y cualquier información extra necesaria para tomar una decisión.

El repositorio puede ser, dependiendo de la implementación, un específico repositorio de datos que almacena reglas de políticas, sus condiciones y acciones y sus datos relacionados. Una base de datos o directorio LDAP podrían ser un ejemplo. En [PCIM01] se define un modelo más amplio y especifico para representar información de políticas.

La información intercambiada entre el PEP y el PDP requiere un protocolo de comunicación, una variedad de protocolos pueden ser usados; el más usual es el

protocolo COPS Common Open Policy Service [COPS01] definido por el IETF. Aunque también se puede utilizar SNMP (Simple Network Management Protocol).

COPS es definido como un protocolo estándar para intercambiar información de políticas y decisiones. COPS fue diseñado para operar fiablemente y en tiempo real, con mínima sobrecarga, para proveer un control dedicado de políticas por el PEP. COPS fue originalmente diseñado para proveer control de políticas al protocolo de reservación de recursos RSVP, sin embargo, su diseño genérico posibilita el intercambio de otras decisiones de políticas. COPS proporciona la seguridad a nivel de mensaje para la autenticación, protección e integridad del mensaje, aunque también puede utilizar los protocolos como IPSEC para la seguridad o TLS para la autenticación y para asegurar el canal entre el PEP y el PDP. El protocolo COPS es explicado en las detalle en el capítulo 3.

Existen muchos beneficios para sistemas de gestión de redes basado en políticas, algunos de ellos son: una alternativa inteligente para la asignación de recursos, proveer mejor seguridad, gestionar la complejidad de los dispositivos, gestionar servicios y trafico complejos, cambiar configuraciones de dispositivos dentro de una ventana de tiempo especifica y desempeñar funciones de provisión programadas [IBA08].

## 2.4 WS-Policy

Un proveedor de Web Services puede definir condiciones o políticas sobre las cuales el servicio es provisto. El Web Services Policy Framework (WS-Policy) [WSP06] provee un modelo y sintaxis correspondiente para describir las políticas de un Web Service. WS-Policy forma parte de la familia de especificaciones de tecnologías basadas en Web Services del W3C. Es una recomendación del W3C desde septiembre de 2007. Microsoft, IBM, BEA, y SAP trabajan sobre esta especificación con el fin de obtener un marco de trabajo genérico de políticas. WS-Policy define un modelo genérico y una sintaxis para describir y comunicar las políticas de los servicios Web.

WS-Policy define una gramatica flexible y extensible para expresar las capacidades, requerimientos y características generales de entidades en un sistema basado en Web Services. WS-Policy define una arquitectura y modelo para expresar esas propiedades como políticas.

WS-Policy representa los requerimientos de un Web Services en su interacción con otros Web Services o consumidores [AND04]. Aplicada al modelo de Web Services las políticas son usadas para comunicar condiciones en una interacción entre dos puntos finales de Web Services.

Permite a los programadores de Web Services especificar sus políticas relativas a Autorización, Autentificación, Calidad de Servicio, opciones específicas del servicio, etc.

Este framework no especifica como las políticas son adjuntadas en un Web Services. Otras especificaciones son libres de definir los mecanismos y tecnología para asociar las políticas con varias entidades y recursos. WS-PolicyAttachment [WSPA06] define tales mecanismos.

La meta de WS-Policy [WSP06] es proveer el mecanismo necesario para habilitar a Web Services especificar información de políticas.

WS-Policy [WSP06] provee un lenguaje flexible para expresar las capacidades y requerimientos del servicio. Algunos ejemplos de lo que se puede expresar utilizando políticas son:

- Un servicio que soporte MTOM Attachment [MTOM05]
- Un servicio que requiera WS-ReliableMessaging [WSRM03]
- Un servicio que requiera usar WS-Security [WSS04] sobre HTTP.

El siguiente ejemplo ilustra una definición de política usando WS-Policy:

```
<wsp:Policy
  xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        <wsp:ExactlyOne>
          <sp:Basic256Rsa15 />
          <sp:TripleDesRsa15 />
        </wsp:ExactlyOne>
</wsp:Policy>
```

Este ejemplo representa una política de seguridad usando aserciones definidas en [WSSP05]. Ilustra una política para el algoritmo requerido para ejecutar operaciones de criptografía. El elemento <wsp:ExactlyOne> es usado para expresar que una de las dos aserciones debe ser encontradas. Los elementos <sp:Basic256Rsa15/> y <sp:TripleDesRsa15 /> son definidos como parte de WS-SecurityPolicy [WSSP05] para expresar requerimientos de seguridad en el servicio.

A continuación se definen los principales elementos incluidos en la especificación [WSP06] utilizados en el modelo WS-Policy

### 2.4.1 Aserciones de Política

WS-Policy define las políticas como un conjunto de políticas "alternativas." En este contexto una política alternativa es un conjunto de aserciones de políticas usadas por la aplicación procesando el mensaje WS-Policy.

Una aserción de política representa una capacidad o requerimiento. Por ejemplo, una aserción de política puede estipular que una solicitud a un Web Services sea encriptada. De la misma manera, una aserción de política puede definir el máximo tamaño del mensaje que un Web Service puede aceptar.

Aserciones generales de políticas son definidas en la especificación WS-PolicyAssertions [WSPA06] que provee un inicial conjunto de aserciones de políticas para WS-Policy. Las aserciones de políticas pueden ser combinadas en diferentes caminos para expresar combinaciones consistentes de comportamiento. Hay tres operadores de políticas para combinar aserciones de políticas: *Policy, All* y *ExactlyOne*. Las aserciones son definidas por desarrolladores, diseñadores de productos, autores de protocolos y usuarios.

### 2.4.2 Expresión de Política

Una Expresión de Política es una representación formal en XML de una política que permite declarar las políticas de una manera más operable. El esquema para la expresión formal de una política es el siguiente:

*<wsp:Policy ... >*

*<wsp:ExactlyOne>*

*( <wsp:All> ( <Assertion ...> ... </Assertion> )* </wsp:All> )**

*</wsp:ExactlyOne>*

*</wsp:Policy>*

A continuación se describen los elementos:

- *<wsp:Policy>* : una expresión de política.

- *<wsp:ExactlyOne>* : una colección de políticas alternativas.

- *<wsp:All>* : una colección de aserciones de políticas.

[WSP06] define dos formas para expresiones de políticas, normal y compacta. Para simplificar el procesamiento y mejorar la interoperabilidad la forma normal de una expresión de política debe ser usada donde sea practica.

## 2.5 Estado del Arte

Dentro del área de investigación sobre el tema planteado en esta tesis podemos encontrar publicaciones que, si bien definen aplicaciones óptimas para el uso de políticas, no abordan un trabajo general, que propongan algún protocolo como COPS que soporte varios tipos de políticas.

Los trabajos más afines con lo desarrollado en esta tesis, proponen el uso de políticas para gestión de un recurso o aplicación específica, como por ejemplo políticas para manejo de calidad de servicio, control de acceso, seguridad.

A continuación se describen algunos de estos trabajos y se plantean las ventajas y desventajas de cada uno.

En [CHC07] Cheng, Hung y Chiu se centran en la negociación de políticas de privacidad entre usuarios y proveedores de servicios adoptando eXtensible Access Control Markup Language XACML como lenguaje de definición de políticas. El proceso de negociación se formaliza en el contexto de Web Services.

El trabajo resalta los requerimientos sugeridos en [ANDR05] que debe cumplir un lenguaje de definición de políticas para soportar aplicación de políticas de privacidad y realiza una comparación entre diferentes lenguajes, como por ejemplo, P3P del W3C, EPAL de IBM o XACML de OASIS.

De la comparación realizada se destaca que XACML es el único que cumple con todos los requerimientos citados por [ANDR05] y es uno de los motivos por los cuales lo adoptan como lenguaje de definición de políticas en su framework de negociación de políticas de privacidad sobre Web Services.

De acuerdo a la especificación de XACML que comparte el modelo abstracto para aplicación de políticas definido en [PFIETF99], el modelo de gestión propuesto agrega un nuevo componente, el Punto de Negociación de Política PNP, que definen como el punto donde la política es negociada. Después de recibir un requerimiento desde el PEP, el PNP chequea si el requerimiento es permitido o no para reenviar el mensaje al PDP. Si cualquier conflicto es encontrado, el PNP solicita más información desde el PEP y trata de resolver el conflicto.

Dentro de la arquitectura de gestión de políticas definida en [PFIETF99], sobre la cual se basa el modelo propuesto en la tesis, en [CHC07] se agrega un elemento mas, el PNP, si bien se puede tomar como un componente adicional mediante el cual se tiene un menor acoplamiento entre cliente (PEP) y servidor (PDP) ya que actúa como intermediario, no es muy claro si puede manejar o no a múltiples PDP. El PNP puede resultar un componente útil al comienzo de la comunicación si el PEP no conoce cual es el PDP al que debe solicitar políticas y la información que requiere, pero durante la comunicación, es mejor la arquitectura propuesta en [PF-IETF] donde la comunicación entre el PEP y el PDP es directa y no requiere de ningún intermediario. En ciertos escenarios podría no ser necesario tener un PNP entre la negociación de políticas entre el PEP y el PDP ya que el PEP puede conocer de antemano cual es su PDP y la información que se requiere para negociar las políticas.

Adoptar XACML como lenguaje de definición de políticas es ventajoso dentro del contexto de políticas de privacidad, pero si se requiere extender el framework a ser aplicable a otros servicios de negociación de políticas presenta ciertas restricciones, ya que es un modelo de solicitud/response, no define una forma de enviar políticas no solicitadas, lo cual se requiere por ejemplo para gestión dinámica de calidad de servicio, donde el servidor puede asignar recursos (ancho de banda) mediante políticas, de acuerdo al estado de la red o las solicitudes de los clientes, no solo mediante un requerimiento de un PEP.

Uno de los tópicos mas importantes en el proceso de descubrir Web Services para los proveedores y clientes de Web Services es negociar y encontrar una solución que sea aceptable para ambos, para ello se requiere un modelo con características de negociación mas sofisticadas. En [HLJ04] Hung, Li, Jeng proponen un lenguaje declarativo XML llamado WS-Negotiation conteniendo 3 partes: Mensajes para negociación, que describe el formato para intercambio de mensajes entre las partes negociantes, Protocolo de negociación que describe el mecanismo y las reglas que las partes deberían seguir y Creación de Decisiones de negociación que es un proceso de decisión privado basado en el modelo costo-beneficio.

El trabajo propone una aplicación del modelo de WS-Negotiation como un framework genérico para Service Level Agreement (SLA). Después de una secuencia de mensajes de negociación utilizando WS-Negotiation, un documento SLA es creado y usado para especificar los parámetros de nivel de servicio que describen las garantías y obligaciones de las partes negociantes.

La estructura de WS-Negotiation puede ser utilizada para la negociación de políticas sobre Web Services, si bien el protocolo y formato de mensajes es definido por los autores y no usan estándares, presentan un modelo abierto que permitiría agregar cualquier tipo de política a la negociación. Aunque el modelo para la toma de decisiones basado en costo-beneficio no siempre es aplicable dentro del dominio de políticas. En el trabajo solo e discute la negociación entre dos partes, pero están investigando la negociación uno a muchos y la posibilidad de aplicar WS-Negociation con otros lenguajes XML tales como BPEL4WS.

Debido al incremento de las aplicaciones con alta demanda de recursos, las características actuales que ofrecen las redes 3G no son suficientes, conforme las redes inalámbricas se vuelven mas complejas, nuevos métodos de control jerárquicos son necesarios. Un método para resolver este dilema es utilizar políticas de control o gestión en lugar de configuración de parámetros de servicio. Gerrero Ibáñez y Barba Marti en [AGRP-3G] proponen la implementación de una arquitectura jerárquica de gestión de red basada en políticas para un ambiente integrado 3GWLAN.

Para la gestión basada en políticas, Gerrero Ibáñez y Barba Marti se basan en la arquitectura presentada en [PFIETF99] utilizando COPS como protocolo de comunicación de políticas, definen un esquema de políticas formado por 6 niveles con los cuales cubren facturación, tarifación, contabilidad, control de acceso, control de calidad de servicio y continuidad del servicio. El proceso de cómo se toman las decisiones de las políticas para el usuario es mediante una serie de operaciones que se llevan a cabo a fin de seleccionar la mejor red para el usuario.

La negociación de calidad de servicio sobre Internet puede alcanzar protocolos para la negociación de niveles de servicios. Tales protocolos de propósito general incluyen COPS para especificación de políticas dentro de un dominio. Políticas de calidad de servicio dan al usuario la posibilidad de especificar el nivel de servicio deseado.

En [SLNP02] Nguyen y Boukhatem proponen una extensión del protocolo COPS para negociación de nivel de servicio. Consta de dos fases, configuración y negociación. En la fase de configuración, el PDP le indica al cliente como requerir un nivel de servicio. Por ejemplo, le informa el intervalo de tiempo para renegociar. El PEP instala la configuración y envía un reporte al PDP. En la fase de negociación, el PEP envía mensajes requiriendo nivel de servicio, el PDP puede aceptar, rechazar o proponer otro nivel de servicio para el cliente. Una decisión no solicitada puede ser enviada por el PDP en caso de que la red necesite el nivel de servicio del cliente o reconfigurar la fase de negociación.

La información intercambiada entre el PEP y el PDP es representada por una estructura de datos conocida como Policy Information Base (PIB) [CHAN01]. La PIB es descripta usando notación ASN.1. Una PIB puede ser descripta como un conjunto de clases.

Una de las ventajas de usar una PIB es que el protocolo se vuelve independiente de la información de políticas acarreada. Otra ventaja es que las clases definidas en una PIB son extensibles por otras PIBs. La organización de COPS-SLS en dos fases hace una negociación dinámica y configurable. Como aplicación del modelo propuesto se presenta una plataforma de red global donde se aplica una arquitectura de control IP mediante el protocolo COPS-SLS. El artículo presenta COPS-SLS como protocolo para acuerdo de nivel de servicio, mostrando la flexibilidad del protocolo COPS y la utilidad del uso de una PIB para representar la información.

Otra de las áreas de investigación en el uso de políticas sobre sistemas distribuidos es el manejo de calidad de servicio. Requerimientos no funcionales relacionados a la calidad del software tales como seguridad, fiabilidad, disponibilidad, performance varían dependiendo del contexto y algunas veces en tiempo de ejecución, complicando la tarea del reuso de componentes. Además, diferentes consumidores de servicios pueden tener distintos requerimientos de calidad de servicio, el proveedor del servicio debería asegurar la calidad de sus servicios de acuerdo a los requerimientos de sus clientes.

De los trabajos encontrados en el área se puede citar [STEF04] de Wohlstadter y Stefan que presentan una aproximación basada en capa-intermedia para manejar dinámicamente cambios en requerimientos de calidad de servicio. Las políticas son usadas para características no funcionales. Proveen un lenguaje declarativo para especificar propiedades, preferencias y conflictos de calidad de y un mecanismo de resolución que razona usando esas especificaciones para encontrar dinámicamente un conjunto de propiedades de calidad de servicio que satisfagan los requerimientos.

La aproximación basada en políticas esta inspirada por un estándar emergente denominado WS-Policy. El propósito del uso de la especificación WS-Policy en [STEF04] es como sintaxis para expresar combinaciones lógicas de datos XML llamados aserciones que describen características no funcionales de los Web Services. El protocolo de mediación de políticas dentro de [STEF04] es una extensión de WS-Policy para intercambiar información de políticas entre cliente y servidor. Las políticas

del cliente y servidor deben ser consideradas en conjunto por la capa-intermedia para encontrar confidencias y atribuir valores. El mediador definido dentro de [STEF04] hace esto utilizando un protocolo de mediación de políticas que facilita el actual intercambio de datos y un algoritmo que involucra tres pasos: reducción de políticas en tiempo de ejecución, calcular un conjunto de aserciones y encontrar un conjunto comparable de aserciones.

En el modelo definido en [STEF04] se destaca la posibilidad de manejar dinámicamente los requerimientos de calidad de servicio, pero la utilización de una capa intermedia y la definición de una sintaxis particular para las políticas pueden tornarlo un poco complicado de implementar.

En otro trabajo relacionado a la calidad de servicio [RAJN99] Rajan, Verma y Kamat, examinan la definición, desarrollo y gestión de políticas relacionadas a calidad de servicio sobre una red IP.

En [WCWF04] se describe una integrada arquitectura de gestión de Calidad de Servicio y sus servicios. Clasifica características de Calidad de Servicio en 4 categorías y cada una de las cuales es descompuesta dentro de un conjunto de atributo. Integra esas características en un lenguaje XML para la aplicación y para expresar requerimientos de calidad de servicio y contratos.

En comparación con otros trabajos en gestión de Calidad de Servicio, esta arquitectura y solución provee técnicas, extensiones y generalizaciones usando gestión de tareas mediante una capa intermedia.

# 3. PROTOCOLO COPS (COMMON OPEN POLICY SERVICE)

## 3.1 Introducción:

COPS [COPS01] RFC 2748 y COPS-PR [CHAN01] RFC 3084, son dos protocolos descriptos por el IETF, que en el contexto de sistemas de gestión basado en políticas proveen la comunicación entre el PDP y PEP. La necesidad de ambos protocolos puede ser atribuida al trabajo desarrollado por el IETF en el área de distribución de políticas de calidad de servicio.

El protocolo COPS [COPS01] es definido como un protocolo de consulta y respuesta que es usado para intercambiar información entre un servidor (Punto de Decisión de Políticas PDP) y un cliente (Punto de Ejecución de Políticas PEP). El modelo cliente/servidor soporta control de políticas sobre protocolos de señalización. Los elementos de políticas son independientes del tipo de señalización usado.

Un objetivo de este protocolo es tener un diseño simple pero extensible, significando que este puede ser utilizado como base para nuevas variantes, llamados clientes.

Las principales características de este protocolo incluyen:

1. El protocolo emplea un modelo cliente/servidor donde el PEP envía solicitudes, actualizaciones y eliminación de mensajes al PDP remoto y el PDP retorna decisiones al PEP.

2. El protocolo usa TCP como su medio de transporte para intercambiar mensajes entre cliente y servidor.

3. El protocolo es extensible ya que este es diseñado para acarrear objetos descriptos por si mismo y puede soportar diversa información de cliente sin cambios al protocolo. El protocolo fue creado para general administración, configuración y aplicación de políticas.

4. COPS provee un nivel de seguridad, permitiendo autentificación, integridad de mensajes. También puede rehusar protocolos existentes para seguridad tales como IPSEC o TLS para autenticar o asegurar el canal entre el PEP y el PDP.

5. COPS es un protocolo de estados (stateful) en dos aspectos:

    a. Comparte el estado Solicitud/Decisión entre el cliente y el servido, es decir, el cliente pide un requerimiento al servidor y este le responde en función a lo que solicite. La respuesta que envía el servidor es generada en forma asincrónica, ya que puede responder sin que haya una solicitud previa por parte del cliente.

b. Asocia el estado Solicitud/Decisión en función a varios eventos, es decir, el servidor puede responder a las solicitudes en forma diferente en función a los estados previamente instalados.

6. Además, es un protocolo de estado, ya que permite que el servidor le envíe información de configuración al cliente, y cuando el servidor crea que la configuración instalada no es de utilidad puede pedir al cliente la remueva.

## 3.2 Modelo básico

La **Figura 3.1** ilustra un ejemplo de COPS usado para comunicar información de políticas entre un PEP (cliente) y un PDP (servidor) dentro del contexto de un particular tipo de cliente.

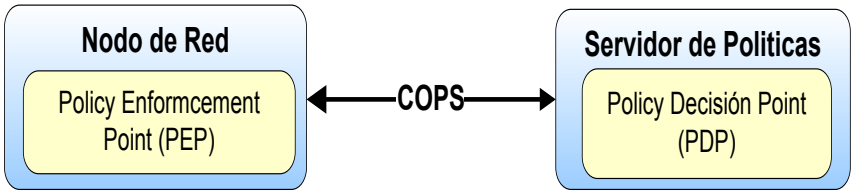



**Figura 3.1** *Arquitectura COPS*

El PEP es responsable de iniciar una conexión persistente con el PDP. El PEP usa esta conexión para enviar una solicitud de política y recibir Decisiones desde el PDP. A través de esta conexión el PDP puede enviar Decisiones no solicitada al PEP forzándolo a cambiar una política previamente enviada. Si la decisión tomada por él fue correctamente instalada o se ha producido una falla, para tareas de monitoreo y control, esto lo realiza por medio del mensaje Report (RPT). De modo similar, el PEP puede solicitar el borrado de algunas políticas utilizando el mensaje Delete State Request (DRQ).

El protocolo es diseñado para comunicar objetos identificados por si mismo que contienen los datos necesarios para identificar un Requerimiento, estableciendo el contexto del mismo, refiriendo a requerimientos previamente instalados, identificando el tipo de requerimiento, reportando errores, y transferir información especifica del cliente.

Para distinguir entre diferentes clases de clientes, el tipo de cliente es identificado en cada mensaje. Diferentes tipos de clientes, pueden tener diferente información específica y pueden requerir diferentes tipos de políticas en la decisión. Es de esperar que cada nuevo tipo de cliente tenga un correspondiente documento especificando su interacción con este protocolo.

El contexto de cada requerimiento corresponde al tipo de evento que dispara esto. El contexto de objetos COPS [COPS01] identifica tres tipos de eventos: 1) el arribo de mensajes, 2) asignación de recursos locales y 3) el envió de mensajes.

Cuando el cliente PEP pierde conexión con el servidor de políticas remoto (PDP), este puede seguir operando mediante un servidor local (LPDP – Punto de Decisión de

Políticas Local). Cuando vuelve a recobrar la conexión con el servidor remoto, el cliente le informa de todas las actualizaciones que se realizaron en el intervalo de tiempo en que la conexión fue perdida.

La tolerancia a fallos es una capacidad requerida del protocolo, el PEP y PDP verificaran su conexión constantemente mediante el envío de mensajes Keep Alive. Cuando una falla es detectada el PEP debe tratar de reconectarse al remoto PDP o a algún PDP alternativo. Mientras esta desconectado, el PEP debe usar su servidor local de políticas LPDP, cuando la conexión es restablecidas, todas las decisiones de políticas generadas localmente son reenviadas al PDP.

## 3.3 El Protocolo*:*

EL protocolo [COPS01] especifica los mensajes y objetos que serán utilizados para intercambio de información de políticas. En esta sección describe el formato de los mensajes y objetos intercambiados entre el PEP y el PDP, en el Anexo A se describe con mas detalle los objetos y mensajes.

### 3.3.1 Objetos

Los objetos son utilizados para encapsular la información que se transfiere entre las entidades PEP y PDP. Según el tipo de cliente con el que se esté trabajando, en los mensajes se agregarán más o menos objetos, esto quiere decir que la longitud de los mensajes nunca es fija, salvo algunas excepciones. Cada objeto consiste en un encabezado y el contenido (en donde encapsula la información a transmitir). La estructura del encabezado consta de tres campos Length, C-Num y C-Type; y el contenido puede ser desde un string hasta otro objeto de algún protocolo.

| Objeto | Descripción |
|---|---|
| Handle | Encapsula un único valor que identifica un estado instalado; responsable de identificar una conexión entre PEP y PDP. Especifica un dominio |
| Context | Especifica el tipo de evento que esta motivando al mensaje, por ejemplo una solicitud de configuración, asignación de recursos, control de admisión. |
| In Interface | Es usado para identificar el interfaz entrante en el cual una petición particular se aplica y la dirección de donde provino el mensaje recibido. |
| Out Interface | Es usado para identificar el interfaz saliente al cual una petición específica se aplica y la dirección para donde el mensaje expedido debe ser enviado. |
| Reason Code | Especifica la razón por la cual el estado de la petición fue borrada.<br>Este objeto aparece en el mensaje DRQ (Delete Request State) |
| Decision | Es la decisión tomada por el PDP. Devuelve una respuesta a la petición dependiendo el tipo de cliente. La información transportada es la configuración solicitada por el clliente. |
| LPDPDesicion | Son las decisiones tomadas por el servidor de políticas local (LPDP). Puede aparecer en los mensajes REQ. Los subtipos especificados en el objeto anterior son validos para esté. |
| Error | Utilizado para identificar los errores en el protocolo, como por ejemplo mensaje mal formado, cliente no soportado, falla de autentificación. |
| ClientSI | Es utilizado en los mensajes REQ, OPN y RPT. Contiene información específica de los clientes. |
| KeepAliveTimer | Es usado para especificar el tiempo máximo del intervalo en que un mensaje COP debe ser enviado o recibido. |
| PEPIdentificacion | Es utilizado para identificar al cliente PEP en el servidor PDP. Se emplea en los mensajes OPN. Por ejemplo este puede ser una dirección IP o nombre de dominio. |
| Report Type | Es utilizado en el mensaje RPT, reportando el estado de una decisión. |
| PDP Redirect Address | Cuando el PDP cierra una sesión con el PEP de un client – type, puede usar este objeto para desviar el PEP a otra dirección especificando la dirección del PDP y el número de puerto TCP. |
| Last PDP Address | Es utilizado en el mensaje OPN, indica la dirección del servidor PDP que el PEP utilizó por última vez |
| Accounting Timer | Determina el intervalo de tiempo que debe esperar el PEP antes de realizar una petición. |
| Message Integrity | Incluye una secuencia de números y un resumen del mensaje, usado para la autentificación y validación del mensaje |

**Tabla 3.1** Resumen objetos COPS

### 3.3.2 Mensajes:

Los mensajes se caracterizan por ser utilizados para realizar la comunicación entre las entidades PEP y PDP. Estos se integran por objetos que permiten transportar, de una unidad a otra, la información suficiente para tomar una decisión en función de las políticas previamente establecidas. Cada mensaje posee una función particular en la comunicación entre el PEP y PDP, esto nos lleva a que hay mensajes que solo son utilizados por el servidor (PDP) y otros por el cliente (PEP).

Además, al igual que los objetos, los mensajes se constituyen de un encabezado, que nos entrega información referida al tipo de mensaje que se ha enviado, que tamaño posee y cual es el cliente con el que se está trabajando. Además, nos dice si el mensaje es una respuesta a una solicitud o ha sido enviado por cuenta de cualquiera de las entidades que intervienen en la comunicación.

| Mensaje | Dirección | Descripción |
|---|---|---|
| Client-Open | PEP -> PDP | Inicia la comunicación entre el PED y el PEP. Abre una nueva sesión para un tipo de cliente. |
| Client-Accept | PDP -> PEP | Cuando la comunicación se inicio correctamente este mensaje da una respuesta positiva al mensaje Client-Open |
| Client-Close | PEP <-> PDP | Termina una sesión para un tipo de cliente. También es usado para una respuesta negativa a un mensaje Client-Open |
| Request | PEP -> PDP | El PEP envía un mensaje de Request al PDP solicitado políticas. También es usado para informar las decisiones tomadas por el servidor de políticas local LPDP. |
| Decision | PDP -> PEP | Mediante este mensaje el PDP envía una respuesta a la solicitud realizada por el PEP. En el mismo se incluye toda la información de la decisión para ser aplicada por el PEP |
| Report State | PEP -> PDP | Es usado por el PEP para notificar al PDP si la decisión enviada en el Decisión mensaje fue aplicada con éxito |
| Delete Request State | PEP -> PDP | Determina que la política aplicada previamente dejo de ser utilizada. |
| Synchronize State Request | PDP -> PEP | El PDP solicita que el PEP reenvíe su información para una sincronización con el PDP |
| Synchronize State Complete | PEP -> PDP | Es una respuesta al SSQ, indicando que la sincronización fue finalizada. |
| KeepAlive | PEP <-> PDP | Es un mensaje enviado entre entidades como forma de señalar su estado activo. |

**Tabla 3.2** *Resumen mensajes COPS*

## 3.4 Esquema de Comunicación:

COPS utiliza TCP como protocolo de transporte para un fiable intercambio de mensajes entre el PEP y el PDP. El PEP es el encargado de iniciar la comunicación, el PDP debe escuchar en un puerto pre-deifnido (3288).

Para identificar con quien se ha establecido la comunicación, el servidor PDP utiliza el objeto PEPID, proporcionado por el cliente PEP para identificarse como cliente físico. El cliente puede trabajar con varios tipos de clientes, para cada uno de ellos debe realizar la inicialización de una sesión COPS, empleando el mensaje OPN especificando el tipo de cliente en el campo Client Type en el encabezado del mensaje. Esto crearía un túnel entre el cliente y el servidor, tal como se observa en la figura 3.2 con color celeste. Cada tipo de cliente maneja en forma independiente los Handles, esto implica que el servidor realizará un control de los Handles por cada sesión iniciada. Otro punto a observar es el objeto Keep-Alive, en donde cada tipo de cliente puede tener por defecto un tiempo mínimo designado para el Keep-Alive, pero por conexión TCP se utiliza el menor valor de tiempo para todas las sesiones iniciadas. Por último, si se utiliza seguridad en los mensajes, se debe iniciar previamente a cualquier otra sesión COPS y se manipulará el campo Secuence Number del objeto Integrity por conexión TCP.

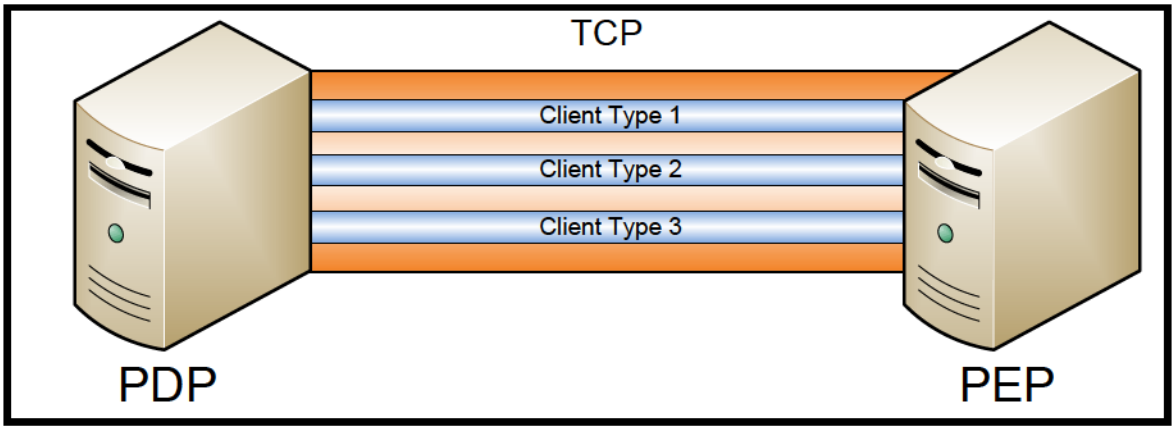



**Figura 3.2**: *Conexión TCP con tres sesiones COPS iniciadas por distintos tipos de clientes.*

### 3.4.1 Intercambio de Mensajes

La **Figura 3.3** muestra el flujo de mensajes entre un PEP y un PDP

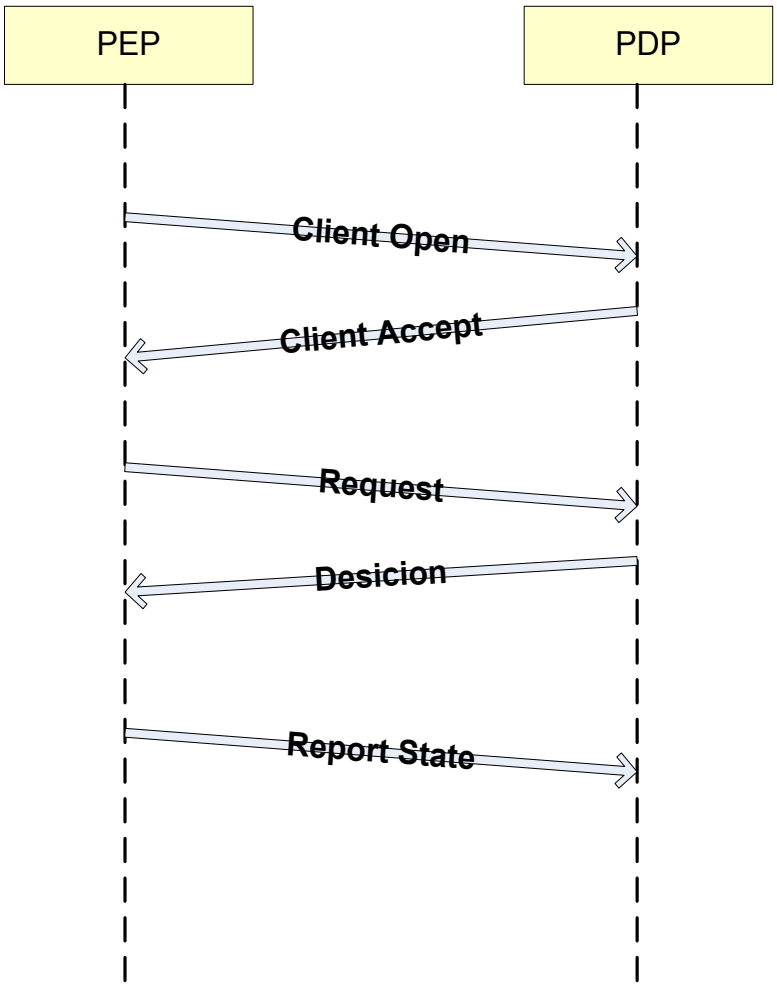



**Figura 3.3** *Intercambio de mensajes entre el PEP y el PDP*

En el flujo de mensajes el PEP, la inicialización se da con los mensajes Client-Open Client-Accept, luego el PEP comienza el envío de solicitudes de políticas de configuración mediante mensajes de Request y las decisiones del servidor PDP son encapsuladas en mensajes Decisión, cada un periodo dado el PEP envía mensajes de Report-State reportando el estado de cada política enviada.

El PEP puede enviar otro Request para actualizar un estado previo o crear uno nuevo. Si la política en el PDP son cambiadas y esto afecta a cualquier PEP este debería enviar de Decisión no solicitados a todos los PEP afectados con la nueva política.

### 3.4.2 Seguridad:

El protocolo provee un objeto Integrity que puede autentificación e integridad de mensajes. Todas las implementaciones COPS deberían implementar integridad. El objeto Integrity es usado entonces para validar todos los mensajes enviados sobre una conexión TCP entre el PEP y el PDP.

La seguridad de mensajes COPS es negociada una sola vez por conexión y cubre toda la comunicación sobre una particular conexión. Si un nivel de seguridad es requerido este

debe ser negociado mediante los mensajes Client-Open/Client-Accept especificando el Client-Type en 0. En caso de que el Servidor no trabaje con seguridad, este enviará un mensaje Client Close (CC).

### 3.4.3 PEP Inicialización:

Después de que una comunicación es establecida entre el PEP y un remoto PDP y luego de negociar la seguridad (si es necesario), el PEP enviara uno o más Client-Open mensajes al PDP, uno para cada Client-Type soportado por el PEP. El PDP responderá con separados Client-Accept para cada Client-Type requerido. Si un Client-Type no es soportado por el PDP, este enviara un Client-Close indicando tal situación

### 3.4.4 Modo de Operación:

COPS tiene dos modos de operación, el *outsourcing* escenario y el *configuring* escenario, la diferencia esta en el modo que ellos manejan las políticas.

*Outsourcing* escenario:
Cuando un evento ocurre en el PEP que requiere una nueva política de decisión, el PEP, el PEP solicita la decisión al PDP enviándole un Request Message, acarreando la información necesaria. El PDP entonces envía un Decision Message al PEP diciéndole que hacer. Un Report State Message es enviado como respuesta de la operación.

*Configuring* escenario:
En este escenario el PEP envía un Request de configuración notificándole al PDP que desea ser configurado. La configuración para el PEP es enviada en uno o más Decision Messages. Para cada Decisión Messages el PEP envía un Report State Message conteniendo el resultado de la configuración.

### 3.4.5 Keep-Alive

Un Keep-Alive Message es usado para validar que la conexión entre el cliente y el servidor esta aun funcionando siempre que no haya flujo de mensajes. El PEP debe enviar este mensaje de acuerdo al valor del intervalo de tiempo indicado por el PDP. El cliente siempre envía el mensaje y el servidor le contesta con el mismo mensaje. En caso de que el servidor no reciba este mensaje, cerrará automáticamente la conexión con el cliente.

### 3.4.6 PEP/PDP Close

El Client-Close mensaje es usado para terminar una conexión entre el PEP y el PDP para un tipo de cliente, se puede enviar luego de un Client-Open si el PDP no soporta el tipo de cliente requerido por el PEP o puede ser enviado por el PEP para notificar que no enviara mas solicitudes para un tipo de cliente.

### 3.4.7 Sincronización

Cuando se desconecta desde el PDP, el PEP debe comenzar a utilizar su servidor de políticas local LPDP. Cuando la conexión es establecida nuevamente, el PEP debe enviar al PDP todas las decisiones obtenidas localmente. Adicionalmente, el PDP puede requerir que todos los PEP se sincronicen nuevamente enviando mensajes de sincronización de estados (Synchronize State Request), el PEP debe pasar todos sus estados configurados al PDP seguidos por un mensaje Synchronize State Completed.

COPS Es un protocolo estándar para intercambiar información de políticas y decisiones, es definido como un protocolo de consulta y respuesta. COPS fue diseñado para operar fiablemente y en tiempo real, con mínima sobrecarga, para proveer un control dedicado de políticas por el PEP.

Al utilizar COPS se logra un modelo para la negociación de cualquier tipo de políticas, tanto de configuración para controlar dispositivos que usen los recursos disponibles, como políticas que permitan manejar QoS o políticas de privacidad.

# 4. ARQUITECTURA ORIENTADA A SERVICIOS

## 4.1 Introducción:

En los últimos años, el mundo de los negocios y el mundo de las tecnologías de la información estuvo estrechamente relacionado, las compañías llevaron sus procesos de negocios al mundo de las tecnologías de la información, surgieron aplicaciones empaquetadas, aplicaciones legadas que son críticas dentro de una organización, estas representan su ventaja competitiva. Hasta ahora estas aplicaciones eran prácticamente inflexibles a cambiar los requerimientos de negocios.

La evolución de los procesos de negocios ha resultado en una gran número de aplicaciones así como software empaquetado tales como Enterprise Resource Planing (ERP), Customer Relationshp Managment (CRM), y Supply Chain Management (SCM) que hoy en día proveen los datos y la lógica de negocio.

La orientación a servicios tiene el potencial de ser un paradigma que verdaderamente atraen la tecnología y los negocios en conjunto en un nivel donde la gente de ambos lados puede igualmente entender y hablar de los conceptos esenciales.

Una de las principales metas de SOA es alinear el mundo de los negocios con el mundo de tecnologías de la información en una manera que ambos sean efectivos.

La orientación a servicios es el resultado de un largo proceso de evolución y tiene el potencial de finalmente proveer estabilidad en un ambiente que ha estado evolucionando constantemente en los últimos 30 años.

La orientación a servicios yace su raíz en la teoría de ingeniería de software conocida como “separation of concern”. Que se basa en la noción que es beneficioso desglosar un problema grande dentro de una serie de problemas más pequeños. Se puede ver esto como una *Arquitectura Basada en Componentes* donde un problema grande es dividido dentro de funciones más pequeñas, cada una encapsulada dentro de un componente y gracias a los *Sistemas Distribuidos* estos componentes pueden existir en diferentes localizaciones físicas.

La computación orientada a servicios representa una nueva generación de computación distribuida. Como tal, esta rodea muchas cosas, incluyendo sus propios paradigmas y principios de diseño, patrones de diseño, un modelo de arquitectura distinto y conceptos relacionados [ERL07].

Si vemos a SOA como una arquitectura de software, podemos empezar por revisar definiciones acerca de lo que es una Arquitectura de Software:

- “Una arquitectura es un conjunto de decisiones significativas acerca de la organización de un sistema de software” según Booch, Rumbaugh, and Jacobson en [BCK03].
- “La arquitectura de software de un programa o sistema de computación es la estructura del sistema, que comprende elementos de software, las propiedades

externamente visibles de esos elementos, y las relaciones a través de ellos" según Brass, Clements, and Kazman en [BRJ99].
- "Arquitectura es la estructura organizacional del sistema" The IEEE Standard 610.12-1990
- "Arquitectura de software es el conjunto de decisiones de diseño que, realizadas incorrectamente, pueden causar que su proyecto sea cancelado" Eoin Woods, software architect.

A continuación se dará una definición de Arquitectura Orientada a Servicios (SOA), se presentaran los componentes fundamentales de SOA, como lo son servicios, contrato de servicio, repositorio de servicios y quien es el consumidor y productor de servicios. Se explicarán cuales son los principios de diseño de arquitectura que debe tener una aplicación orientada a servicios, como por ejemplo bajo acoplamiento. Se explicarán también los conceptos de extensión para SOA y una sección sobre Web Services y sus componentes.

## 4.2 ¿Qué es Arquitectura Orientada a Servicios?

Se propone citar un buen ejemplo para entender SOA:

Supongamos que estamos trabajando con programación orientada a objetos, y tenemos el objeto CD. Si hacemos caso a este paradigma el objeto CD "sabe" como reproducirse. Por otro lado, si trabajamos dentro de una Arquitectura Orientada a Servicios, el objeto CD ya se desliga de eso y le pide el servicio de reproducción ya sea a un gran equipo de música como a un pequeño discman. Ambos ofrecen el mismo servicio de reproducción de CDs pero la calidad del servicio es diferente.

Cualquier persona llevando a cabo una tarea distinta en soporte de otras está proveyendo un servicio. Cualquier grupo de individuos colectivamente realizando una tarea están también demostrando la disponibilidad de un servicio. Similarmente, una organización que lleva a cabo tareas asociadas con su propósito o negocio está también proveyendo un servicio.

Arquitectura Orientada a Servicios (SOA) se refiere a una arquitectura de software que ofrece sus funcionalidades como servicios, con énfasis en alcanzar el bajo acoplamiento de los servicios, exposición funcional gruesa (nivel de negocios), componentes re usables (servicios). Determinar como invocar esos servicios debería ser a través de una interfaz independiente de la plataforma.

Como forma de tecnología, una implementación SOA puede consistir de una combinación de tecnologías, productos, APIs, soportando extensiones de infraestructuras y varias otras partes. La fase de desarrollo de una arquitectura orientada a servicios es única en cada empresa, sin embargo está tipificada por la introducción de nuevas tecnologías y plataformas que soportan la creación, ejecución y evolución de soluciones orientada a servicios [ERL07].

Dentro de la arquitectura se cuenta con proveedores de servicios que definen como son sus servicios y como invocarlos y consumidores de servicios que usan las interfaces para construir los datos necesarios e invocar el servicio. También se puede contar con

un mecanismo de repositorio de servicios al cual el proveedor publica la interfaz de sus servicios y desde el cual los consumidores las descubren.

SOA es un buen camino para diseñar un sistema de software que provea servicios a aplicaciones finales u otros servicios en la red a través de publicar y descubrir servicios.

En una Arquitectura orientada a servicios, el sistema opera como una colección de servicios. Cada servicio puede interactuar con los otros cumpliendo una cierta tarea. La operación de un servicio puede ser la combinación de varias funciones de bajo nivel. En tal caso, esas funciones de bajo nivel no son consideradas servicios [ESBP04].

Hoy en día las organizaciones TI emplean diferentes sistemas y tecnologías. La mayoría de los analistas predicen que J2EE y .NET continuarán coexistiendo en las organizaciones al igual que la tendencia de tener tecnologías heterogéneas. Por otra parte, crear aplicaciones que aprovechen estas tecnologías diferentes ha sido históricamente una tarea desalentadora. SOA provee una clara solución para permitir que estos sistemas expongan su funcionalidad por medio de interfaces operables estandarizadas [PAN05].

El uso de SOA tiene varias ventajas:

- Adaptar las aplicaciones a las tecnologías cambiantes.
- Integrar fácilmente las aplicaciones con otros sistemas.
- Aprovechar las inversiones hechas en sistemas legados.
- Crear procesos fácilmente a partir de servicios existentes.
- Incrementa la federación
- Aumenta la operabilidad
- Diversificación de opciones
- Facilidad de intercambio de datos

## 4.3 Elementos de una Arquitectura Orientada a Servicios:

### 4.3.1 Servicios:

La unidad fundamental de una solución orientada a servicios es el servicio. Un servicio es un componente de software que encapsula una funcionalidad bien definida que opera independientemente del estado de otros servicios dentro del sistema. Los servicios tienen un conjunto de interfaces bien definidas y operan a través de un contrato entre el cliente del servicio y el servicio en si mismo [ESBP04].

Los principales componentes de un servicio son:

- *Contrato*: el contrato del servicio provee una especificación informal del propósito, funcionalidad, restricción y uso del servicio.
- *Interfaz*: la funcionalidad del servicio es expuesta por la interfaz del servicio a los clientes que están conectados al servicio.
- *Implementación*: es la realización del contrato del servicio, provee la lógica de negocio y los datos requeridos.

Un servicio puede esencialmente actuar como un contenedor de capacidades relacionadas. Este esta comprendido de una lógica de diseño que lleva a cabo esas capacidades y un contrato de servicio que expresa cuales de sus capacidades están disponibles para ser invocadas públicamente.

Los servicios poseen las siguientes características:

- Los servicios pueden ser individualmente usados o pueden ser integrados para componer servicios de más alto nivel.
- Los servicios se comunican con sus clientes por medio del intercambio de mensajes.
- Los servicios pueden participar en un workflow, donde el orden en el cual los mensajes son enviados y recibidos afecta al resultado de la operación que realiza el servicio. Esto es conocido como coreografía de servicios.
- Los servicios publican detalles tales como su capacidad, interfaces, políticas, y protocolos de comunicación soportados.

Técnicamente se puede implementar cualquier funcionalidad como un servicio, pero esto lleva a una arquitectura difícil de mantener. El concepto de identificar que funcionalidades se convertirán en un servicio no es sencillo y es clave realizar un buen análisis para la selección.

### 4.3.2 Contrato de Servicios:

Los servicios expresan su propósito y capacidades vía un contrato de servicios, esta contiene una descripción del servicio, con información técnica, propósito, funcionalidad, restricciones, políticas de acceso al servicio, ubicación, información del proveedor del servicio. Un elemento del contrato del servicio puede ser una formal descripción de la interfaz basada en lenguajes tales como WSDL, IDL.

Esto permite a la aplicación o desarrolladores examinar la descripción del servicio y determinar que es lo que hace el servicio, que protocolos de seguridad deben ser usados, como invocarlos.

Es importante poner énfasis en aspectos específicos del diseño del contrato, incluyendo la manera en la cual los servicios expresan funcionalidades, como son definidos los tipos de datos y modelos de datos, y como las políticas son adjuntadas. El contrato de servicio debe ser optimizado, apropiadamente granular y estandarizado para asegurar que los puntos finales establecidos por el servicio sean consistentes, confiables y gobernables.

Un contrato de servicio puede también incluir documentos no técnicos. Un clásico ejemplo de esto es el acuerdo de nivel de servicio (SLA), un documento que establece un contrato asociado con la calidad de características del servicio, tales como disponibilidad, accesibilidad, y performance.

Si tomamos un Web Service, un contrato de servicio consistirá de un documento WSDL, un esquema de definición XML, y posiblemente una definición de políticas WS-Policy.

### 4.3.2 Repositorio de Servicios:

Un repositorio de servicios provee las facilidades para descubrir un servicio y adquirir toda la información para usarlo. Mucha de la información requerida ya es parte del contrato del servicio, el repositorio puede proveer información adicional, tal como locación física, información acerca del proveedor. El primer elemento que debe estar disponible dentro del repositorio es el contrato de cada servicio.

Un repositorio de servicios es un elemento muy usado en SOA, pero en ciertos casos puede ser innecesario como por ejemplo si el alcance de un servicio es solo un proyecto o si este tiene un alcance local a la empresa.

### 4.3.3 Proveedor de Servicios

Provee los servicios a los posibles consumidores (aplicaciones, servicios, clientes, etc.). Este declara y publica la descripción del servicio que provee en un repositorio de servicios.

El rol del proveedor del servicio es similar al rol del servidor en una arquitectura cliente-servidor. Dependiendo del tipo de mensaje intercambiado cuando se invoca un proveedor de servicio, este puede replicar con un mensaje de respuesta o invocar otro servicio, transformándose así en consumidor de servicio.

### 4.3.4 Consumidor de Servicios

Los servicios pueden ser invocados por otras aplicaciones ya sea desde una aplicación Web o un servicio a fin de realizar una tarea de más alto nivel. Los consumidores de servicios realizan consultas al repositorio y buscan el servicio que ellos necesitan por medio de palabras claves o nombre de servicios. Con toda la información obtenida se conectan al proveedor de servicio para invocar el servicio.

Es importante tener una buena gobernabilidad en SOA, que permita un buen proceso de negociación de contrato del servicio en el cual los potenciales consumidores interactúan con el proveedor para requerir y negociar acceso a los niveles del servicio o conjunto de servicio y un buen proceso de aplicación del contrato en tiempo de acceso o ejecución. Un buen sistema de gestión basado en políticas permite controlar el proceso de negociación de contrato del servicio para que un posible consumidor pueda requerir acceso al servicio y negociar las políticas de acceso y nivel de servicio adecuado.

En la Figura 4.1 se observa el modelo donde intervienen los elementos básicos de una arquitectura orientada a servicios y la relación entre ellos.

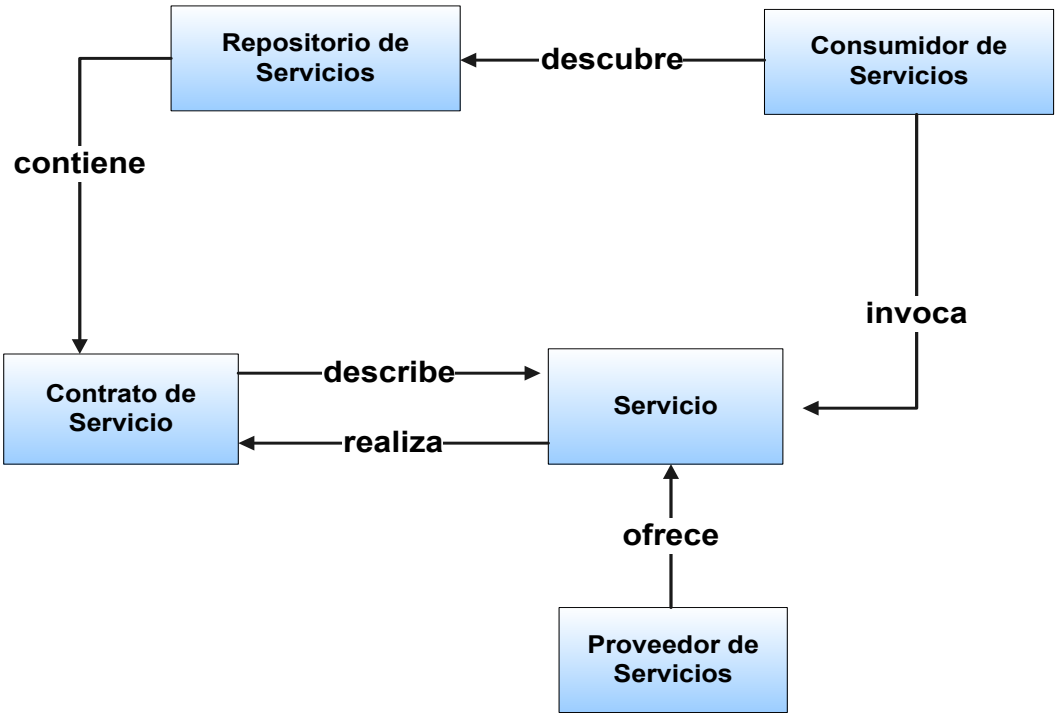



**Figura 4.1** *Componentes de una Arquitectura Orientada a Servicios*

En este escenario, se ve la interacción de los componentes de una arquitectura orientada a servicios. Como primera etapa el proveedor de servicios genera un contrato de servicios en el que expresa sus capacidades, describe el servicio, define la forma de acceso al mismo, las políticas que lo gobiernan y el tipo de datos necesario. Este contrato es publicado en un repositorio de servicios; un potencial cliente que puede ser otro servicio, consulta al repositorio, obtiene la descripción del servicio (contrato), y utiliza la misma para ligarse al proveedor del servicio e invocar el servicio.

## 4.4 Principios de Arquitectura.

A continuación se presentan algunos principios de arquitectura que debe tener una aplicación orientada a servicios.

### 4.4.1 Bajo Acoplamiento:

Acoplamiento se refiere a la conexión o relación entre dos cosas. Cualquier cosa que se conecte tiene acoplamiento y acoplar las cosas puede generar dependencias entre ellas. Una medida de acoplamiento entre dos cosas es comparable al nivel de dependencias que existe entre ellas.

Uno de las principales características que debe poseer una aplicación SOA es lograr el bajo acoplamiento entre los servicios, esto significa que los diferentes servicios tienen una mínima dependencia unos con otros.

La noción del tipo de acoplamiento esta ligada a los sistemas orientados a objetos y orientados a servicios. Sistemas distribuidos orientados a objetos tales como CORBA o DCOM los clientes tienen conocimiento estático de los objetos y se comunican a través

de invocación sincrónica de métodos remotos, usando protocolos específicos. Esto implica que la comunicación de objetos esta altamente acoplada

En un ambiente orientado a servicios, por otro lado, los servicios son bajamente acoplados, estos se comunican por intercambio de mensajes que pueden ser en un camino, sincrónicos o asíncronos [SHP05].

Bajo acoplamiento significa que los componentes de software que interactúan minimicen su conocimiento en su construcción de los demás componentes. Ellos descubren la información que ellos necesitan en el tiempo que la necesitan [SHP05].

Los beneficios de bajo acoplamiento incluyen:

- Flexibilidad: un servicio puede ser localizado en cualquier servidor y cambiarlo cuando sea necesario, solo debe mantener la referencia en su registro y los clientes lo podrán encontrar.
- Escalabilidad: los servicios pueden ser agregados y removidos de acuerdo a la demanda.
- Tolerancia a fallos: si un software, un componente o la red fallan, el servicio seria inaccesible, pero los clientes pueden consultar al repositorio por servicios alternativos.

Para lograr bajo acoplamiento se deben tener en cuenta dos aspectos:

1. Un pequeño conjunto de pequeñas interfaces para todos los servicios. Solamente semántica genérica es definida en las interfaces. Las interfaces deben ser universalmente disponibles para todos los consumidores y proveedores de servicios.
2. Mensajes descriptivos limitados por un esquema extensible desarrollado a partir de sus interfaces. Un esquema limita el vocabulario y estructura de los mensajes, un esquema extensible permite introducir nuevas versiones de servicios sin romper servicios existentes.

### 4.4.2 Exposición funcional gruesa:

Una de las calves de SOA es la definición del correcto nivel de exposición de los servicios, que esta determinada por su contexto funcional. La exposición total de un servicio no refleja la cantidad de lógica que este encapsula pero se puede ver la cantidad de potencial lógica que este podría encapsular, basado en su contexto. Un servicio de exposición funcional gruesa, por ejemplo, podría tener un amplio contexto funcional, a pesar de si este inicialmente expresa una o diez capacidades.

Servicios con exposición funcional fina deberían ser servicios que provean una pequeña cantidad de procesos de negocios usualmente usados, como acceso a datos básicos o funciones de conversión de datos. En cambios, servicios con exposición funcional gruesa ofrecen funciones de negocios de más alto nivel que son accedidas por sistemas expertos. Servicios con exposición funcional gruesa deben ser expuestos con sus interfaces para ser accedidos por otros servicios, y servicios de contexto funcional fino podrán ser usados internamente por los componentes de software u otros servicios pero no ser expuestos.

### 4.4.3 Transparencia de Locación:

Cuando un servicio debe ser posicionado y se requiere un rápido uso y retorno de la inversión, necesita ser fácilmente identificados y alcanzables. El diseño del servicio necesita tener calidad en la comunicación y mecanismo de descubrimiento. Los servicios deben tener su definición y información de locación en algún repositorio de servicios tal como UDDI y ser accesibles por una variedad de clientes.

### 4.4.4 Comunicación:

Para proveer bajo acoplamiento y flexibilidad con respecto a la comunicación, la arquitectura debe soportar varios estilos de comunicación, como por ejemplo uno a uno, muchos a muchos, llamadas remotas. El intercambio de mensajes debe ser tanto en forma sincrónica como asíncrona. Varios patrones de comunicación y de intercambio de mensajes pueden ser utilizados.

### 4.4.5 Contrato de servicio estandarizado

Como se menciono en sección anterior, los servicios expresas sus capacidades por medio de contratos. Cuando se diseña la interfaz de un servicio que será publico, se debe utilizar estándares de la industria en la definición de tipos de datos, políticas modelos de datos; de esta forma la negociación y comunicación entre los puntos finales es mas consistente y gobernable.

### 4.4.6 Orquestación y Coreografía de Servicios:

Una Orquestación de Servicios es una manera de especificar cómo se van a conectar (enlazar, secuenciar) dos ó más servicios para conseguir una funcionalidad compuesta (que no se podría alcanzar mediante la ejecución de ninguno de ellos de manera individual). En Orquestación hablamos de la manera en la que los servicios se conectan (secuencia, elección, en definitiva, las primitivas básicas que se vienen utilizando en la gestión de procesos desde hace muchos años).

Una Coreografía de Servicios es una manera de especificar de qué manera se va a comunicar el usuario del servicio con el servicio en sí (sea éste compuesto o no). De este tema surgen aspectos como patrones de intercambio de mensajes (MEP), sincronización, etc.

### 4.4.7 Reuso de Servicios:

El reuso está fuertemente ligado con la arquitectura SOA; tal es así, que es una parte central en el proceso de análisis y diseño, y también forma las bases para los modelos de servicios. El propósito del reuso es simple, tener un componente de software (servicio) disponible para más de un propósito.

El principio de reuso de servicios hace énfasis en la posición de servicios como recurso de la empresa con un contexto funcional agnóstico [ERL07]. Numerosas consideraciones de diseño deben ser formuladas para asegurar que las capacidades de

servicios sean definidas en función de que este pueda ser reusado en nuevos requerimientos.

Entre los beneficios del reuso de servicios podemos ver: ensamblar nuevos procesos de negocios desde servicios existentes, reducción de costo, mejorar el ciclo de desarrollo de una solución SOA y también reducir los riesgos teniendo código bien testeado.

### 4.4.8 Composición de Servicios:

Este principio de arquitectura se refiere al hecho de que varios servicios puedan ser ensamblados y coordinados para formar servicios compuestos. La teoría de "separation of concern" fomenta dividir un problema grande dentro de problemas mas pequeños. Esto nos da la oportunidad de construir pequeñas piezas de solución, cada una de las cuales soluciona un pequeño problema. De la misma forma podemos ver a los servicios como unidades que pueden ser compuestas para trabajar de manera coordinada en la solución de un problema mayor.

La composición de servicios esta estrechamente relacionada con la capacidad de reuso del servicio ya que la composición se puede ver como una forma de reuso. Es clave en el diseño del servicio asegurarse que los servicios puedan participar de múltiples composiciones, aún cuando el requerimiento de composición no existe.

## 4.5 Bus de Servicios

Muchas empresas que desean implementar una arquitectura orientada a servicios necesitan una infraestructura más sofisticada que soporte altos volúmenes de interacciones individuales. Tal infraestructura debería soportar niveles de calidad de servicio definidos por la empresa, políticas de seguridad. Para integrar viejas y nuevas soluciones SOA, necesitamos una infraestructura que pueda conectar cualquier recurso, cualquiera sea su tecnología y dondequiera que estos estén desplegados. Desarrollar los Web Services y exponer sus funcionalidades no es suficiente. Nosotros también necesitamos un camino para componer esas funcionalidades en el correcto orden. Los bus de servicios están surgiendo como un concepto unificado para tal infraestructura, comúnmente se los llama Enterpise Service Bus o ESB.

El concepto de ESB fue primero descrito como "una nueva arquitectura que abarca Web Services, capa de mensajes, enrutamiento y transacciones" [ESB03].

Un ESB es una infraestructura de software que simplifica la integración y flexible reuso de componentes de negocios dentro de una arquitectura orientada a servicios. Un ESB provee una infraestructura escalable que conecta aplicaciones dispares y recursos de tecnología de la información, media sus incompatibilidades y comunica sus recursos.

Actualmente se define a un ESB caracterizado por un conjunto de capacidades de infraestructura, implementado por una tecnología intermedia, que posibilita la integración de servicios en SOA. Esta centrado en la naturaleza sincrónica de los servicios y asincrónica de los eventos. Combina los paradigmas de arquitectura orientada a servicios y arquitectura conducida por eventos. Implementa interfaz

estandarizadas para proveer conectividad, comunicación, transformación, portabilidad y seguridad.

En la Figura 4.2 se observa la arquitectura de un ESB y sus componentes.

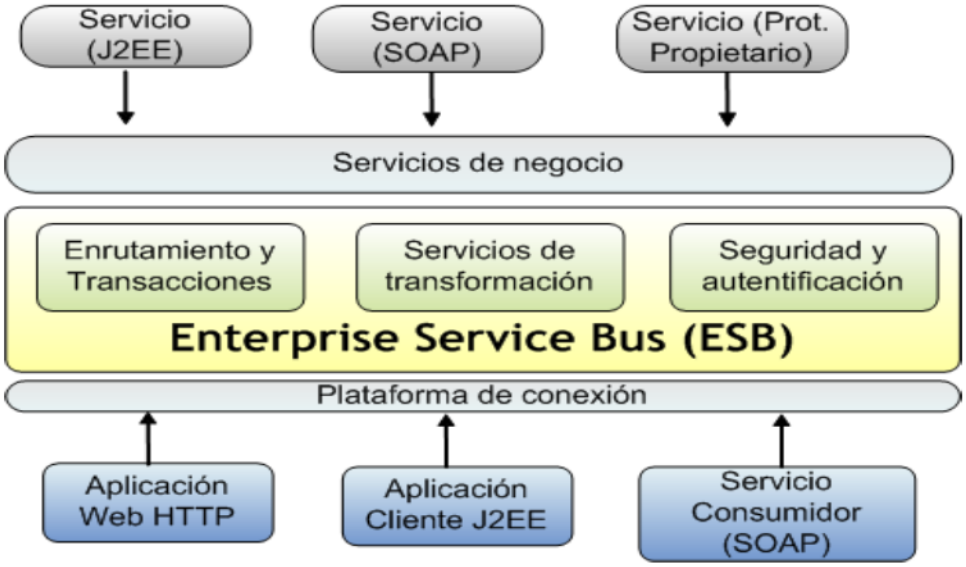



**Figura 4.2** *Bus de Servicios*

Dentro de las capacidades que debe soportar un ESB se pueden citar:

- Conectividad: el principal propósito de un ESB es proveer la conectividad a todos los participantes de una arquitectura orientada a servicios.
- Comunicación Heterogénea: un ESB debe soportar diversos modos de comunicación entre servicios (sincrónicos, asincrónicos), como así también manejar diferentes protocolos de comunicación.
- Tecnología Heterogénea: un ESB debe poder conectar participantes que están basados en diferentes lenguajes de programación, diferentes plataformas, sistemas operativos.
- Servicios de valor agregado: un ESB debe proveer servicios de autentificación, seguridad, transformación de mensajes, transacciones.
- Interoperabilidad entre múltiples y propietarios protocolos de comunicación
- Seguridad, disponibilidad y fiabilidad: un ESB debería proveer niveles de calidad de servicio configurables para asegurar que la comunicación de servicios es tan segura y fiable como se necesite para encontrar un particular requerimiento de negocio
- Flexibilidad: esto permite a una organización cambiar orquestación, reglas, mapeo de datos y relaciones entre aplicaciones con un mínimo esfuerzo.

Entre los beneficios de un ESB se pueden citar:

- Reduce tiempo y esfuerzo para integrar nuevas y existentes aplicaciones
- Crea nuevos procesos a través de existentes aplicaciones y datos
- Entrega de mensajes fiable entre servicios
- Provee una infraestructura de gestión de servicios que es altamente distribuida

- Incrementa la flexibilidad
- Puede ser desplegada incrementalmente, reduciendo el riesgo para largos y complejos proyectos

## 4.6 Web Services

Mucha gente esta familiarizada con acceder a la Web a través de un Web Browser, que provee una interfaz orientada al usuario para ofrecerle información. Cuando un usuario requiere una pagina Web, la solicitud es manejada por un servidor remoto, que retorna la información en HTML.

Los Web Services son componentes de software distribuido que proveen información a las aplicaciones a través de una interfaz orientada a la aplicación. La información es estructurada usando XML, así esta puede ser interpretada y procesada fácilmente.

De la definición de la W3C: "Un Web Services es una aplicación de software identificada por una URI, cuyas interfaces pueden ser definidas, descriptas y descubiertas por documentos XML y soportan interacción directa con otras aplicaciones de software usando mensajes basados en XML utilizando protocolos estándares sobre Internet".

Un típico Web Service esta compuesto de lo siguiente:

- Un contrato de servicio, consistiendo de una definición WSDL, un esquema XML, y posiblemente una definición de políticas. Este contrato expone las funciones publicas del servicio
- Una lógica de programación, que implementa las funcionalidades expuestas en el contrato.
- Lógica de procesamiento de mensajes que existe como una combinación de intérpretes, procesadores y agentes de servicios. Permite el intercambio de mensajes entre las partes, pudiendo adaptar distintos protocolos.

Un Web Services puede ser asociado con roles temporarios, dependiendo de su utilización en tiempo de ejecución. Por ejemplo, este actúa como proveedor de servicio cuando recibe y responde a un mensaje, pero también puede asumir el rol de consumidor de servicio, cuando este requiere un servicio a otro Web Service.

Un Web Service puede ser accesible desde cualquier tipo de cliente: navegadores, aplicación, programa, PDA, móvil, etc. La principal cualidad de los Web Services es en la comunicación aplicación-aplicación pues permite una conexión estandarizada entre sistemas y aplicaciones dispares a través de la red, favoreciendo la interoperabilidad. Se puede simplificar la integración de aplicaciones gracias a los Web Services, reducir los costes o automatizar los procesos de negocio eliminado las tediosas tareas manuales.

La diferencia fundamental es que el cliente de una página Web es una navegador mientras que un WS puede tener cualquier tipo de cliente (aplicaciones, móviles, navegadores, etc.). Realmente los WS están pensados para la interacción aplicación-aplicación.

La Figura 4.3 muestra el modelo básico de interacción soportado por Web Services.

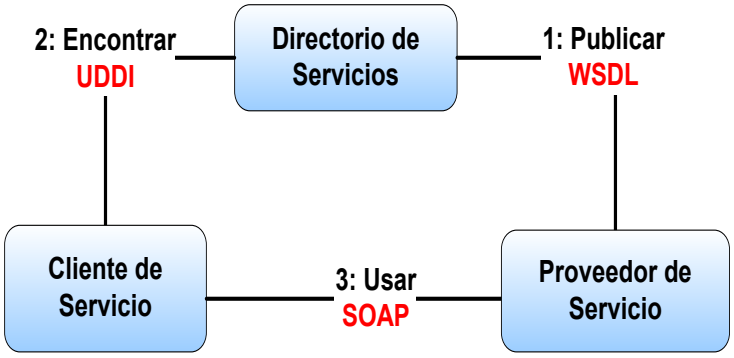



**Figura 4.3** *Modelo básico de Web Service*

Cuando un Web Service es creado, este es registrado en un repositorio de servicios publico (UDDI), al hacer esto, el proveedor expone la interfaz del servicio a cualquier cliente que este interesado en acceder a este. Cuando el cliente requiere un cierto servicio, este primero busca en un registro de servicio, es la fase de descubrimiento del Web Service. Cuando un apropiado servicio ha sido localizado, el cliente obtiene el documento WSDL (contrato del servicio) que describe la interfaz del Web Service ofrecido por el proveedor del servicio. Utilizando contrato del servicio WSDL, el solicitante enlaza el Web Service utilizando mensajes sobre protocolo SOAP.

El modelo básico se define en término de estándares SOAP, WSDL y UDDI, que son detallados a continuación.

### 4.6.1 SOAP: Simple Object Access Protocol

SOAP es un protocolo liviano, previsto para intercambiar estructuras de información, en un ambiente distribuido, descentralizado. Utiliza en XML para el envío de mensajes, define un modelo dentro del cual los mensajes pueden ser intercambiados sobre una variedad de protocolos. SOAP provee un constructor de mensaje que puede ser intercambiado sobre una variedad de protocolos, SOAP actúa como un envoltorio. Normalmente SOAP es comunicado sobre http, aunque se puede usar SMTP u otros protocolos de transporte sobre la Web. SOAP soporta llamada a procedimientos remotos (RCP). SOAP ha sido diseñado para ser simple, extensible, independiente de cualquier modelo de programación.

Un mensaje SOAP es formado por un envoltorio SOAP y cero o más adjuntos, estos últimos son opcionales y permiten incluir datos tales como una imagen JPG.

El envoltorio esta compuesto de un encabezado y un cuerpo. El encabezado es opcional, pasar información que no es de sobrecarga para la aplicación. Tal información de control incluye, por ejemplo, pasar directivas relacionadas al procesamiento del mensaje, puede contener información de contexto tal como seguridad o información relacionada a una transacción. El cuerpo del mensaje es obligatorio y esta formado por uno o mas bloques, cada bloque contiene datos de la aplicación o métodos para llamada a procedimientos remotos o puede contener información de error y control de estado.

### 4.6.2 WSDL: Web Services Description Language

Lenguaje de descripción de Web Services, permite especificar en XML las operaciones y tipos de datos de un Web Services. Un documento WSDL Cumple la función de contrato del servicio. Es una descripción estándar de un servicio, esto permite automatizar la comunicación entre aplicaciones.

Sobre WSDL un Web Service es descrito como un conjunto de puntos de comunicación que son capaces de intercambiar mensajes. Esos puntos de comunicación son llamados puertos. Un punto de comunicación esta dividido en dos partes, la primera es la definición abstracta de las operaciones y mensajes. La segunda, son los enlaces concretos a las operaciones definidas para definir protocolo de red e intercambio de mensajes. Un concreto ejemplo es SOAP sobre http.

Un Web Services es registrado en UDDI, se publica el documento WSDL del servicio que puede ser encontrado por un potencial cliente. Con el WSDL, un programador puede, por ejemplo, generar objetos y aplicaciones que invoquen y usen las capacidades que ofrece el servicio.

El valor clave de un documento WSDL como una descripción estándar de Web Services es que este posibilita la automatización de los detalles de comunicación entre dos aplicaciones. Una maquina puede obtener desde un documento WSDL que servicios están disponibles y como invocarlos sin ningún pre-acuerdo o configuración previa.

### 4.6.3 UDDI: Universal Description Discovery and Integration

UDDI define un esquema para publicar y descubrir información acerca de Web Services. Un registro UDDI es esencialmente un directorio online habilitando a los proveedores de servicios publicar lo que ofrecen y permite a los consumidores de servicios encontrar servicios que concuerden con sus necesidades. Provee “paginas blancas” listando los proveedores de servicios y “paginas amarillas” conteniendo los servicios que se ofrecen; y información técnica necesaria para acceder al servicio

SOAP puede ser utilizado para realizar consultas a un repositorio UDDI y obtener información de los servicios deseados o locaciones de documentos WSDL.

Directorios públicos UDDI son provistos por Microsoft, IBM, SAP.

### 4.6.4 SOA y Web Services

La mayoría de los desarrolladores piensan generalmente que Servicios Web y SOA son sinónimos. Hay también muchos que piensan que no se pueden hacer aplicaciones SOA sin usar Web Services. Para aclarar, SOA es un principio de diseño y los Web Services una implementación de esta tecnología. Se puede construir una aplicación SOA sin usar Web Services, por ejemplo, usando otra tecnología tal como RMI.

Se describió la Arquitectura Orientada a Servicios sin utilizar Web Services y así mismo al describir los Web Services. Esto es porque ellos son ortogonales: SOA es un estilo de arquitectura, mientras que los Web Services son una tecnología de implementación, los dos pueden ser usados en conjunto pero no son mutuamente dependientes [TREA05].

Por ejemplo, en un sistema distribuido, podemos definir una arquitectura basada en SOA, donde los servicios sean procesos individuales que se comunican usando canales locales. Similarmente, al implementar Web Services es ventajoso en la mayoría de los casos emplear los principios de SOA, aunque no hay nada en su definición que requiera de esto.

Es importante ver y posicionar a SOA como un modelo de arquitectura que es agnóstico a cualquier tecnología. Haciendo esto, una empresa esta dando la libertad para seguir las metas de estrategias asociadas con la computación orientada a servicios. En el mercado actual, la plataforma mas asociada con la realización de SOA es Web Services [ERL07].

# 5. ARQUITECTURA PROPUESTA DE SERVICIOS GESTION DE POLÍTICAS

## 5.1 Introducción

Muchos de los sistemas analizados en 2.5 utilizan lenguajes propios [SPL99][PNDR00][EIBM0] para definir las reglas que definen las políticas, ningún trabajo plantea gestionar políticas independientemente del tipo de recurso administrado y la mayoría basa su diseño en un modelo básico cliente-servidor, sin utilizar una arquitectura orientada a servicios.

El modelo planteado en esta tesis tiene como objetivo definir un esquema de negociación de políticas sobre una arquitectura orientada a servicios, que permita negociar diferentes tipos de políticas para administrar cualquier tipo de aplicacion o recursos.

En este capítulo, se define una arquitectura de servicios de gestión de políticas tomando como base la arquitectura de gestión de recursos mediante políticas definido por el grupo de trabajo del IETF, permitiendo negociar políticas de una manera independiente del tipo de políticas involucrado, utilizando el protocolo COPS para la comunicación entre un cliente y un proveedor de políticas y aprovechando sus ventajas para lograr manejar diversos tipos de políticas.

Al final de este capítulo se muestra la implementación de la arquitectura propuesta para negociar políticas de calidad de servicio sobre los Web Services que requieren utilizar las aplicaciones a fin de que garanticen ciertos parámetros de calidad sobre el uso del servicio, como por ejemplo performance.

## 5.2 Políticas en SOA

Una importante diferencia entre el paradigma orientado a objetos y una arquitectura orienta a servicios es el uso de políticas. Si una interfaz o contrato en la arquitectura orientada a servicios, separan la especificación de la implementación, las políticas separan la especificación dinámica de la especificación estática o semántica.

Las políticas representan las condiciones de disponibilidad de la especificación semántica para los consumidores del servicio. El principal aspecto de políticas es que éstas pueden ser actualizadas en tiempo de ejecución y que éstas son externas a la lógica de negocio. Las políticas expresan propiedades dinámicas como seguridad (encriptación, autentificación, etc.), auditoría, calidad de servicio, privacidad, SLA etc.

Comunicaciones basadas en políticas enriquecen el mantenimiento y adaptación de soluciones dentro de una arquitectura orienta a servicios. El hecho que varios aspectos como seguridad, monitoreo, etc. sean configurables, liberan algunas responsabilidades al equipo de desarrollo.

En la Figura 5.1 se puede observar como un modelo orientado a servicios hace utilización de diferentes modelos entre los cuales se presenta un modelo de políticas.

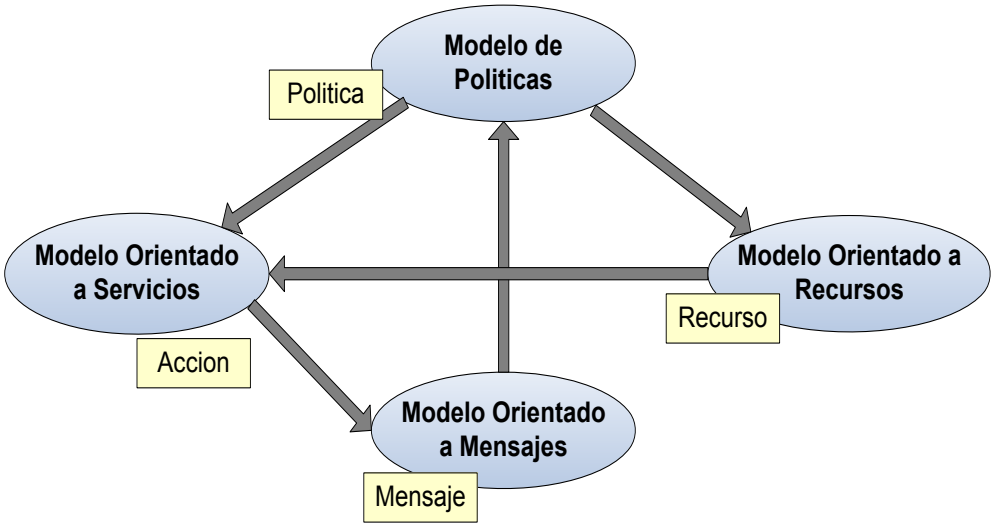



**Figura 5.1** *Modelos SOA*

- El Modelo de Mensajes define mensajes en términos de su contenido (cabecera y cuerpo), método de transporte y agentes de ejecución.
- El Modelo de Recursos define recursos (implementaciones) en términos de URI, representación y organización propia.
- El Modelo de Políticas define políticas en términos de sus sujetos (recursos y acciones) y organización, soportando políticas.

Un servicio es un mecanismo para habilitar acceso a una o mas capacidades o recursos, donde el acceso es provisto usando una interfaz prescripta y su operación consistente con restricciones y políticas especificadas por la descripción del servicio. Un natural punto de contacto entre los participantes del servicio y políticas asociadas al servicio está en la descripción del servicio.

En una arquitectura orienta a servicios el comportamiento de los servicios está gobernado por políticas que son externas al servicio en si mismo.

Al hacer uso de un Modelo de Políticas como parte de una arquitectura orientada a servicios se definen los siguientes componentes dentro del contexto de ejecución del servicio: Servicio, Punto Final, Mensajes, Contratos, Políticas y Consumidores del Servicio. También se definen las interacciones que los componentes pueden tener. La Figura 5.2 muestra los componentes y sus relaciones.

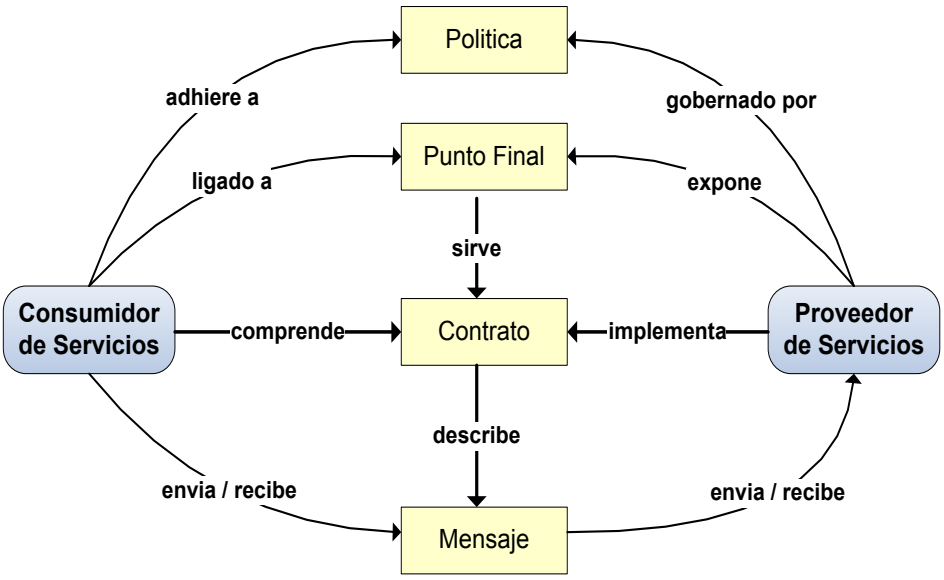



**Figura 5.2** *Componentes de una arquitectura orientada a servicio*

Además de los componentes básicos definidos en el capítulo de SOA como Servicio y Mensajes, tenemos el Contrato que el servicio implementa, el Punto final, donde el servicio puede ser accedido o contactado; Mensajes que son enviados entre el consumidor y el servicio; Políticas que gobiernan el servicio y a las cuales el consumidor adhiere.

Si nosotros miramos todos los componentes en una arquitectura orientada a servicios, podemos ver el énfasis en su arquitectura está puesto en su interfaz; de los componentes que hacen el servicio, cuatro son considerados parte de su interfaz. Entre ellos las políticas que gobiernan el comportamiento del punto final. Este énfasis en su interfaz, es lo que da a SOA muchos de los beneficios arquitecturales.

### 5.2.1 Punto Final

Representa el punto donde el proveedor del servicio expone el acceso al mismo. Este puede ser una dirección, una URI, un lugar específico donde el servicio puede ser encontrado. Las aplicaciones del consumidor del servicio están ligadas a este punto final para obtener los resultados esperados por las capacidades que el servicio expone en su contrato.

### 5.2.2 Políticas y Contratos

Una política representa alguna restricción o condición en el uso, desarrollo o descripción de una entidad definida por cualquier participante. Un contrato, por el otro lado, representa un acuerdo de dos o más partes. Como las políticas, los acuerdos hacen referencia acerca de las condiciones de uso del servicio; ellos también restringen el efecto esperado al usar un servicio. El modelo de referencia está enfocado primariamente en el concepto de políticas y contratos aplicados a los servicios.

Contrato:

La colección de todos los mensajes soportados por el servicio es conocida como el contrato del servicio. El contrato puede ser considerado la interfaz del servicio semejante a la interfaz de un objeto en OOP

Así como una política está relacionada con el punto de vista de los participantes individuales, un contrato representa un acuerdo entre dos o más participantes. Como las políticas, los contratos pueden cubrir un amplio rango de aspectos del servicio: acuerdos de calidad de servicio, interfaz, coreografía, acuerdos comerciales. Notar que no necesariamente se refiere a acuerdos legales.

Política del Servicio:

Conceptualmente hay dos aspectos de políticas: aserción de políticas y la aplicación de políticas.

Aserción de políticas se refiere a la manera en que el servicio es realizado, muestra la relación entre el servicio y su contexto de ejecución.

Una política debe ser aplicada. Las técnicas para la aplicación de políticas dependen de la naturaleza de la política. Conceptualmente, el servicio de aplicación de política asegura que una aserción de política es consistente con el mundo real.

### 5.2.3 Contexto de ejecución

El contexto de ejecución de la interacción de un servicio es el conjunto de entidades, procesos, políticas y acuerdos que son parte de un servicio.

La descripción del servicio contiene información que puede incluir protocolos preferidos, semántica, políticas y otras condiciones que describen como un servicio puede y debe ser usado. Los participantes (proveedores y consumidores) deben aceptar y conocer un consistente conjunto de acuerdos en orden de tener una exitosa interacción del servicio. El contexto de ejecución es la colección de estos acuerdos.

El contexto de ejecución es central en muchos aspectos de la interacción de un servicio. Éste define, por ejemplo, un punto de decisión para aplicación de políticas relacionadas al servicio.

## 5.3 Modelo básico de gestión de políticas del IETF

Tomando como base la arquitectura de gestión de recursos mediante políticas definida por el grupo de trabajo del IETF [REF], que cuenta con un punto de aplicación de políticas y un punto de decisión de políticas como componentes principales de la arquitectura, donde la comunicación entre los puntos está basada en un modelo cliente/servidor, es que se plantea extender el modelo a una arquitectura de servicios para que permita definir servicios de negociación de políticas.

### 5.3.1 Extensión del modelo utilizando servicios.

En primer lugar se modifica la comunicación sincrónica cliente/servidor a una comunicación distribuida utilizando servicios que permita un menor acoplamiento entre el punto de aplicación de políticas y el punto de decisión de políticas.

Al modelo de gestión de políticas definido por el IETF se agregaron los servicios de aplicación de políticas y servicio de decisión de políticas. En la **Figura 5.3** se muestra la extensión de la arquitectura definida por el IETF y los servicios definidos

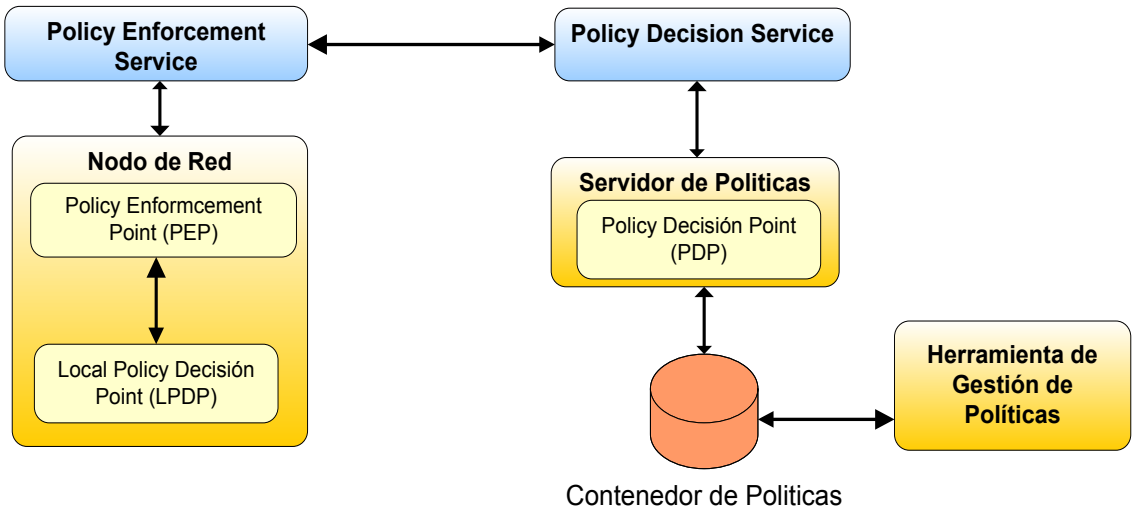



**Figura 5.3** *Modelo IETF extendido*

Policy Enforcement Service (PES): representa el servicio de aplicación de políticas mediante el cual la entidad PEP envía solicitudes de políticas al PDP y aplica o ejecuta las políticas recibidas del PDP. Este servicio permite al PEP acceder al PDP de una manera flexible, automatizada.

Policy Decision Service (PDS): servicio de decisión de políticas mediante el cual la entidad PDP es responsable de generar decisiones de políticas basadas en los estados de la red y los requerimientos enviados por el PEP. Este servicio da lugar a que el PDP este disponible dentro de una arquitectura distribuida y pueda ser accedido por los clientes PEPs utilizando su servicio PES desde cualquier ámbito, con esto se logra un acoplamiento débil entre el PEP y el PDP.

En la arquitectura definida por el IETF la información intercambiada entre el PEP y el PDP requiere un protocolo de gestión; el más usual es el protocolo COPS definido por el IETF. En la extensión del modelo a una arquitectura de servicios, la comunicación en un nivel superior se puede implementar utilizando SOAP y dentro del mismo incluir el protocolo COPS para intercambiar la información de políticas entre en PEP y el PDP.

Una de las principales características que se busca al definir una arquitectura orientada a Servicios es lograr un débil acoplamiento entre cliente y servidor. Para esto se definió una arquitectura de servicios de políticas según se ve en la **Figura 5.3** y se incluye un

Policy Service Discovery **(Figura 5.4)**, que permite implementar un servicio para registro y descubrimiento de servicios de políticas, a través del cual el PES puede consultar por servicios de decisión de políticas de acuerdo a los recursos que administre.

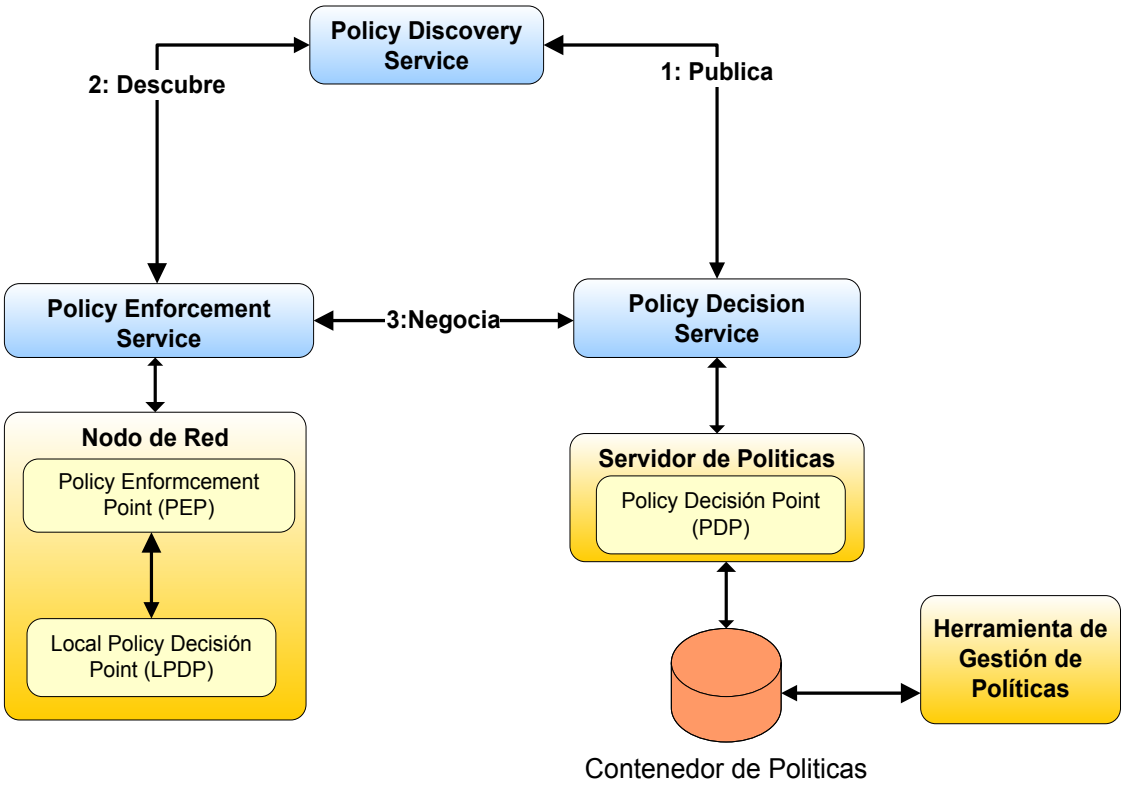



**Figura 5.4** *Arquitectura de Servicio de Gestión de Políticas*

La secuencia de interacción entre los componentes se explica a continuación:

1. El Policy Decision Service registra su servicio de decisión de políticas al Policy Service Discovery mediante el envio del contrato del servicio. Ésto le da un punto de acceso al PDP.

2. El Policy Enforcement Service, ante un requerimiento de aplicación de políticas en su PEP para administrar un recurso (por ejemplo calidad de servicio de la red), consulta al Policy Service Discovery por un servicio de decisión de políticas donde se definan las políticas requeridas (en este caso calidad de servicio).

3. El Policy Enforcement Service con los datos obtenidos del Policy Decision Service, comienza la negociación de políticas. Primero se conecta al servicio de decisión de políticas PDS para obtener acceso a su PDP, envía un mensaje requiriendo administración de recursos en su red (calidad de servicio), el PDP envía la decisión de políticas al PES mediante el PDS, si el PEP envía la información de acceso a su servicio PES, entonces el PDP puede enviar políticas no solicitadas o consultar estado de los recursos para control y monitoreo de la red que administra.

En el esquema se presentó la negociación entre un PEP y un PDP utilizando servicios de políticas. La arquitectura permite al PDP manejar varios PEP al mismo tiempo.

Aprovechando las ventajas de la arquitectura distribuida podemos adaptar el Policy Decision Service para contar con un servicio de políticas, que posea soporte para diversos tipos de clientes (calidad de servicio, seguridad) como se ve en la **Figura 5.5.**

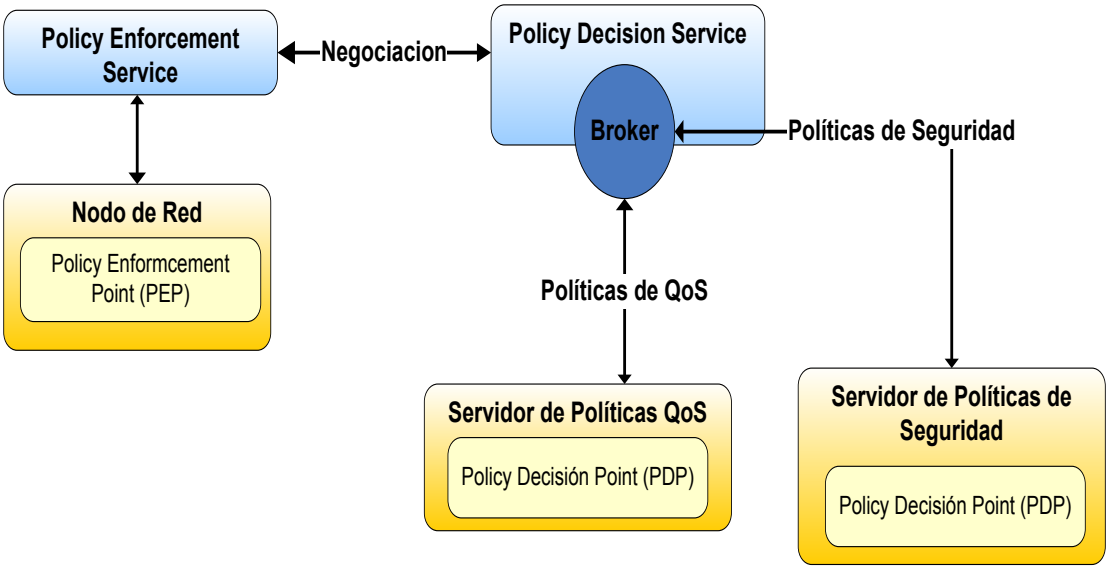



**Figura 5.5** *Enlace de servicios de políticas*

En la arquitectura se agrega un componente más denominado *Service Broker o Broker*. Es un intermediario entre el proveedor y el consumidor del servicio. Cuando el PDS recibe una petición del PES, en lugar de invocar directamente el servidor de políticas requerido, accede a través de un *Broker.* El rol fundamental del *Broker* es determinar el servidor de políticas que requiere el consumidor y enlazar la petición a tal servidor. El *Broker* provee un menor acoplamiento y más flexibilidad y da la posibilidad de realizar alguna función extra de transformación de datos.

Si el modelo plantea usar un bus de servicios dentro de la arquitectura, es útil definir un componente de decisión de políticas dentro del bus, el cual puede controlar el uso del bus mediante políticas definidas para cada componente.

En la **Figura 5.6** se observa un bus de servicios el cual es controlado por un servicio de decisión de políticas.

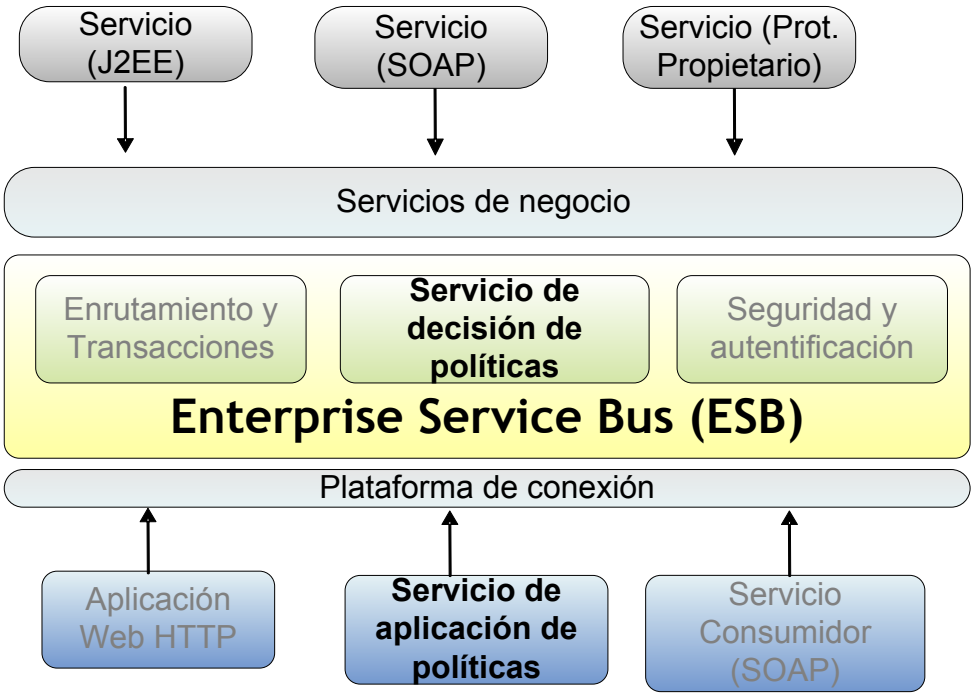



**Figura 5.6** *Gestión de políticas en bus de servicios*

Los bus de servicios ofrecen varias funcionalidades extras dentro de una arquitectura de servicios, como puede ser seguridad, encriptación, enrutamiento de mensajes. Si se define un modelo de políticas para el uso del bus, el servicio de decisión de políticas formara parte del bus y controlara el uso del mismo. Estas políticas definirán las funcionalidades que el bus le proporciona a cada componente o servicio que se conecte, por ejemplo, en el caso de las aplicaciones Web que usen el bus podemos tener una política que defina que solo se le da acceso a las funcionalidad de seguridad que ofrece el bus; Cada vez que una nueva aplicación se conecta para uso del bus, mediante una negociación se le informa las políticas de uso del bus (seguridad) y los servicios a los cuales se le permiten acceder.

### 5.3.2 Inclusión de protocolo COPS en la arquitectura.

En el punto anterior se extendió la arquitectura definida por el IETF para la gestión de políticas utilizando servicios de políticas. La información intercambiada entre el PEP y el PDP requiere un protocolo de gestión, una variedad de protocolos pueden ser usados; el más usual es el protocolo COPS definido por el IETF.

En este punto se presenta la inclusión del protocolo COPS en tal arquitectura a fin de utilizarlo para el intercambio de información de políticas entre los diferentes componentes de la arquitectura y aprovechar la secuencia de mensajes que define el protocolo COPS para la negociación de políticas.

La **Figura 5.7** muestra la secuencia de mensajes dentro de la arquitectura.

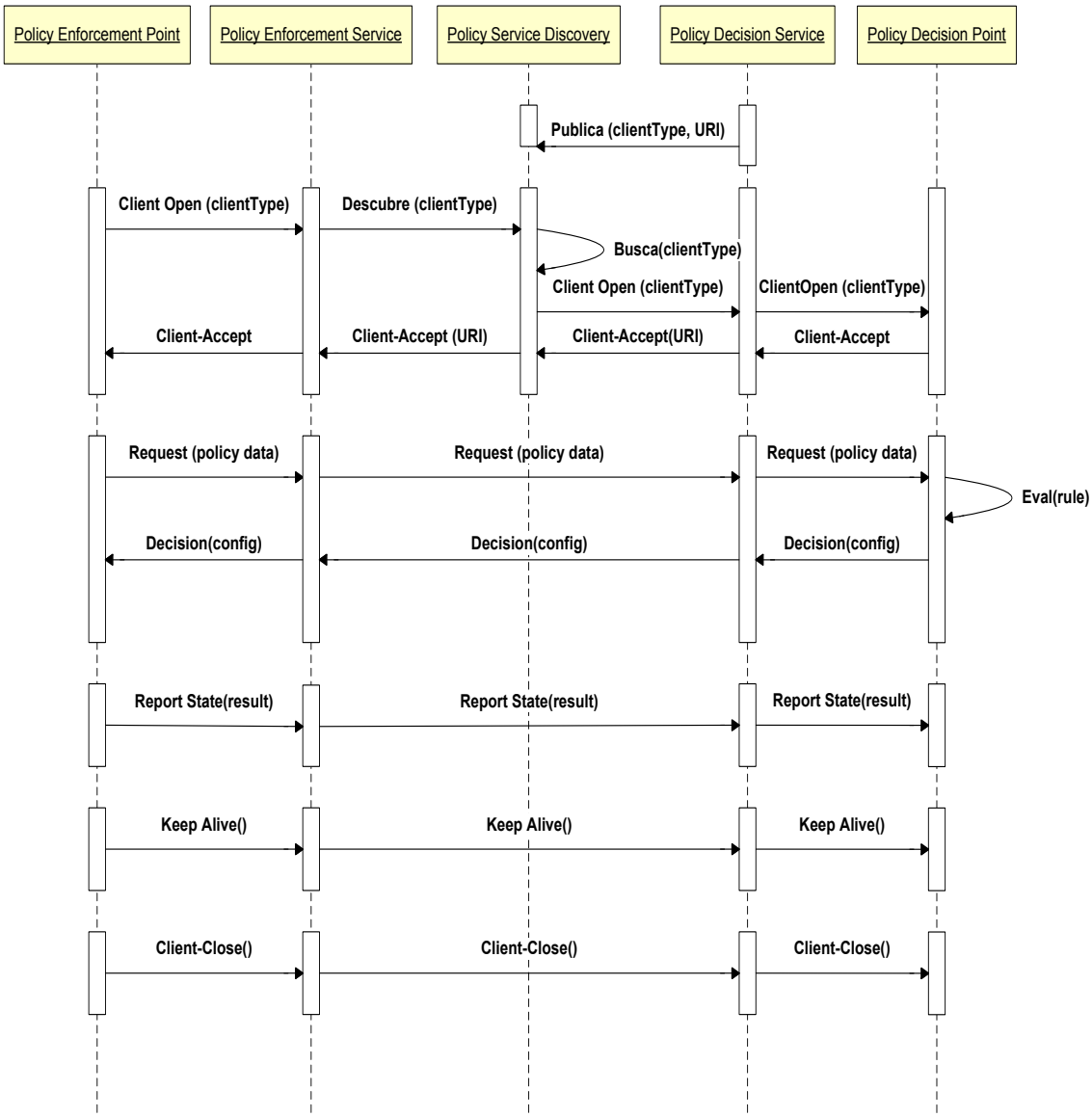



**Figura 5.7** *Secuencia de mensajes COPS*

A continuación se explica la secuencia de mensajes y las fases que permiten una negociación de servicio de políticas.

**Inicialización:**

`Publica`: El Policy Decision Service registra su servicio de decisión de políticas al Policy Service Discovery, entre la información enviada por el PDS para registrar sus servicios de políticas, se debe incluir información necesaria para el funcionamiento del protocolo. Entre estos datos deben figurar en punto de acceso al servicio (URI), el tipo de políticas que el servidor gestiona o sobre las cuales toma decisiones. Este dato representa el tipo de cliente COPS dentro del protocolo.

`Descubre`: El punto de aplicación de políticas (PEP) requiere conocer cual es el servicio de decisión de políticas al que debe acceder para gestionar los recursos que administra.

El Policy Enforcement Service consulta al Policy Service Discovery por un servicio de decisión de políticas donde se definan las políticas requeridas (por ejemplo manejo de calidad de servicio), para ello se realiza la siguiente secuencia de mensajes COPS:

El PES envía un mensaje COPS Client Open (OPN) donde figure el tipo de cliente (clase de política) para el cual se requiere el servicio. El PSD consulta los servicios disponibles cual gestiona políticas para el tipo de cliente solicitado, si se registro algún servicio para ese tipo de cliente, le reenvía al mismo el mensaje a fin de abrir una cesión COPS para negociación de políticas entre el PEP y el PDP.

Si el servicio responde con un mensaje COPS Client Accept, el PSD le envía el mensaje al PES para notificar al PEP que esta disponible el servicio de políticas solicitado, incluyendo los datos para acceder al mismo (URI).

En caso de no contar con el servicio solicitado, el PDS envía un mensaje COPS Client Close para terminar la cesión ya que no se cuenta con el servicio solicitado. Este mensaje también se puede enviar en caso de que el PDP no quiera iniciar una cesión con el PEP que solicita el servicio.

**Negociación:**

`Negocia`: en caso de que la inicialización COPS haya resultado exitosa e Policy Enforcement Point, con los datos obtenidos del Policy Decision Service, esta en condiciones de aplicar las políticas a los nodos que administra utilizando el Policy Decision Point con el cual inicio sesión COPS.

Ante un requerimiento de una entidad conectada al PEP, este envía una solicitud de políticas al PDP a través de su interfaz de servicio PES mediante un mensaje COPS Request (REQ), en el mismo se adjunta la información que especifica el requerimiento solicitado, por ejemplo en caso de manejar calidad de servicio en una red se puede enviar el ancho de banda que se requiere.

El PDP, elabora una decisión aplicando reglas de políticas que hayan sido configuradas o estén en su contenedor de políticas, para luego enviar un mensaje COPS Decision como respuesta a la solicitud del PDP.

**Reporte de estado:**

Para un mejor control de los recursos, el PEP debe enviar un mensaje COPS Report State informado el estado de las configuraciones que hayan sido negociadas por los servicios; además sirve como control y monitoreo del sistema que se esta gestionando.

A fin de conocer a ambos que el servicio con el cual iniciaron sesión aun esta disponible el cliente debe enviar el mensaje Keep-Alive dentro de un periodo de tiempo estipulado por el servidor.

**Finalización:**

La sesión puede ser culminada por ambas partes, en caso de que el PEP no gestione más los recursos, debe enviar al PDP un mensaje COPS ClientClose al PDP. El PDP también puede enviar el mismo mensaje el PEP para informarle que ya no ofrece más su servicio de decisión de políticas. En tal caso también puede dar de bajas su servicio en el PDS.

**Mensajes utilizados**:

Para la comunicación de políticas en el modelo propuesto mediante el protocolo COPS se utilizan los siguientes mensajes COPS:

1) Client-Open (OPN)
2) Client-Accept (CAT)
3) Client-Close (CC)
4) Request (REQ)
5) Decision (DEC)
6) Report State (RPT)
7) Delete Report State (DRQ)
8) Keep-Alive (KA)

Los objetos utilizados son los especificados por la norma [COPS01] para cada mensaje, incluyendo la información particular de políticas en el objeto ClientSI según el lenguaje y tipo de políticas utilizado.

En caso de requerir autentificación y seguridad se utiliza el objeto Integrity dentro de cada mensaje, pero esto puede ser manejado por la capa de servicios.

## 5.4 Diseño orientado a objetos para implementar COPS

La mayoría de las implementaciones del protocolo COPS trabajan definiendo datos binarios para representar mensajes y objetos según la norma. Esto genera un diseño mas complejo de implementar ya que cuando se recibe un mensaje COPS, tanto en el servidor como en el cliente, se debe realizar una lectura bit a bit para interpretar cada uno de los objetos y campos que componen el mensaje. Esto también hace mas difícil la aplicación de las reglas que definen las políticas.

A fin de tener una arquitectura mas simple, aprovechando las ventajas de la programación orientada a objetos y teniendo una arquitectura definida por servicios donde se pueden transportar los objetos directamente utilizando SOAP. Se define un modelo orientado a objetos para representar los mensajes y objetos del protocolo COPS, como así también una representación de políticas.

### 5.4.1 Objetos

Los objetos son utilizados para encapsular la información que se transfiere entre las entidades PEP y PDP. Cada objeto consiste en un encabezado y el contenido (en donde encapsula la información a transmitir). La estructura del encabezado consta de tres

campos Length, C-Num y C-Type; y el contenido puede ser desde un string hasta otro objeto de algún protocolo y esta formado por n campos de 32 bits.

En la Figura 5.8 se observa el encabezado de los objetos COPS.

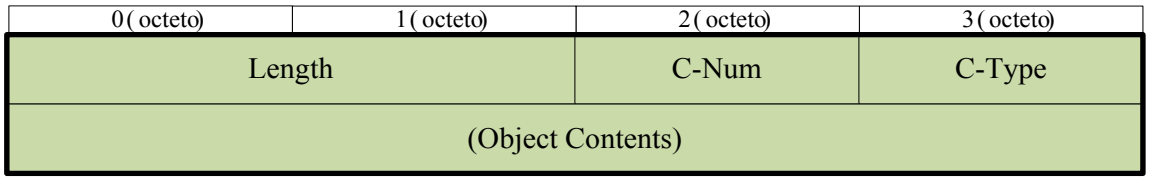



**Figura 5.8**. *Estructura de objetos*

Los campos que integran los objetos son:

- **Length:** Define el tamaño del objeto en cantidad de octetos, donde el campo se compone de dos octetos (16 bits). Tiene en cuenta la cabecera más el contenido.

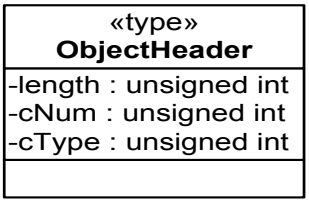
- **C – Num:** Identifica la clase de información contenida en el objeto. El tamaño del campo es de un octeto (8 bits). Los valores que puede tomar son:

  0x01 = Handle
  0x02 = Context
  0x03 = In Interfaz
  0x04 = Out Interfaz
  0x05 = Reason code
  0x06 = Decision
  0x07 = LPDP Decision
  0x08 = Error
  0x09 = Client Specific Info
  0x0A = Keep-Alive Timer
  0x0B = PEP Identification
  0x0C = Report Type
  0x0D = PDP Redirect Address
  0x0E = Last PDP Address
  0x0F = Accounting Timer
  0x10 = Message Integrity

- **C – Type:** Es un subtipo de objeto relacionado con lo especificado en el campo C-Num. El campo esta definido por 8 bits.

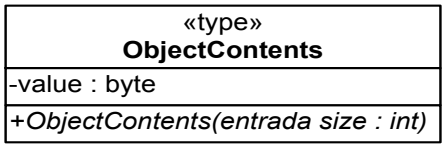
- **Object Contents:** Información que encapsula un objeto, puede ser un string u otro objeto de otro protocolo.

Todos los objetos COPS tienen un encabezado común y su contenido depende del tipo de objeto. El encabezado está compuesto de 3 campos, definimos la clase **ObjectHeader** que represente el encabezado del objeto con sus atributos:



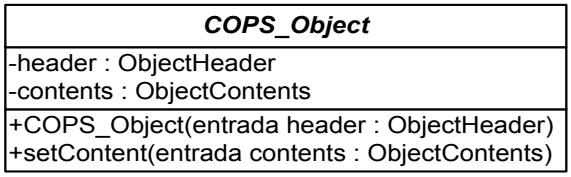

Los atributos del encabezado están definidos de acuerdo a la norma [COPS01].

El contenido de los objetos (Object Contents) no tiene un tamaño fijo, depende del tipo de objeto y del modelo de políticas que se negocie. Compuesto por n campos de 32 bits que pueden ser entero sin signo o un string o un objeto defino por otro protocolo (políticas). De la misma forma del encabezado se define una clase **ObjectContents**:




En el constructor se ingresa el tamaño del contenido, es decir, la cantidad de campos de 32 bits que lo forman.

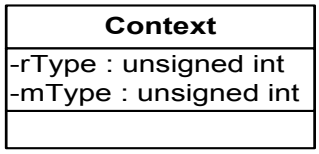
Para representar un objeto COPS en un modelo orientado a objetos se define la clase ***COPS_Object*** compuesta por el encabezado y el contenido:




***COPS_Object*** se define como una clase abstracta y cada clase que represente los objetos COPS será una subclase de esta.

El constructor de la clase recibe el encabezado como parámetro de entrada, además se define el método setContent, mediante el cual cada subclase es responsable de asignar su contenido a los atributos correspondientes.

Por ejemplo considere el objeto **Context**, para representarlo se define la clase **Context,** que extiende de ***COPS_Object***:




En el caso de la clase **Context**, el método setContent asignara el contenido del objeto a sus atributos rType y mType según esta especificado en la norma [COPS01].

### 5.4.2 Mensajes

Los mensajes se caracterizan por ser utilizados para realizar la comunicación entre las entidades PEP y PDP. Todos los mensajes se integran por objetos que permiten transportar, de una unidad a otra, la información suficiente para tomar una decisión en función de las políticas previamente establecidas. Los objetos usados dependen del mensaje.

Cada mensaje posee una función particular en la comunicación entre el PEP y PDP, esto nos lleva a que hay mensajes que solo son utilizados por el servidor (PDP) y otros por el cliente (PEP).

Además, al igual que los objetos, los mensajes se constituyen de un encabezado, que nos entrega información referida al tipo de mensaje que se ha enviado, que tamaño posee y cual es el cliente con el que se está trabajando.

En la Figura 5.9 se bosqueja el encabezado de un mensaje COPS.

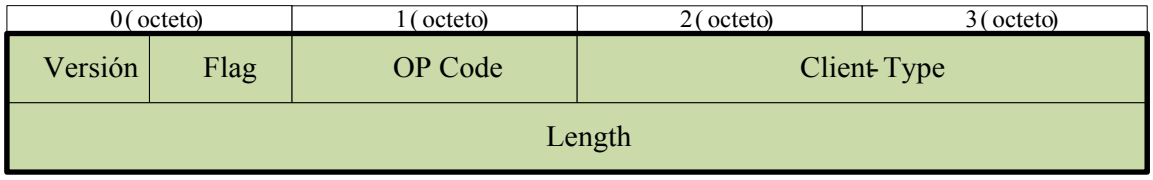



**Figura 5.9**. *Encabezado (Common Hearder)de mensajes COPS*

Cada campo que integra el encabezado se describe a continuación:

- **Versión:** Es la versión del protocolo COPS, la cual es habitualmente 1. El campo se compone de 4 bits y es el primer dato que recibe cualquier entidad.

- **Flags:** Es utilizado para saber si el mensaje corresponde a la respuesta de una solicitud, o si este es un mensaje no solicitado.

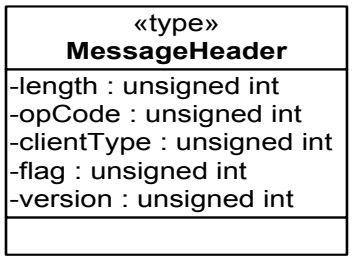
- **OP Code:** Identifica el tipo de mensajes, esto nos da una idea de los posibles objetos que pueden llegar a integrar el mensaje. El valor del campo se representa por medio de 8 bits. Los valores posibles que se pueden asignar a este campo son:

  0x01 = Request (REQ)
  0x02 = Decision (DEC)
  0x03 = Report State (RPT)
  0x04 = Delete Request State (DRQ)
  0x05 = Synchronize State Request (SSQ)
  0x06 = Client-Open (OPN)
  0x07 = Client-Accept (CAT)
  0x08 = Client-Close (CC)
  0x09 = Keep-Alive (KA)
  0x0A = Synchronize Complete (SSC)

- **Client – type:** COPS puede trabajar con distintos tipos de clientes, para identificarlos se utiliza un campo de 16 bits, en donde el rango posible de valores que es posible asignar va desde 0x8000 hasta 0xFFFF.

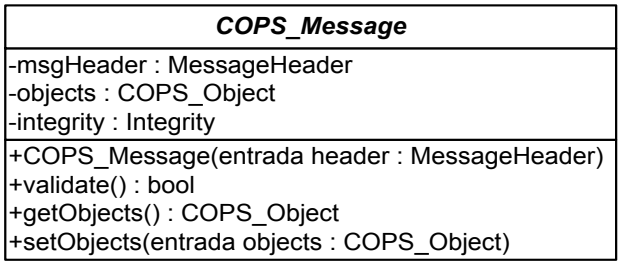
- **Message Length:** Define la longitud del mensaje en cantidad de octetos, desde la cabecera hasta el final del mensaje. La representación se la realiza utilizando un campo de 32 bits.

Como los objetos, Todos los mensajes tienen un encabezado en común, por eso definimos una clase llamada **MessageHeader**.




Cada mensaje contiene, además del encabezado, el objeto Integrity en caso de usar seguridad y un grupo de objetos, que dependen del tipo de mensaje utilizado.

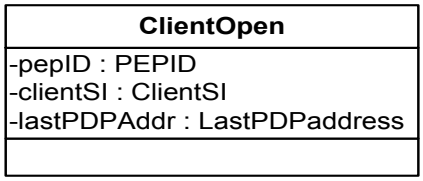
Definimos una clase abstracta llamada ***COPS*_Message**, compuesta por el encabezado y una colección de objetos ***COPS_Objects***.




El constructor de la clase recibe el encabezado donde se tiene información del tipo de mensajes, la cantidad de objetos, el tipo de cliente o políticas que negocia.

***COPS*_Message** es una clase abstracta y cada mensaje específico será una subclase de ésta, cada subclase debe implementar el método setObject que asignará los objetos recibidos a sus objetos específicos según la norma [COPS01], además el método getObject, retornando una lista de los objetos que lo componen y el método validate que retornara un booleano para indicar si el mensaje esta bien formado de acuerdo a la norma, es decir, si los objetos obligatorios fueron correctamente asignados.

Por ejemplo veamos el caso del mensaje ClientOpen representado mediante la clase **ClientOpen**:




Este mensaje contiene el objeto PEPID como obligatorio y los objetos **ClientSI** e **LastPDPAddress** como opcionales [COPS, 2001].

El método validate debe retornar true si el objeto pedID fue asignado a una instancia de la clase, en ese caso el mensaje estará bien formado y se puede proceder a procesarlo o realizar la acción que corresponda.

## 5.5 Implementación de Políticas de Calidad de Servicio sobre Web Services.

Durante los últimos 30 años investigadores han estudiado como un sistema alcanza atributos de calidad de software. Boehm y International Organization for Standarization (ISO) introdujeron conceptos de atributos de calidad [BOE78]. Bass, Bergey, Klein, y otros han analizado la relación entre arquitectura de software y atributos de calidad [BCK03].

Aproximaciones tradicionales a la calidad ponen énfasis en encontrar la especificación de requerimientos que son primordialmente funcionales, por ejemplo ISO/IEC 9126 define la calidad desde una perspectiva del usuario tales como funcionalidad, usabilidad, portabilidad, eficiencia [BEV98]. Este punto de vista deriva de ISO 8402: ¿Qué es calidad?

"Conjunto de propiedades y de características de un producto o servicio, que le confieren su aptitud para satisfacer unas necesidades explícitas e implícitas" [ISO94].

La calidad de un producto está determinada por la presencia o ausencia de esos atributos de calidad que pueden ser diseñados con el producto. [BEV98]

Uno de los aspectos más importantes de SOA es el manejo de calidad de servicio debido a la dinámica y flexible composición de SOA. La gestión de calidad de servicio optimiza los recursos del sistema a fines de satisfacer los requerimientos de calidad de servicio de muchas aplicaciones o clientes. Requerimientos de calidad de servicio se pueden dividir en varias categorías incluyendo performance, seguridad, fiabilidad.

En principio se muestra un modelo de calidad haciendo énfasis en la calidad de uso asociada al uso del software o servicio, luego se presentan parámetros de calidad que son deseados a la hora de utilizar un Web Services. Se presenta el flujo de mensajes utilizando protocolo COPS sobre Web Services para llevar a cabo la negociación de las políticas de calidad de servicio. Al final se hace referencia a trabajos relacionados comparándolos con la arquitectura propuesta en esta tesis.

### 5.5.1 Modelo de calidad

Recordando que la calidad es lo que necesitamos para la implementación y es el atributo fundamental sobre el cual vamos a definir las reglas o políticas, buscamos tratar de encontrar cuales son los requerimientos de calidad de un cliente que usa un servicio web

ISO/IEC 0126 ha sido revisada para incluir un nuevo modelo de calidad que distingue entre diferentes aproximaciones a la calidad de un producto dentro de la cual define calidad de uso por el nivel en el cual el software satisface las necesidades del usuario en el ambiente de trabajo (tales como productividad).

Para nuestro modelo en una arquitectura orientada a servicios, cuando un cliente usa un servicio, requiere que el mismo posea atributos de calidad que justifiquen su uso, como eficiencia, tiempo de respuesta, disponibilidad, etc.

ISO/IEC 9241-11 define como la calidad de uso de un producto puede ser definida, documentada y verificada como parte de un sistema de calidad conforma a ISO 9001 (**Figura 5.10**). El contexto de las necesidades de uso en el servicio necesita ser identificado, requerimientos para calidad de uso especificados, monitoreados y la calidad alcanzada evaluada. [BEV98]

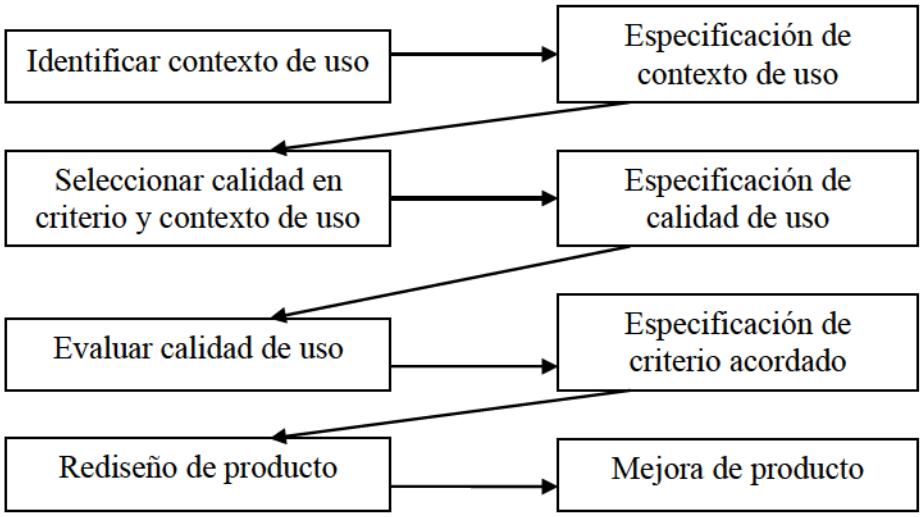



**Figura 5.10**. *Calidad de uso*

En lo que refiere a la calidad de uso de un servicio dentro de la implementación aquí planteada podemos definir:

Identificar el contexto de uso: Obtener información acerca del contexto donde el servicio es usado o el propósito del mismo, por ejemplo, identificar si el servicio es usado en sistemas de tiempo real o se requiere cierta precisión en los resultados.

Especificar la calidad de uso en los requerimientos: los requerimientos de calidad en cuanto al uso del servicio deben ser especificados en una etapa previa al desarrollo.

Monitorear la calidad de uso: durante el proceso de desarrollo se debe evaluar la calidad del software que se esta desarrollando.

Evaluar la calidad de uso: El objetivo de la calidad de uso es que el servicio pueda ser usado por otras aplicaciones y satisfaga sus requerimientos de calidad. Incrementar la calidad del servicio trae significantes beneficios.

### 5.5.2 Implementación de la arquitectura propuesta

Utilizando el modelo propuesto en la **Figura 5.4** como arquitectura para negociar políticas de calidad de servicio sobre Web Services (**Figura 3.3**) se implemento la arquitectura propuesta empleando SOAP como protocolo de comunicación y definiendo un registro UDDI del servicio definido.

En la **Figura 5.11** se muestra la secuencia de mensajes intercambiados entre los componentes para llevar a cabo la negociación.

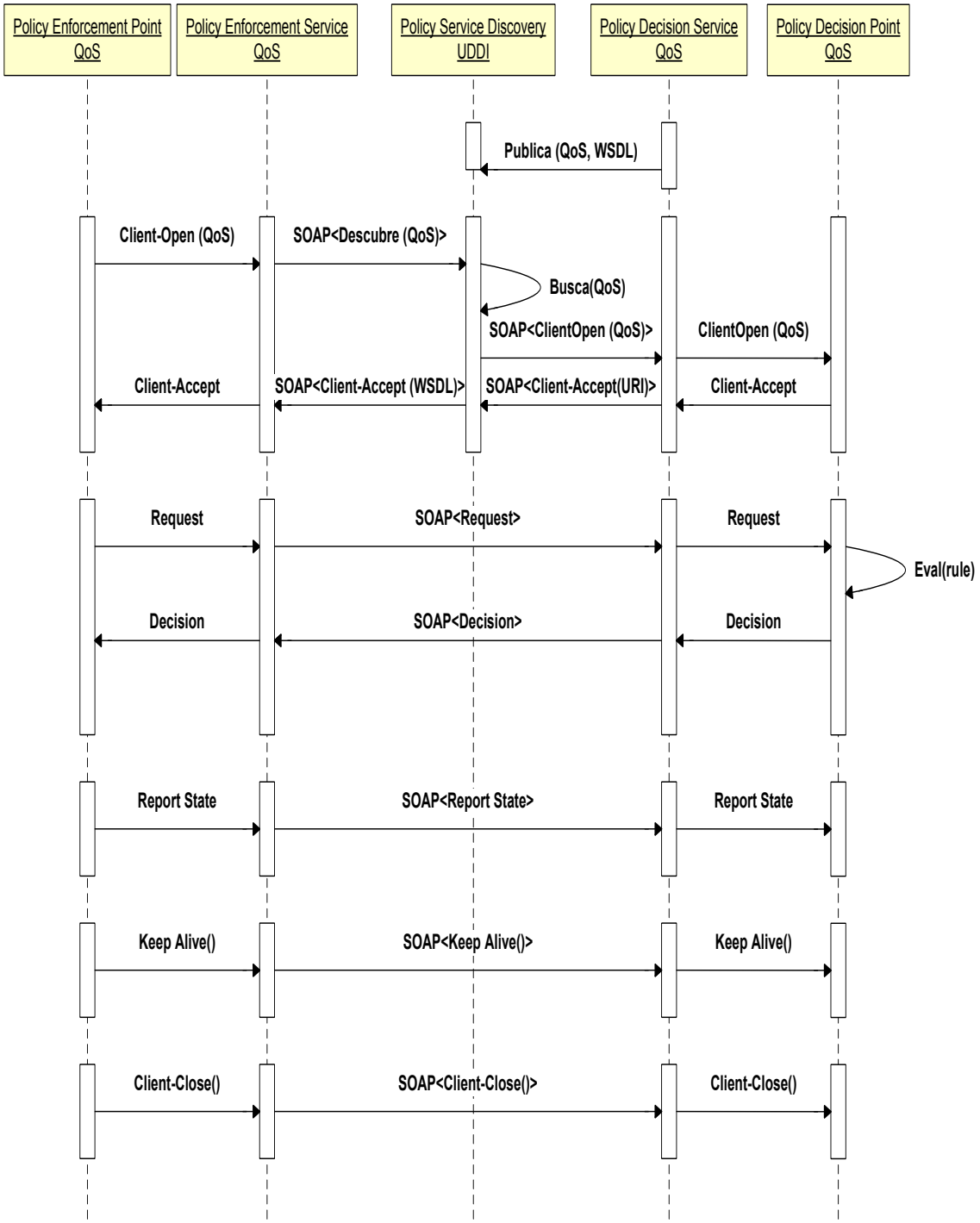



**Figura 5.11** *Secuencia de mensajes QoS*

El proceso inicia cuando el PEP recibe una solicitud de acceso a los servicios con alguna especificación en la calidad para el uso del mismo, el PEP debe acceder al PSD para buscar algún servicio que cumpla con los requerimientos del cliente y con el cual pueda negociar políticas de calidad de servicio, este envía un mensaje (SOAP <Descubre (QoS)>) al Policy Service Discovery indicando el tipo de políticas que necesita negociar. El Policy Service Discovery, selecciona el servicio de políticas que se adecue a las necesidades del cliente, en este caso el PEP requiere un servicio de políticas de QoS, el Policy Service Discovery notifica al PDP mediante un mensaje

(SOAP<Client Open (QoS)>) para iniciar la negociación de políticas entre el PEP y el PDP. En caso de no contar con un servicio de políticas solicitado, el Policy Service Discovery enviara como respuesta mensaje (SOAP<Client Close>) en el cual puede indicar otro Policy Service Discovery disponible en la red al cual puede solicitar el servicio.

Una vez seleccionado el servicio y que este acepte el cliente como valido, el PEP recibirá un mensaje (SOAP<Client-Accept (WSDL)> para culminar la inicialización e iniciar la negociación de políticas entre el PEP y el PDP a través de Web Services.

En el proceso de inicialización el PEP comunicara su punto de acceso (URIs) al PDP a fin de que este pueda enviarle mensajes con COPSDecision (DEC) en caso de que se actualice la definición de una política o se genere algún evento externo que lleve al PDP reconfigurar las políticas de sus clientes

Ya realizada la inicialización entre el PEP y el PDP y cada vez que un cliente (CLT_n) requiere un recurso de la red le envía una solicitud a su PEP local, el PEP envía un mensaje (SOAP<Request>) al PDP solicitando el recurso para su cliente, el PDP basado en el estado de los recursos de la red y en las políticas definidas en su PIB le enviara al PEP como respuesta un mensaje con COPSDecision (SOAP<Decision>) el en cual indicara el acceso o no a los recursos de la red. El PEP enviara un mensaje con COPS Report-State (SOAP< Report-State >) informando el estado del request para que el PDP pueda llevar control de los recursos.

El siguiente mensaje muestra un pedido de política de calidad de servicio desde un PEP identificado por su URI en el mensaje SOAP, en el COPSMessage se indica en el opCode que es un Request Message (REQ) y el tipo de políticas (clientType) que es de calidad de servicio (QoS). Dentro de los objetos COPS se envía el valor del handle para llevar un registro de a que configuración se asigno la política y la información especifica de cliente donde se puede incluir los parámetros de calidad de servicio que se están negociando.

```
<SOAP:Envelope      xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP:Body URI=" http://192.168.3.1/pep/dqos">
<COPSMessage opCode="REQ" flag=1 clientType="QoS">
   <COPSObject cNum=1>"Handle Value"</ COPSObject>
   <COPSObject cNum=9>" Client Specific Info"
   </ COPSObject>
</ COPSMessage >
</SOAP:Body>
</SOAP:Envelope>
```

Cabe destacar que en el campo "*Client Specific Info*" se pone la información de política solicitad en el formato deseado para la implementación, la arquitectura muestra que es independiente del tipo de políticas negociado y el formato que se represente las mismas.

La implementación del servidor de políticas definido se realizo en lenguaje C++, utilizando gSOAP [GSOAP10] para la definición del servicio de Web. gSOAP es un

Framework open source implementado en C/C++ que fue utilizado para generar el código del servidor a partir del documento descriptor del servicio WSDL, el código generado implementa el protocolo SOAP para transporte de mensajes COPS que contengan las peticiones de políticas.

El código de cliente de políticas se realizo en Java, utilizando apache AXIS [AAX10] para generar clases Java a partir del documento WSDL que interactúan a nivel de protocolo SOAP y así podes recibir del lado de cliente los mensajes COPS en respuesta a las políticas de calidad de servicio solicitadas al servidor. El entorno de desarrollo usado fue Eclipse [ECL10].

Se puede ver como a partir de un documento descriptor del servicio WSDL, se logro implementar independientemente el servidor y cliente logrando así un bajo acoplamiento dentro de la arquitectura.

El uso de COPS como protocolo de políticas para gestionar la calidad de servicio hace independiente la implementación de servicios para gestionar otro tipo de políticas, es una clara ventaja de utilizar COPS y no otro tipo de lenguaje como por ejemplo [PLB99] con el cual no hubiera sido tan transparente la adaptación para gestionar otro recurso o aplicación.

# 6. CASO DE ESTUDIO: GESTION DE LICENCIAS DE SOFTWARE

## 6.1 Introducción

En este capítulo se va a describir y estudiar una de las posibles aplicaciones de la arquitectura propuesta en esta tesis.

Muchos sistemas actuales definen sus políticas de uso mediante licencias o archivos de licencia, se definen sobre diferentes modelos de licencia y se habilitan al usuario con alguna clave o archivo enviado vía email o activando un usuario en algún servidor. Actualmente hay compañías que se encargan de ofrecer servidores dedicados a administrar y controlar las licencias para que sus clientes puedan establecer quienes están usando sus sistemas.

Dado que los modelos de gestión de licencia deben definir políticas de uso de un sistema se mostrara como utilizar el protocolo COPS sobre Web Services en un sistema distribuido de gestión de licencia. En la primera sección se describirán el contexto general de licenciamiento de software y los diferentes tipos de licencias utilizados, luego se mostrara una arquitectura de los sistemas comerciales para administración de licencias. Finalmente, tomando como referencia un sistema general de gestión de licencia, se presentara un modelo para la gestión de licencias utilizando la arquitectura propuesta en esta tesis.

## 6.2 Contexto general de los sistemas de licencia

Con el avance de las redes, la movilidad de los usuarios, el desarrollo de sistemas distribuidos y los SaaS surgieron nuevos paradigmas en el modo de gestionar y asignar licencias de software, algunos paradigmas de licencia se volvieron obsoletos y tomaron mas importancias el uso de licencias flotantes o licenciamiento dinámico de software, surgieron empresas que ofrecen servicios de administración de licencias de software mediante un servidor exclusivo o ofreciendo un sistema dedicado.

A la hora de comenzar a vender un software, muchas compañías se encuentran con el problema de como generar las licencias que administraran o permitirán el uso del mismo por parte de los usuarios, hacerlo de manera eficiente a fin de evitar el uso indebido o ilegal del mismo, requiere de mucho conocimiento, poseen un alto grado de dificultad, costo y tiempo de desarrollo. En algunas empresas hay un área dedicada exclusivamente a la administración de licencias de software, otras se ven obligadas a contratar algún software o sistema de gestión de licencias para realizar tal tarea, en el caso de que por se requiera un servidor de licencias para la habilitación del uso del software, este debe ser estable y estar disponible el 100% del tiempo, lo cual demanda una importante inversión, por lo tanto muchas empresas prefieren contratar algún servidor para gestionar las licencias del software que se ofrezca a los usuarios.

Una Licencia de Software es una autorización o permiso donde el autor otorga el derecho de usar su programa, pero habiendo determinado sus límites y derechos con respecto al uso.

Para comprender los sistemas de gestión de licencias es importante conocer las diferentes tipos de licencias usadas hoy en día por desarrolladores de software

Licencia de nodo: Es una de las mas comunes formas de licencia. En la misma se asume que el software es asignado a una computadora en particular. Se utilizan en aplicaciones de estaciones de trabajo dedicadas a ejecutar un software particular.

Licencia basada en el usuario: En este formato de licencia se asigna una licencia a un usuario particular con alguna identificación. Este tipo es usual en productos que son dependientes del usuario, tales como e-mail o transacciones de negocios.

Licencia de red: Este modelo es conocido como "licencia flotante". Se relacionan con el uso sobre sistemas distribuidos donde se adquiere un numero de licencias y se gestiona el uso de las mismas independientemente de quien ocupe una licencia (usuario o computadora). Las Licencias flotantes son asignadas dinámicamente, por ejemplo, una compañía puede comprar un conjunto de 50 licencias pero soportar una comunidad de cientos de usuarios ocasionales del software, es decir, que en promedio no mas de 50 usuarios podrán usar el sistema simultáneamente. Proveen beneficios para usuarios, administrados y vendedores:

- Usuarios pueden acceder a un recurso compartido
- Administradores pueden controlar el nodo donde las licencias estarán disponibles a través de una red
- Vendedores pueden controlar quien usa la aplicación

Muchos sistemas pueden contener atributos que restrinjan su uso. Algunas políticas aplicadas definen:

- Tiempo de inicio
- Tiempo de expiración
- Limite en la funcionalidad de la aplicación
- Licencia de prueba o evaluación

## 6.3 Arquitectura de un sistema de gestión de licencia

Los primeros gestores de licencias fueron escritos por desarrolladores de software para controlar sus propios productos. hoy en día los desarrolladores raramente escriben su propio sistema de control para las licencias del producto que ofrecen, generalmente, ellos contratan servicios de administración de licencias para licenciar el uso de sus productos

Un software de gestión de licencias es una herramienta usada por vendedores de software independientes o organizaciones para controlar donde y como sus productos de software están habilitados para ejecutarse, difieren de los software de control de copia y basan su protección en la ejecución en lugar de la copia del mismo. La administración de licencias protege a los vendedores de software contra perdidas causadas por privacidad de software o uso de copias ilegales. Sistemas de administración de licencias

ofrecen un amplio rango de modelos de licenciamiento de software, tales como activación de producto, licencias de evaluación, licencias basadas en funcionalidad del sistema o licencias flotantes.

El principal beneficio de una herramienta de gestión de licencias es que reducen la dificultad, costo y tiempo requerido y puede incrementar la transparencia con el objetivo de prevenir costos legales asociados a un robo de software.

Los sistemas comerciales tiene tres componentes bases:

- Una librería de cliente: modulo que generalmente se integra con el software a distribuir y es el encargado de comunicarse con el servidor de licencia, enviar in formación del usuario o PC requerida para habilitar el uso del software y codificar el archivo de licencia obtenido del servidor. Debe proveer una APIs de comunicación estándar para fácil integración con el software que se desea licenciar.

- Un Servidor de Licencias: Aquí reside el sistema central de administración de licencias, el mismo podría contar con una base de datos para almacenar las licencias activas y la información de los clientes o usuarios del software a gestionar, una herramienta de administración para que los clientes definan sus políticas o reglas de licencia y controlen quienes están usando su software, modulo de generación de claves de activación o archivos de licencia. En el caso de licencias flotantes el servidor puede manejar el paquete de licencias contratados y asignarlas dinámicamente a los usuarios.

- Un archivo de descripción licencia: El archivo de licencia describe los derechos de software que el usuario a contratado, información del servidor de licencia, el tipo de licencia y las políticos de uso del software (funcionalidad habilitada). Generalmente son archivos de texto cifrados para prevenir la modificación de su contenido por parte de un tercero.

### 6.3.1 Proceso de obtención de licencia

Los siguientes pasos describen el funcionamiento de un sistema de gestión de licencia:

1. El desarrollador del software integra la librería de cliente para la gestión de la licencia

2. Se definen los tipos de licencia y las políticas sobre los que se habilitara el software

3. Se configura el servidor de licencia en el cual se definen las políticas de licencias y generan los archivos o claves de activación de licencia que serán entregados junto con el software

4. Se entrega el software al usuario junto con la clave de activación o archivo de licencia. Típicamente las licencias son enviadas a los clientes en forma de texto. Hoy en día el medio mas común de entrega de licencia es Internet mediante email o activación automática en un servidor.

5. El software a través de su librería de cliente establece una conexión con el servidor de licencia a fines de requerir una licencia para el uso del software

6. Se enviar la información del usuario al servidor de licencias para almacenar en su base de datos y tener un control de quienes usan el software. El servidor controla con la información proporcionada por el desarrollado y envía la respuesta al cliente para habilitar o denegar el uso del software.

7. En caso de que se habilite el uso del software, dependiendo del tipo de licencia se puede enviar la información para controlar el tiempo de ejecución en caso de ser una versión de evaluación o las funcionalidad habilitada.

8. El desarrollador del software puede monitorear los usuarios que usan su software y cuantas licencias están activas.

9. Se puede habilitar el servidor de licencias para que administre dinámicamente las licencias flotantes

## 6.4 Modelo de gestión de licencias utilizando COPS sobre Web Services

El modelo planteado en esta tesis tiene como objetivo mostrar una arquitectura de gestión de políticas utilizando protocolo COPS sobre Web Services. En la presente sección, a fin de evaluar la arquitectura propuesta, se presenta un modelo para gestionar licencias utilizando servicios de políticas sobre Web Services.

Utilizando el modelo IETF extendido propuesto en la **Figura 5.3** como arquitectura para negociar políticas sobre Web Services en la **Figura 6.1** se implementa la arquitectura para un servicio de gestión de licencias de software empleando políticas sobre protocolo COPS.

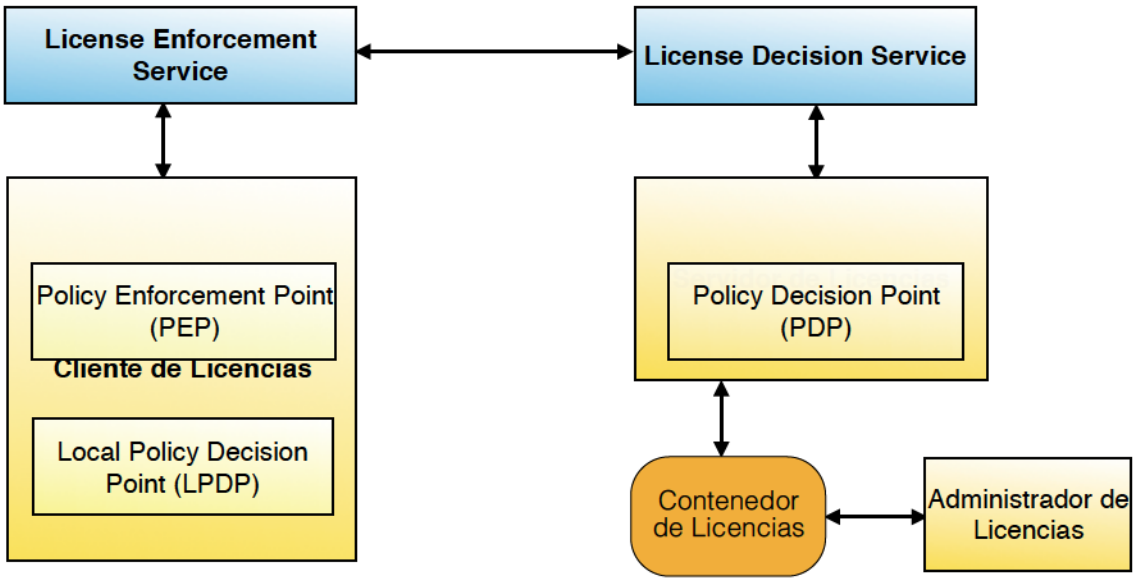



**Figura 6.1** Arquitectura de gestión de licencias

License Enforcement Service (LES): representa el servicio de aplicación de políticas de licencias mediante el cual la entidad PEP envía solicitudes de licencias al PDP y aplica o ejecuta las políticas recibidas del PDP para administrar una licencias de software.

License Decision Service (LDS): servicio de decisión de licencias mediante el cual la entidad PDP es responsable de generar decisiones de políticas de licencia basadas en información del usuario, licencias disponibles y los requerimientos enviados por el PEP. Este servicio da lugar a que las licencias estén disponible dentro de una arquitectura distribuida y puedan ser administradas por los clientes.

Contenedor de Licencias: Representa la base de datos donde se almacena toda la información de licencias que se desean gestionar, en la misma se incluye la información del usuario y las licencias disponibles.

Administrador de Licencias: Herramienta disponible a los clientes del servicio para administrar las licencias que entregan a los usuarios de su software, pueden habilitar o bloquear licencias dinámicamente, gestionar las políticas mediante las cuales las licencias serán habilitadas, controlar quien esta usando su sistema con una licencia valida.

Servidor de Licencias: Servidor central de licencias donde reside el PDP, será el encargado de crear, habilitar y validar licencias de software utilizando el PDP como punto de decisión basado en las políticas de licencia definidas. El servidor se comunicara con el cliente utilizando el servicio LDS y la información de licencias será transmitida utilizando protocolo COPS.

Cliente de Licencias: Aplicación que reside en el nodo en donde el usuario accederá al sistema y solicitara una licencia para su uso. Se encarga de acceder al LES para solicitar una licencia utilizando el servicio de políticas, enviar información de cliente requerida y permitir la comunicación entre el PEP y el PDP para el intercambio de políticas que definirán la licencia solicitada. La librería de cliente también debe proveer una APIs de comunicación estándar para fácil integración con el software que se desea licenciar.

En la arquitectura definida por el IETF la información intercambiada entre el PEP y el PDP requiere un protocolo de gestión; el más usual es el protocolo COPS definido por el IETF. En la extensión del modelo a una arquitectura de servicios, la comunicación en un nivel superior se puede implementar utilizando SOAP y dentro del mismo incluir el protocolo COPS para la negociación de licencias intercambiando políticas entre el PEP y el PDP.

### 6.4.1 Escenario de obtención y validación de licencia

La **Figura 6.2** muestra la secuencia de mensajes para la obtención y validación de una licencia en la aplicación cliente sobre los servicios de gestión de licencia utilizando políticas.

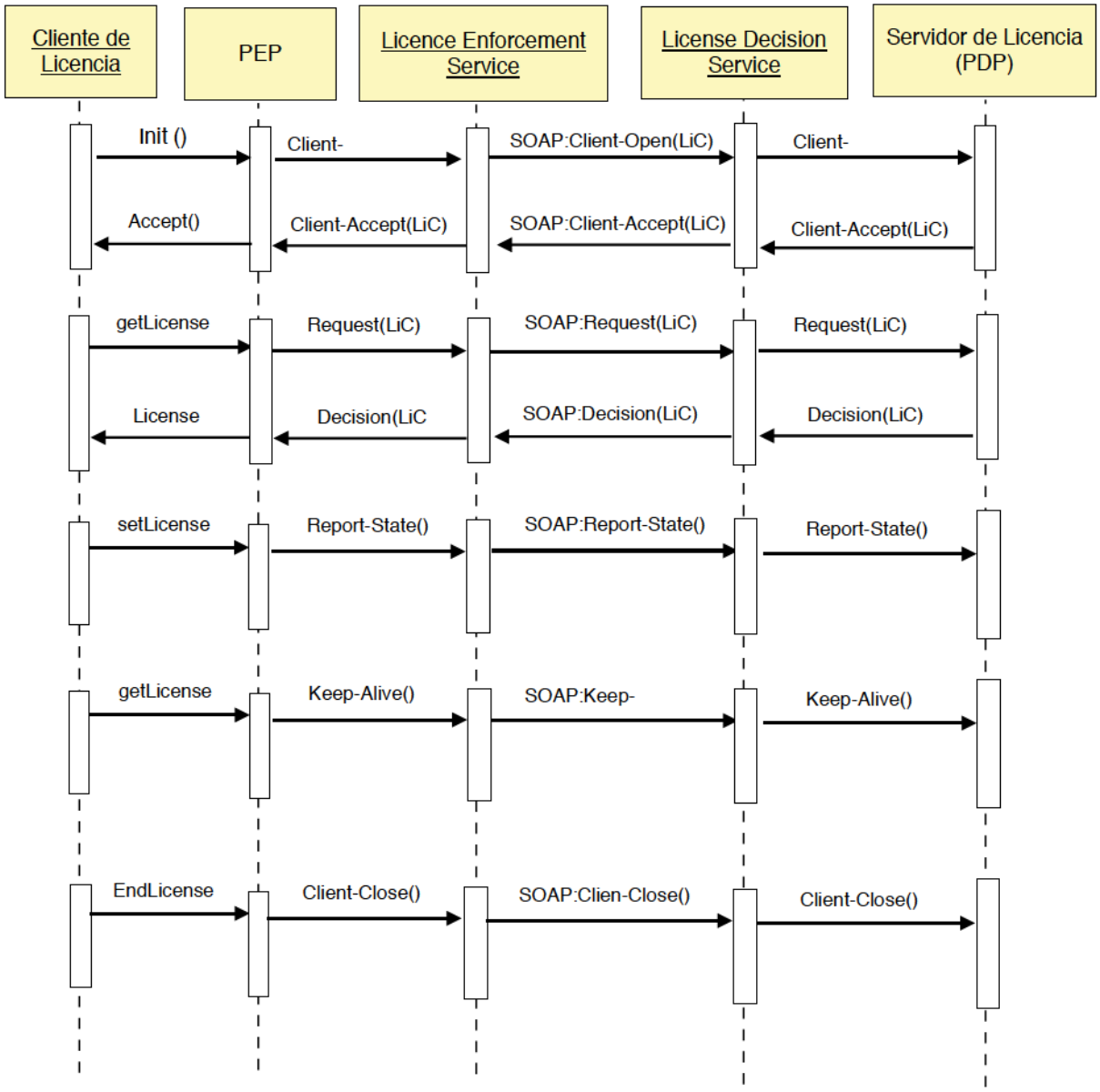



**Figura 6.2** Mensajes para intercambio de licencia

A continuación se explica el intercambio de mensajes y las fases que permiten una negociación de una licencia flotante entre la aplicación de cliente y el servidor de licencias utilizando un servicio de políticas basado en la arquitectura propuesta en esta tesis.

**Inicialización:**

El cliente de licencia inicia el pedido de una licencia flotante enviando un mensaje de *Init* al punto de aplicación de políticas PEP, el cual envía un mensaje *COPS-Client Open (OPN)* donde figure el tipo de cliente (licencia) para el cual se requiere el servicio de politicas. El License Decision Service consulta al servidor de licencias flotantes si dispone de licencias para la aplicación solicitada, en caso de haber licencias disponibles, la solicitud es enviada al PDP a fin de abrir una cesión COPS para negociación de una licencia entre el PEP y el PDP.

Si el servicio responde con un mensaje *COPS-Client Accept*, el LSD le envía el mensaje al LES para notificar al PEP que hay licencias disponibles en el servidor, incluyendo los datos que se requieren para obtener una licencia y el mensaje de *ClientAccept*.

En caso de no contar con licencias disponibles, el PDS envía un mensaje *COPS-Client Close* para terminar la cesión ya que no se cuenta con el servicio solicitado. Este mensaje también se puede enviar en caso de que el PDP no quiera iniciar una sesión con el PEP que solicita el servicio.

**Negociación de Licencia:**

En caso de que la inicialización COPS haya resultado exitosa, el Policy Enforcement Point (PEP), con los datos obtenidos del Policy Decision Service (PDP), esta en condiciones de aplicar las políticas a los nodos que administra utilizando el PDP con el cual inicio sesión COPS.

Ante un requerimiento de licencia de un cliente conectado al PEP, se envía una solicitud de políticas al PDP a través de su interfaz de servicio LES mediante un mensaje *COPS Request(REQ)*, adjuntando la información de cliente requerida para generar la licencia.

El PDP, elabora una decisión aplicando reglas de políticas de licencia que hayan sido configuradas o estén en su contenedor de políticas de licencia, para luego enviar un mensaje *COPS-Decision* como respuesta a la solicitud del PDP con la licencia solicitada por el cliente.

**Validacion de Licencia y Reporte de estado:**

Para llevar un control de las licencias flotantes en uso, el PEP debe enviar un mensaje *COPS-Report-State* informado el estado de las configuraciones que hayan sido negociadas por los servicios; además sirve como control y monitoreo del sistema que se esta gestionando.

A fin de conocer a ambos que el servicio con el cual iniciaron sesión aun esta disponible el cliente debe enviar el mensaje *Keep-Alive* dentro de un periodo de tiempo estipulado por el servidor.

**Finalización:**

La sesión puede ser culminada por ambas partes, en caso de que el PEP no gestione más las licencias o no se disponga de licencias flotantes, debe enviar al PDP un mensaje *COPS-ClientClose* al PDP. El PDP también puede enviar el mismo mensaje el PEP para informarle que ya no se utiliza la licencia obtenida a fin de liberarla en el servidor.

**Mensajes utilizados**:

Para la comunicación de políticas en el modelo propuesto mediante el protocolo COPS se utilizan los siguientes mensajes COPS:

1) Client-Open (OPN)
2) Client-Accept (CAT)
3) Client-Close (CC)
4) Request (REQ)

5) Decision (DEC)
6) Report State (RPT)
7) Keep-Alive (KA)

Los objetos utilizados son los especificados por la norma [COPS01] para cada mensaje, incluyendo la información particular de políticas en el objeto *ClientSI* según el lenguaje y tipo de políticas utilizado, en este caso el objeto *ClientSI* incluira toda la informacion de licencias.

Es recomendable utilizar autentificación y seguridad en el objeto *Integrity* dentro de cada mensaje, dado que esto puede ser manejado por la capa de servicios.

En este capitulo se vio como podemos aplicar la arquitectura propuesta en un sistema que requiere gestión, habilitación y validación de datos mediante reglas definidas como es el caso de las licencias, pudimos aprovechar las ventajas de un protocolo de políticas estándar dentro de una arquitectura orientada a servicios para gestionar de manera segura un servicio de licencias de software.

# 7. TRABAJOS RELACIONADOS

## *7.1.* Introducción

En el capítulo 5 se presentó la arquitectura de servicios de gestión de políticas integrando los conceptos que fueron desarrollados en los capítulos 2, 3 y 4. También se presentó un caso de estudio aplicando el diseño propuesto en la tesis. De esta manera se cubrieron los objetivos propuestos para este trabajo. No obstante, es importante poder revisar en este capítulo los trabajos mas influyentes que fueron de fundamental aporte para lograr el objetivo de esta tesis y los comentarios respecto a la bibliografía utilizada y su influencia o vinculación con el trabajo realizado.

## 7.2 Utilización de protocolo COPS

El protocolo COPS [COPS01] parte de la suite de protocolos definidos por el IETF en la RFC2748 presenta un diseño simple pero extensible, es por ello que mediante diferentes enfoques, existen muchas extensiones que lo utilizan como base para ofrecer una solución al intercambio de políticas.

Una de las extensiones del protocolo mas conocidas es COPS-PR [CHAN01] el cual se definió para el aprovisionamiento de políticas en COPS. En el caso de COPS-PR el PDP provee continuamente al PEP las directivas de políticas necesarias para reaccionar a eventos, el PEP le provee al PDP información acerca de sus capacidades y configuración. En el ejemplo de un firewall, el PEP puede proveer al PDP con información respecto a los servicios que este provee y su interfaz de red, el cual obtendrá respuesta con políticas apropiadas para limitar el acceso a esos servicios.

COPS-PR fue diseñado para distribuir políticas a múltiples dispositivos de red. Las políticas usadas están formadas por subconjuntos de reglas generales y son independientes del dispositivo o recurso que se administra. Las reglas generales están cargadas en una base de información de políticas llamada Policy Information Base PBI.

En este trabajo de tesis se siguió parte de los principios de diseño de [CHAN01] para lograr un sistema de gestión de políticas que sea independiente del servicio o recurso que se está administrando y contemplar una base de datos genérica para la definición de políticas. Esto representa una de las principales ventajas de la arquitectura propuesta respecto de soluciones similares como [WSP06].

COPS-PR [CHAN01] no provee servicios de seguridad para asegurar integridad de mensajes, autenticación y confidencialidad de mensajes. Tales servicios son críticos cuando hablamos de un sistema de gestión de políticas para administrar recursos o servicios. En el sistema planteado en esta tesis se contempla la solución al utilizar el objeto Integrity dentro de cada mensaje COPS.

Packetcable [PKQS02] es un buen ejemplo es el uso de protocolo COPS que permita el soporte de las funciones de DQoS (Calidad de Servicio Dinámica) según lo especificado por CableLabs (http://www.cablelabs.com). En la especificación [PKQS02] se definen por completo todos los mensajes y objetos que involucran los protocolos de

señalización y la información que se envía para cada objeto COPS.

PacketCable es un sistema de protocolos desarrollados para entregar Calidad de Servicio (QoS) relacionados con los servicios de comunicación. Es una iniciativa de CableLabs cuyo objeto es el desarrollo de especificaciones de interfaz interoperables, para la implementación de servicios avanzados multimedia en tiempo real sobre planteles de cable bidireccionales. Construido sobre la exitosa infraestructura de cable modems, las redes de PacketCable utilizan tecnología IP para habilitar una amplia gama de servicios multimedia, tales como telefonía IP, conferencias multimedia, juegos interactivos y aplicaciones multimedia en general.

CableLabs establece que éstas políticas deben ser comunicadas entre los diferentes módulos utilizando el protocolo COPS y establece el perfil COPS para este uso en particular [PKQS02].

La forma de encapsular sus objetos de protocolo en objetos COPS y los mensajes utilizados para implementar la solución de calidad de servicio fueron tomados como ejemplo para el diseño.

El proyecto PacketCable de CableLabs es un ejemplo clásico de utilización de protocolo COPS en las redes de telecomunicaciones para gestionar recursos, en este caso calidad de servicio dinámica, dado que las redes de PacketCable utilizan tecnología IP, se podría utilizar el modelo propuesto en esta tesis para administrar tanto calidad de servicio dinámica como los servicios ofrecidos (servicios multimedia, telefonía IP, etc) gestionando los mismos de manera dinámica mediante políticas definidas sobre una arquitectura orientada a servicios utilizando protocolo COPS.

Migrar el proyecto PacketCable o cualquier sistema similar que utilice protocolo COPS para gestionar recursos ofrecidos en una red de telecomunicaciones al sistema de gestión de políticas sobre Web Services propuesto en esta tesis es factible debido al diseño simple y extensible y como valor agregado da la ventaja de utilizarlo para gestionar recursos de manera mas genérica y no limitarse a un solo servicio en una red fija, dado que los mismos se pueden extender a una arquitectura distribuida.

[CHAN01] y [PKQS02] son algunos sistemas que se tomaron como base para el origen del objetivo central de esta tesis, cuyo propósito principal fue definir un sistema de gestión de políticas extendiendo el protocolo COPS sobre Web Services para eliminar algunas limitaciones de los sistemas descriptos anteriormente y lograr solución mas completa y adaptable a los requerimientos de sistemas administración de políticas.

## 7.3 Gestión basada en Políticas

Hoy en día las empresas de telecomunicaciones ofrecen una amplia gama de productos y servicios relacionados a Internet, desde ancho de banda o telefonía IP hasta servicios de streaming video o música. Existe una gran demanda de recursos y servicios disponibles y es clave poder administrarlos de manera eficiente y diferenciada, por ejemplo el caso de Spotify que ofrece streaming de música por internet provee un servicio básico gratis y uno pago donde la calidad de audio es superior. Esto se puede generalizar como calidad de servicio y una forma de administrarlo es mediante el uso de políticas.

En esta tesis el objetivo principal está centrado en un sistema de gestión de políticas sobre Web Services y uno de los trabajos relacionados tomado como referencia es el Web Services Policy Framework (WS-Policy) [WSP06], que provee un modelo de propósito general y su correspondiente sintaxis para describir las políticas en un sistema basado en Web Services. La especificación [WSP06] es implementada por IBM, BEA System, Microsoft, Verisign y Apache cuyo modelo es implementado mediante Apache Neethi [APN13].

Apache Neethi [APN13] provee una librería general para el uso de WS-Policy. Fue escrito especialmente para habilitar el uso de WS-Policy sobre los Web Services de Apache. Dentro de sus características se pueden destacar una API para procesamiento de políticas y un mecanismo para extender aserciones de políticas especificas, entre otras.

La arquitectura propuesta en esta tesis (Figura 5.4) para la gestión de servicios de políticas está basada en el modelo extendido de IETF (Figura 5.3) como protocolo de gestión para la información intercambiada entre el PDS y el PES. Como alternativa a COPS existen varias opciones que se podrían considerar y una de las mas adecuadas seria la implementación de WS-Policy de Apache [APN13]. Como COPS esta basado en el modelo de IETF y contempla en su definición de arquitectura un PDP y un PEP, es ventajoso utilizarlo como protocolo de intercambio de mensajes respecto a otras opciones sobre WS-Policy como [APN13].

## 7.4 Aplicaciones de gestión de licencias

En el capitulo anterior se presento un caso de estudio a fin de evaluar la arquitectura propuesta, el mismo modela un sistema de gestión de licencias de software mediante políticas sobre Web Services. Existen muchas aplicaciones comerciales [SLPS09] [RLM14] [NAP14] [ITAM14] [SOR14] que ofrecen a los desarrolladores o empresas una gama de productos para la administración de licencias, en general todas contemplan un servidor de licencias, funcionan en un ambiente distribuido, permiten trabajar con diferentes tipos de licencias (licencias de prueba o evaluación, licencia basada en el usuario, licencias flotantes), distribuyen las licencias remotamente y generan reportes para un mejor control del cliente, ninguna de estas soluciones esta basada en un estándar de gestión de recursos mediante políticas, es por ello que presenta una ventaja la solución propuesta definiéndola sobre protocolo de gestión de políticas y adaptando un modelo de IETF utilizado en diferentes ambientes para administrar recursos de manera segura mediante políticas para la gestión de licencias.

Las soluciones comerciales existentes fueron tomadas como referencia para entender el funcionamiento y las características que debería cumplir un sistema de gestión de licencias, con el objetivo de extender el modelo propuesto en la **Figura 5.3** para implementar un servicio de gestión de licencias de software empleando políticas sobre protocolo COPS que pueda ser una alternativa a las aplicaciones comerciales de licencia al momento elegir una solución presenta algunas ventajas respecto como utilizar COPS como un protocolo de políticas estándar para gestionar de manera segura un servicio de licencias de software.

# 8. CONCLUSIONES Y FUTUROS TRABAJOS

Dentro de un sistemas distribuidos se debe poder controlar de manera automatizada los recurso los sistemas asociados, como por ejemplo la calidad de los servicios que se ofrecen (ancho de banda en Internet) o lograr gestionar la seguridad del sistema. Un método optimo que permite resolver estos dilemas es utilizar políticas de control o gestión mediante las cuales se definen reglas mediante las cuales el nodo de control o servidor de acceso debe tomar decisiones y aplicarlas en los sistemas asociados que tienen acceso a los recursos de la red.

Actualmente se ha observado un incremento en la demanda de servicios de red, los cuales requieren de mayores recursos de los que actualmente proporcionan las redes. Esto ha marcado la pauta para el desarrollo de nuevas soluciones para el control de recursos, como es el caso de los servicios de negociación de políticas. Se han elaborado algunas especificaciones donde implementa la arquitectura de redes basadas en políticas (PBN) para control de QoS, esto con el fin de ofrecer calidad de servicio extremo a extremo.

Sistemas actuales basados en políticas definen reglas de gestión bastante limitadas para la diversidad de sistemas y clientes que utilizan los servicios disponibles; en tales aproximaciones se presentan problemas cuando se requiere gestionar negociar nuevos servicios de red o diferentes aplicaciones que usan los mismos.

El uso de políticas para la negociación o administración de recursos ha sido un área de investigación en los últimos años donde podemos encontrar varias publicaciones referidas al tema. En el análisis del estado del arte sobre lo expuesto en 2.5 se puede ver que en general no proponen un estándar de definición de políticas sino que se orientan por el uso de sistemas propietarios. Una ventaja importante a la hora de utilizar COPS para comunicar políticas es poder contar con la definición de varios tipos de políticas sobre un mismo sistema de gestión.

La mayoría de los sistemas actuales de gestión de políticas son aplicables para un único conjunto de reglas que administren por ejemplo seguridad, ningún trabajo se platea un diseño extensible en el que se manejen políticas genéricas independientemente de las reglas que regulen.

La Arquitectura Orientada a Servicios permite implementar servicios distribuidos dentro de un sistema u organización con los cuales se puede gestionar los recursos a través de negociación de políticas. El modelo propuesto mediante servicios permite tener bajo acoplamiento entre el cliente y el servidor para la negociación de políticas.

Al utilizar COPS se logra un modelo para la negociación de cualquier tipo de políticas, tanto de configuración para controlar dispositivos que usen los recursos disponibles, como políticas que permitan manejar QoS o políticas de privacidad.

El protocolo COPS centraliza el control de QoS del tráfico de datos dentro de su dominio administrativo. La filosofía orientada a objetos del protocolo hace que sea fácil agregar nuevos tipos de clientes dentro del protocolo si es necesario, porque el

contenido es transferido como objetos encapsulados dentro de los mensajes. Esto se vio en el caso de estudio donde se utilizo el protocolo para la gestión de licencias.

La gestión de recursos basada en políticas permite administrar el acceso a los recursos mediante una serie de reglas automatizadas para proporcionar clases de servicio basadas en prioridades ya sea de usuario, de servicio o de red.

Un marco de trabajo de políticas permitiría a los desarrolladores expresar las políticas de los servicios de una forma procesable por las computadoras. La infraestructura de los servicios Web puede verse mejorada para entender ciertas políticas y forzar su uso en tiempo de ejecución. Por ejemplo, un desarrollador podría escribir una política que indique que un servicio dado requiere firmas digitales y cifrado. Otros desarrolladores podrían utilizar la información de las políticas para determinar si pueden o no utilizar el servicio, y todo ello en tiempo de ejecución sin la necesidad de ninguna intervención externa adicional. La infraestructura podría imponer estos requisitos sin obligar al desarrollador a tener que escribir ni una sola línea de código. Por lo tanto, un marco de trabajo de políticas no sólo provee una capa de descripción adicional, sino que también ofrecerá a los desarrolladores un modelo de programación más declarativo.

A partir de los modelos propuestos en la tesis se prevé como futuro trabajo profundizar la investigación del tema y de posibles aplicaciones en sistemas distribuidos contribuyendo con publicaciones en congresos.

# REFERENCIAS


[YAV00] R. Yavatkar, D. Pendarakis, y R. Guerin. “A Framework for Policy-based Admission Control”. rfc2753. Enero 2000 http://www.faqs.org/rfcs/rfc2753.html.

[COPS01] D. Durham, J. Boyle. “The COPS (Common Open Policy Service) Protocol”. rfc2748, Enero 2000. http://www.faqs.org/rfcs/rfc2748.html.

[PCIM01]B. Moore, E. Ellessono, J. Strassner. “Policy Core Information Model”. rfc3060, Febrero 2001, http://www.faqs.org/rfcs/rfc3060.html.

[IBA08] A. Guerrero Ibáñez, A. Barba Marti, “Arquitectura de gestión de red basada en políticas para un Ambiente Integrado 3G-WLAN”, Junio 2008.

[WSP06] Siddharth Bajaj, Don Box. “Web Services Policy Framework (WS-Policy)”, March 2006.

[ERL07] Thomas Erl. “SOA: Principlies of service design”, 2007.

[AND04] Anne Anderson. “Comparando WSPL y WS-Policy, IEEE Policy 2004 Workshop”, Junio 2004

[ANDR05] A. Anderson, “A comparison of two privacy policy languages: EPAL and XACML“, Sun Microsystems Laboratory Technical Report #TR-2005-147. http://research.sun.com/techrep/2005/abstract-147.html.

[WSPA06] D. Box. "Web Services Policy Attachment (WS-PolicyAttachment)", Marzo 2006.

[WSSP05] G. Della-Libera. "Web Services Security Policy Language (WS-SecurityPolicy)", Julio 2005.

[WSS04] A. Nadalin. "Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)," Marzo 2004.

[WSRM05] Web Services Reliable Messaging OASIS, Julio 2003.

[MTOM05] “SOAP Message Transmission Optimization Mechanism” Recomendación W3C, Enero 2005. http://www.w3.org/TR/soap12-mtom/

[MPS96] Damian Marriott, Morris Sloman. “Management Policy Service for Distributed Systems”. Imperial College, Deparfment of Computing. 1996.

[PDM94] Sloman. “Policy Driven Management for Distributed Systems”, M.S. Journal of Network and Systems Management, 1994.

[TPBM01] A. Westerinen, J. Schnizlein, J. Strassner.”Terminology for Policy-Based Management”, 2001. http://www.faqs.org/rfcs/rfc3198.html

[PCLD04] J. Strassner, B. Moore, R. Moats. “Policy Core Lightweight Directory Access Protocol”, 2004. http://www.faqs.org/rfcs/rfc3703.html

[PCELD05] M. Pana, A. Reyes, A. Barba. “Policy Core Extension Lightweight Directory Access Pr” 2005. http://www.faqs.org/rfcs/rfc4104.html

[PNDR00] Damianou, N., N. Dulay, E. Lupu and M. Sloman (2000b). “Ponder: A Language for Specifying Security and Management Policies for Distributed Systems. The Language Specification - Version 2.3. Research Report DoC 2000/1, Imperial College of Science Technology and Medicine, Department of Computing”.

[PLPI06] Richard John Anthony . “A Policy-Definition Language and Prototype Implementation Library for Policy-based Autonomic Systems”. 2006.

[PLB99] J. Lobo, R. Bathia, S. Naqvi. “A Policy Description Language”. http://citeseer.ist.psu.edu/lobo99policy.html.

[SPL99] C.Ribeiro, A. Zúquete, P. Ferreira, P. Guedes. “SPL: An access control language for security policies with complex constraints”. http://www.inesc-id.pt/pt/indicadores/Ficheiros/1162.pdf.

[OASIS05] eXtensible Access Control Markup Language (XACML) Version 1.05. OASIS Standard. http://www.oasis-open.org/committees/xacml/repository/ ??

[P3P07] Marco Casassa, Renato Iannella. “Platform for Privacy Preferences (P3P) de W3C”. http://www.w3.org/P3P/.

[EIBM03] Paul Ashley, Satoshi Hada. “Enterprise Privacy Authorization Language”. http://www.zurich.ibm.com/security/enterprise-privacy/epal/Specification/index.html.

[3GPP05] “Evolution of policy control and charging (Rel 7)”, Septiembre 2005.

[CHC07] Vivying S. Y. Cheng, Patrick C. K. Hung, and Dickson K.W. Chiu. “Enabling Web Services Policy Negotiation with Privacy preserved using XACML”. 40th Hawaii International Conference on System Sciences.

[PFIETF99] M. Stevens, W. Weiss. “IETF Policy Framework (policy)”. http://tools.ietf.org/html/draft-ietf-policy-framework-00 . Septiembre 1999

[HLJ04] Hung, Li, Jeng. “WS-Negociation: an Overview of Research Issues”, 2004.

[SLNP02] Nguyen, Boukhatem. “COPS-SLS: A Service Level Negotiation Protocol for the Internet”, 2002.

[CHAN01] K. Chan. “COPS Usage for Policy Provisioning (COPS-PR),” RFC 3084, Marzo 2001. http://www.faqs.org/rfcs/rfc3084.html

[STEF04] Wohlstadter, Stefan. “GlueQoS: Middlewareto Sweeten Quality-of-Service Policy Interactions”. IEEE International Conference on Software Engineering (ICSE’04).

[RAJN99] R. Rajan et al., “A Policy Framework for Integrated and Differentiated Services in the Internet,” IEEE Network, vol. 13, no. 5, pp. 36–41.

[WCWF04] Wang, G. Chen, A. Wang, C. Fung, C. Uczekaj, S.; “Integrated Quality of Service (QoS) Management in Service-Oriented Enterprise Architectures” Enterprise Distributed Object Computing Conference, 2004.

[FRI02] R. Friedrich. “Admisión Control and Resource Reservation on the Internet”, 2002.

[PIB03] B. Gage, B. Kosinski . “Session Authorization Policy Element. Policy Information Base”, RFC3520, Abril 2003.

[BCK03] Bass, Len, Paul Clements, and Rick Kazman . “Software Architecture in Practice”. Addision-Wesley. 2003.

[BRJ99] Booch, Grady, James Rumbaugh, and Ivar Jacobson . “Unified Modeling Language User Guide”. Addision-Wesley, 1999.

[ESBP04] Dirk Krafzig, Karl Banke, Dirk Slama. “Enterprise SOA: Service-Oriented Architecture Best Practices”.

[SHP05] Srinivasan and Jem Treadwell. “An Overview of Service-oriented Architecture, Web Services and Grid Computing”. HP Software Global Business Unit.

[TREA05] Latha Srinivasan and Jem Treadwell. “An Overview of Service-oriented Architecture, Web Services and Grid Computing”.

[ESB03] Roy Schulte of Gartner. “Predicts 2003: Enterprise Service Buses Emerge”. 2003.

[BEV03] Nigel Bevan. “Quality in Use: Meeting User Needs for Quality”, 2003

[BCK98] Bass, Len, Paul Clements, y Rick Kazman . “Software Architecture in Practice”. Addision-Wesley. 1998

[SEI05] Software Engineering Institute. Software Architecture Technology Initiative. http://www.sei.cmu.edu/architecture/sat_init.html (2005).

[BRO04] Brownsword, L. “Current Perspectives on Interoperability (CMU/SEI-2004-TR-009, ADA421613)”. Carnegie Mellon University, 2004. http://www.sei.cmu.edu/publications/documents/04.reports/04tr009.html.

[BOE78] Boehm, B. “Characteristics of Software Quality”. New York: Elsevier North-Holland Publishing Company, Inc. 1978.

[ISO91] International Organization for Standardization (ISO), “Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for Their Use,” ISO/IEC 9126, 1996.

[PAP03] MikeP.Papazoglou. “Service Oriented Computing: Concepts, Characteristics and Directions”. International Conference on Web Information Systems Engineering. 2003.

[GSOAP10] http://www.cs.fsu.edu/~engelen/soap.html

[ECL10] www.eclipse.org

[AAX10] http://ws.apache.org/axis/

[APN14] “Apache Neethi”. http://ws.apache.org/neethi/index.html

[SLPS09] “Microsoft Software License Service” http://www.microsoft.com/slps/

[RLM14] “Reprise License Manager Software” http://www.reprisesoftware.com/products/software-license-management.php

[NAP14] http://www.nalpeiron.com/#

[ITAM14] “Vector License Manager Solution” http://www.vector-networks.com/it-asset-and-service-management/ITAM-SAM-products/license-manager/?source=capterra-profile

[SOR14] http://www.soraco.co

# GLOSARIO

PDP: Punto de Decisión de Políticas

LPDP: Punto de Decisión de Políticas Local

PEP: Punto de Ejecución de Políticas

PBN: Policy-Based Networking

PBM: Policy-Based Management

PDL: Lenguaje de Descripción de Políticas

PIP: Punto de Información de Políticas

PNP: Punto de Negociación de Políticas

COPS: Common Open Policy Service

QoS: Calidad de Servicio

RSVP: Protocolo de Reservacion de Recursos

IETE: Internet Engineering Task Force

LDAP: Protocolo Ligero de Acceso a Directorios

XACML: eXtensible Access Control Markup Language

SLA: Acuerdo de Nivel de Servicio

COPS-SLS: COPS usado para negociación de nivel de servicio

COPS-PR: COPS usado para aprovisionamiento de políticas

DQoS: Calidad de Servicio Dinámica

SOA: Arquitectura Orientada a Servicios

ESOA: Arquitectura Orientada a Servicios Extendida

ESB: Enterprise Service Bus

WSDL: Web Service Description Language

UDDI: Universal Description Discovery and Integration

SOAP: Simple Object Access Protocol

RPC: Llamada a Procedimientos Remotos

PES: Servicio de Ejecución de Políticas

PDS: Servicio de Decisión de Políticas

PSD: Servicio de Descubrimiento de Políticas

# ANEXO A: PROTOCOLO COPS

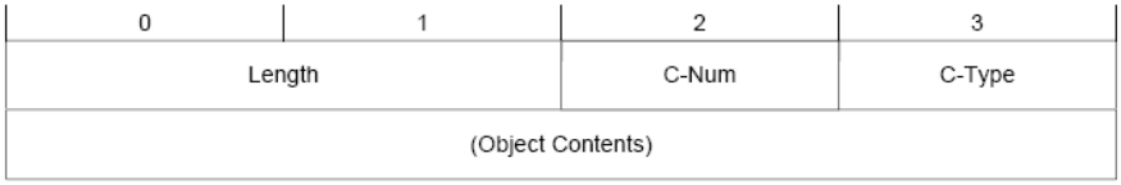
## *A.1: El Protocolo*

Esta sección describe el formato de los mensajes y objetos del protocolo COPS intercambiados entre el PEP y el PDP.

### A.1.1 Objetos

Los objetos son utilizados para encapsular la información que se transfiere entre las entidades PEP y PDP. Cada objeto consiste en un encabezado y el contenido (en donde encapsula la información a transmitir). La estructura del encabezado consta de tres campos Length, C-Num y C-Type; y el contenido puede ser desde un string hasta otro objeto de algún protocolo.

**Common Header**.




Length: Define el tamaño del mensaje, compuesto por dos octetos (16 bits). Identifica la cantidad de octetos que componen el objeto. Tiene en cuenta la cabecera más el contenido.

C – Num: Compuesto por 8 bits. Identifica la clase de información contenida en el objeto.

(1) = Handle
(2) = Context
(3) = In Interfaz
(4) = Out Interfaz
(5) = Reason code
(6) = Decision
(7) = LPDP Decision
(8) = Error
(9) = Client Specific Info
(10) = Keep-Alive Timer
(11) = PEP Identification
(12) = Report Type
(13) = PDP Redirect Address
(14) = Last PDP Address
(15) = Accounting Timer
(16) = Message Integrity

C – Type: Es un subtipo o versión de la información contenida en los objetos. El valor esta definido por 8 bits y depende del C – Num.

Object Contents: Información que lleva cada uno de los objetos.

**OBJETOS**

**Handle (1)**

C–Num: 1, C–Type: 1

El Handle Object encapsula un único valor que identifica un estado instalado; responsable de identificar una conexión entre PEP y PDP. Este identificador es usado por muchos mensajes COPS.

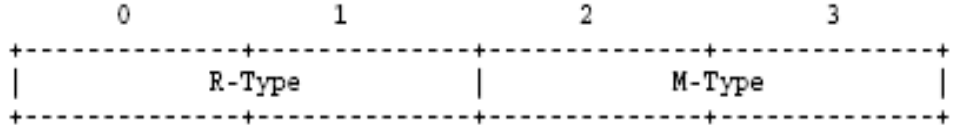
La inicialización del objeto lo maneja el PEP, empleando el mensaje REQ, y puede solicita le borrado del mismo por intermedio del mensaje DRQ. Además, el cliente (PEP) lo utiliza en el mensaje RPT para informar la correcta o no instalación de alguna decisión enviada por el PDP, referida a este Handle. Por último, el PEP puede utilizarlo, en el menaje SSC, para informar que se ha realizado la sincronización de los datos, referidos a este identificador.

**Context (2)**

C–Num: 2, C–Type: 1

Especifica el tipo de evento por el cual se realizo una consulta (request), por ejemplo una solicitud de configuración. Requerido para mensajes de solicitud, control de admisión y asignación de recursos. Facilita al PDP la toma de decisiones en función a lo que solicite el PEP. Este objeto se utiliza en los mensajes REQ y DEC.

Se compone por dos campos:




*R–Type* (Request type flag):

- (0x01) Incoming-Message/Admission Control request: Identifica a mensajes de entrada, relacionados con los protocolos de señalización
- (0x02) Resource-Allocation request: Se utiliza para asignar nuevos recursos al PEP, una vez que realizó la configuración inicial
- (0x04) Outgoing-Message request: Identifica a mensajes de salida, relacionados con los protocolos de señalización
- (0x08) Configuration request: Esta opción es seleccionada por el cliente PEP, cuando desea que el servidor PDP le envíe los parámetros de configuración

*M–Type* (Menssage Type):

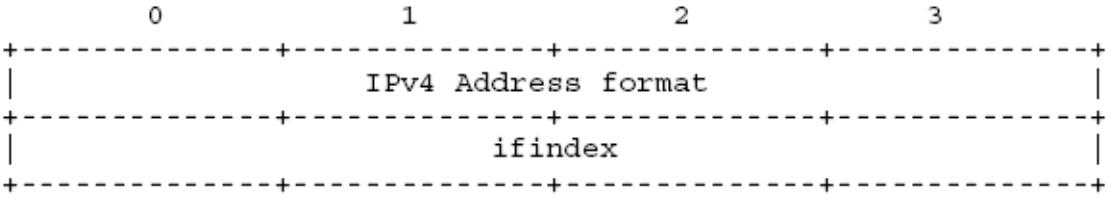
Se describe por 16 bits y especifica el tipo de mensaje

**InInterface (3)**

C–Num: 3, C–Type: 1  IPv4 Address + Interfaz

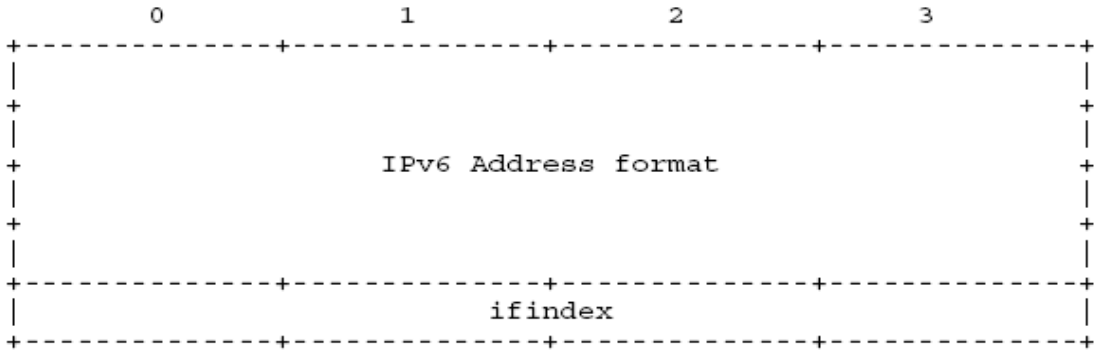
Es usado para identificar el interfaz entrante en el cual una petición particular se aplica y la dirección de donde el mensaje recibido provino.

Este objeto esta constituido por dos subtipos, IPv4 (utilizado cuando se trabaja con direcciones IP versión 4) e IPv6 (utilizado cuando se trabaja con direcciones IP versión 6).




*IPv4 Address*: Dirección IP
*Ifindex:* Puede ser usado para distinguir entre subinterfaces e interfaces sin numerar

C–Num: 3, C-Type: 2  IPv6 Address + Interfaz.




En este caso soporta  IPv6 address.

**Out Interfaz  (4)**

C–Num: 4, C–Type: 1  IPv4 Address + Interfaz

Es usado para identificar el interfaz saliente al cual una petición específica se aplica y la dirección para donde el mensaje expedido debe ser enviado.

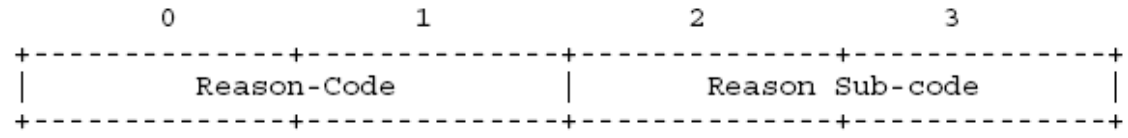
La conformación de los campos es idéntica al objeto In–Interfaz

C–Num: 4, C–Type: 2  IPv6 Address + Interfaz

**Reason code (5)**

C – Num: 5, C – Type: 1

Especifica la razón por la cual el estado de la petición fue borrada. Este objeto aparece en el mensaje DRQ (Delete Request State)




*Reason–Code*:

1. Unspecified : Ocurre cuando las razones de la petición del borrado del Handle no se encuentran en las opciones establecidas
2. Management: La decisión de borrar del estado fue tomada por una aplicación en el cliente
3. Preempted (Another request state takes precedence): Cuando el estado actual va a ser reemplazado por otro estado con más importancia
4. Tear (Used to communicate a signaled state removal): Permite identificar el retiro de un estado por completo
5. Timeout (Local state has timed-out): l tiempo de uso del estado seleccionado llegó a su fin
6. Route Change (Change invalidates request state): Una decisión tomada por el servidor produce que otro estado quede obsoleto
7. Insufficient Resources (No local resource available): Los recursos que posee son escasos y no es posible seguir asignando estados al cliente
8. PDP's Directive (PDP decision caused the delete): El borrado del estado fue solicitado por el servidor PDP
9. Unsupported decision (PDP decision not supported): La decisión que el servidor PDP envió al cliente PEP no es soportada
10. Synchronize Handle Unknown: Cuando el servidor realizo licita una sincronización, puede hacerlo especificando un Handle determinado, si este no es correcto, el cliente seleccionará esta opción
11. Transient Handle (stateless event): El cliente desea borrarlo porque fue un estado transitorio
12. Malformed Decision (could not recover): El cliente selecciona esta opción, cuando el servidor envía un mensaje DEC mal formado
13. Unknown COPS Object from PDP: Ocurre cuando el objeto que se adjunto al mensaje enviado por el servidor, esta mal formado, y el cliente no lo reconoce como un objeto COPS

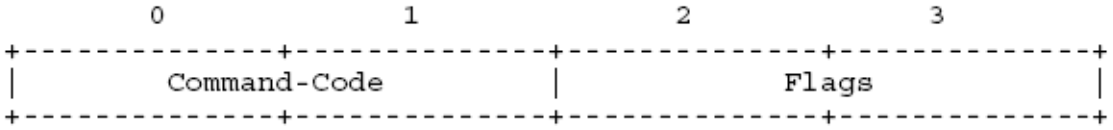
*Reason Sub – Code*: Es especificado por el cliente

**Decision (6)**

Es la decisión tomada por el PDP. Devuelve una respuesta a la petición dependiendo el tipo de cliente.

Este subtipo de objeto es obligatorio en todos los mensajes DEC que se envíen, permite identificar la operación que se debe realizar con la información contenida en los distintos objetos que conformen el mensaje.

C–Num: 6 C–Type: 1 Decision Flag (Mandatory)




*Command - Code*:

0 = NULL Decision (No configuration data available)
1 = Install (Admit request/Install configuration)
2 = Remove (Remove request/Remove configuration)

*Flags*:

1 = Trigger Error (Trigger error message if set)
(Es utilizado para clientes que son capaces de entender mensajes de señalización)

C – Num: 6 C-Type: 2 Stateless Data

Lleva información adicional sin estado que puede ser aplicado localmente al PEP. En el objeto Decisión este subtipo es opcional.

C – Num: 6 C – Type: 3 Replacement Data

Reemplaza datos existentes de un mensaje determinado. En el objeto Decision es opcional.

C – Num: 6 C – Type: 4 Client Specific Decision Data

Las decisiones específicas tomadas para determinados clientes se utiliza con este subtipo de decisión. Su utilización es opcional.

C – Num: 6 C – Type: 5 Named Decision Data

La información que transporta es la configuración solicitada por un REQ.

**LPDP Decision (7)**

C – Num: 7 C – Type: (identificado de la misma manera que el objeto Decision)

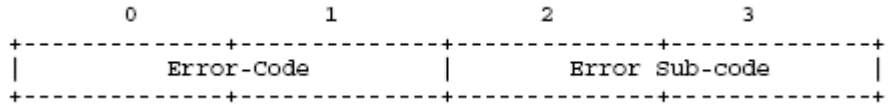
Son las decisiones tomadas por el servidor de políticas local (LPDP). Puede aparecer en los REQ. Los subtipos son definidos de la misma forma que el objeto anterior.

Decisión tomada por el punto de decisión local de política del PEP (LPDP). Puede aparecer en peticiones. Estos objetos corresponden a y se ajustan a formato iguales que los objetos específicos de la decisión del cliente definieron arriba

**Error (8)**

C – Num: 8 C – Type: 1

Utilizado para identificar los errores en el protocolo.




*Error – Code*:

1. Bad handle
2. Invalid handle reference
3. Bad message format (Malformed Message)
4. Unable to process (server gives up on query)
5. Mandatory client-specific info missing
6. Unsupported client-type
7. Mandatory COPS object missing
8. Client Failure
9. Communication Failure
10. Unspecified
11. Shutting down
12. Redirect to Preferred Server
13. Unknown COPS Object: Sub-code (octet 2) contains unknown object's C-Num and (octet 3) contains unknown object's C-Type.
14. Authentication Failure
15. Authentication Required

*Error Sub – Code*: Especificado en forma particular al tipo de cliente

**Client Specific Info (9)**

Es utilizado en los REQ, OPN y RPT. Contiene información específica de los clientes.

C – Num: 9 C – Type: 1 Signaled ClientSI

Especifica información sobre aquellos clientes que manejan protocolos de señalización.

C – Num: 9 C – Type: 2 Named ClientSI

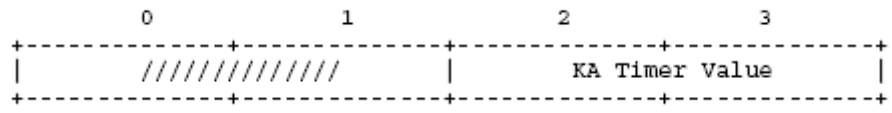
Contiene información necesaria para configurar al cliente PEP, solicitado al servidor PDP.

**Keep-Alive Timer (10)**

C – Num: 10 C – Type: 1 Keep-alive timer value

Es usado para especificar el tiempo máximo del intervalo en que un mensaje COPS debe ser enviado o recibido. Los tiempos se codifican utilizando dos octetos y la unidad

utilizada en el segundo y el rango de valores que se pueden asignar va desde 1 hasta 65535.




**PEP Identification (11)**

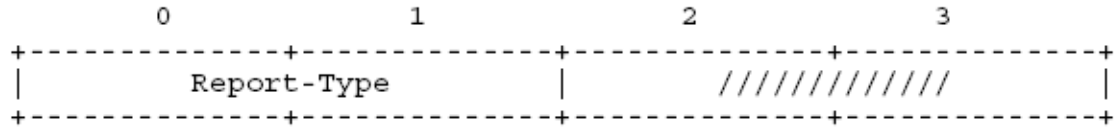
C – Num: 11 C – Type: 1

Es utilizado para identificar al cliente PEP en el servidor PDP. Se emplea en los mensajes OPN. Es de longitud variable, pero con un máximo de 32 bit (4 octetos).

**Report Type (12)**

C – Num: 12 C – Type: 1

Es utilizado para reportar el estado de una decisión enviada por el servidor PDP. El mensaje que utiliza este objeto es el RPT.



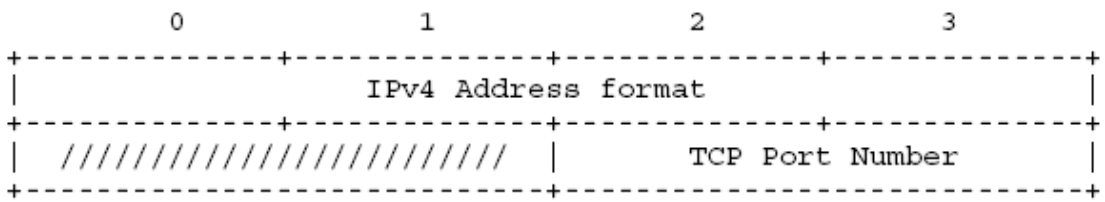

*Report – Type:*

1 = Success: La decisión fue aceptada por el PEP
2 = Failure: La decisión no fue completada por el PEP
3 = Accounting: Actualización del contador de estado instalado

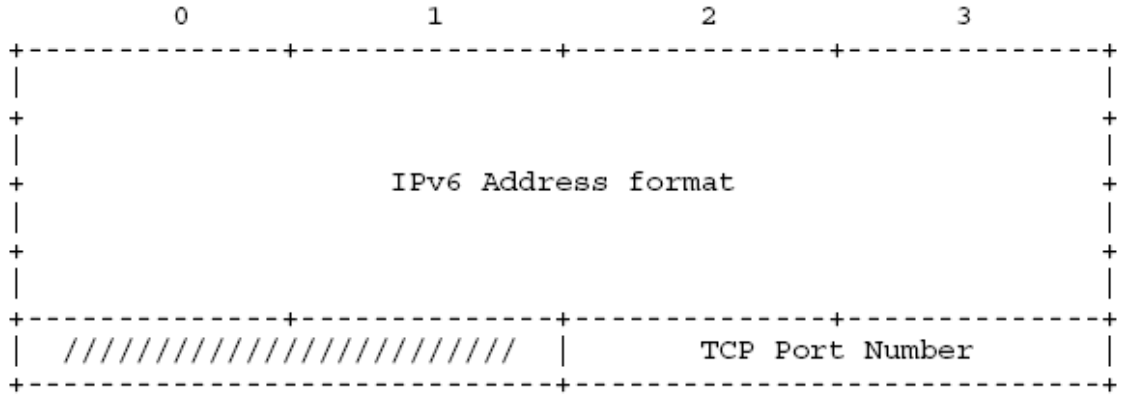
**PDP Redirect Address (13)**

Cuando el servidor PDP cierra una sesión con el cliente PEP, a causa de que no trabaja con un tipo de cliente determina, el servidor PDP le puede informar al cliente PEP que servidor puede utilizar. Es decir, utilizando este objeto, el servidor redirecciona al cliente a otro servidor remoto. El servidor especifica en el objeto la dirección IP y el puerto TCP a donde el cliente debe buscar.

C – Num: 13 C – Type: 1 IPv4 Address + TCP port




C – Num: 13 C – Type: 2 IPv6 Address + TCP port




**Last PDP Address (14)**

Al contrario del objeto anterior, este objeto es utilizado por el cliente PEP para informa al servidor PDP con el cual se está comunicando en ese momento, cual fue el último servidor PDP con el que estuvo conectado. Puede suceder que el cliente se reinicie, lo que implica que no posea ninguna información del último servidor con el cual estuvo conecta, con lo cual el objeto Last PDP Address no se incluye en el mensaje OPN (único mensaje que lo utiliza).

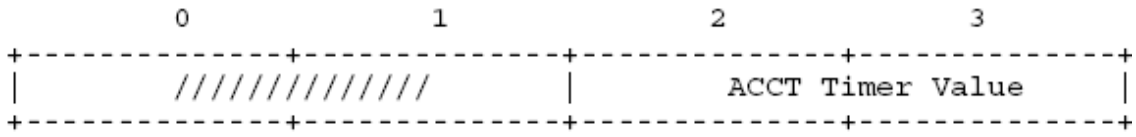
C – Num: 14 C – Type: 1 IPv4 Address (El formato es igual al objeto anterior)

C – Num: 14 C – Type: 2 IPv6 Address (El formato es igual al objeto anterior)

**Accounting Timer (15)**

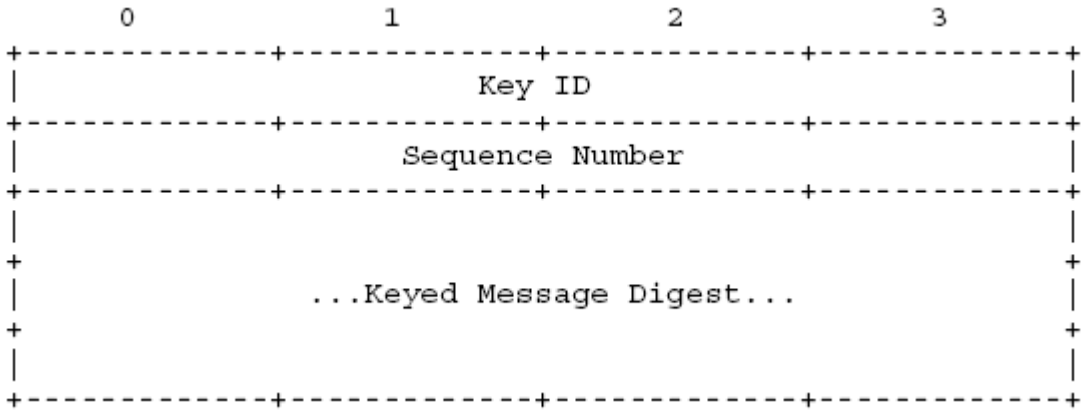
C – Num: 15 C – Type: 1

Determina un intervalo de tiempo que debe esperar el PEP antes de solicitar un REQ. El tiempo se codifica utilizando dos octetos y la unidad es el segundo. El intervalo de valores va desde 1 hasta 65535




**Integrity (16)**

Incluye una secuencia de números y un resumen del mensaje, usado para la autentificación y validación del mensaje. Este objeto se ubica al final de la lista de objetos utilizados en un mensaje.




Key ID: Es un identificador de Key, es decir, el servidor PDP y el cliente PEP utilizan una Key en el algoritmo criptográfico para generar un resumen del mensaje enviado. Este algoritmo trabaja con una Key (llave) privada que no se puede enviar en el mensaje, para identificar a esta llave privada se emplea el Key Id. Este identificador se lo representa con 32 bits.

Sequence Number: Es un entero sin signo de 32 bits, que se incrementa por cada mensaje se envían entre las entidades PEP y PDP. Cada entidad negocia el valor de inicio del Sequence Number por intermedio de los mensajes OPN y CAT y es incrementado cada vez que un nuevo mensaje es enviado por una conexión TCP en una misma dirección. Cuando el conteo llega hasta 0xFFFFFFFF se reinicia el valor a cero.

Keyed Message Digest: Es de longitud variable y se calcula partiendo desde el encabezado del mensaje hasta el objeto Integrity, incluyendo el encabezado del objeto, el Key ID y el Sequence Number. No incluye el Keyed Message Digest. Se debe considerar como un string. En el caso particular del protocolo COPS, utiliza HMAC-MD5-96[1] para generar este campo, esto determina que la longitud del campo sea de 96 bits.

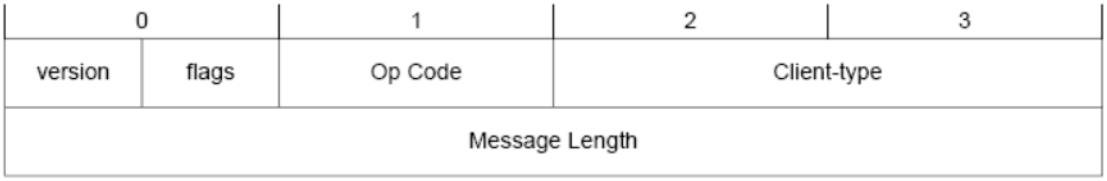
## A.1.2 Mensajes

Los mensajes se caracterizan por ser utilizados para realizar la comunicación entre las entidades PEP y PDP. Estos se integran por objetos que permiten transportar, de una unidad a otra, la información suficiente para tomar una decisión en función de las políticas previamente establecidas.

Además, al igual que los objetos, los mensajes se constituyen de un encabezado, que nos entrega información referida al tipo de mensaje que se ha enviado, que tamaño posee y cual es el cliente con el que se está trabajando. Además, nos dice si el mensaje es una respuesta a una solicitud o ha sido enviado por cuenta de cualquiera de las entidades que intervienen en la comunicación.

**Common Header:**

[1] *HMAC*: Keyed-Hashing for Message Authentication -- *MD5*: Función criptográfica -- *96*: Truncamiento de los bits de salida de la función de encriptación




Versión: Compuesto de 4 bits en donde se detalla la versión del protocolo utilizado. Comúnmente la versión es 1.

Flags: Es utilizado para saber si el mensaje corresponde a la respuesta de una solicitud, es decir, toma el valor 0x01 cuando se envía un mensaje Decisión siempre que haya habido en primera instancia un mensaje Request proveniente del cliente. En caso contrario se mantiene con valor 0x00. El campo se integra por 4 bits

Op Code: Compuesto por 8 bits, identifica la operación a ejecutar.

1 = Request (REQ)
2 = Decision (DEC)
3 = Report State (RPT)
4 = Delete Request State (DRQ)
5 = Synchronize State Req (SSQ)
6 = Client-Open (OPN)
7 = Client-Accept (CAT)
8 = Client-Close (CC)
9 = Keep-Alive (KA)
10 = Synchronize Complete (SSC)

Client–type: Se compone de 16 bits. Identifica el tipo de cliente que utiliza el protocolo.

Message Length: Compuesto por 32 bits. Define el tamaño del mensaje en cantidad de octetos tomando en cuenta la cabecera y los restantes mensajes que contenga.

**Request (REQ) PEP -> PDP**

El PEP envía un Request Message al PDP a fin de solicitar políticas de calidad.

```
<Request> = <Common Header>
            <Client Handle>
            <Context>
            [<IN-Int>]
            [<OUT-Int>]
            [<ClientSI(s)>]
            [<LPDPDecision(s)>]
            [<Integrity>]

<ClientSI(s)>:= <ClientSI> |
                <ClientSI(s)> <ClientSI>

<LPDPDecision(s)>::= <LPDPDecision> |
                 <LPDPDecision(s)> <LPDPDecision>

<LPDPDecision> = [<Context>]
                 <LPDPDecision: Flags>
```

```
                [<LPDPDecision: Stateless Data>]
                [<LPDPDecision: Replacement Data>]
                [<LPDPDecision: ClientSI Data>]
                [<LPDPDecision: Named Data>]
```

El Context object es usado para determinar el contexto dentro del cual los demás objetos serán interpretados.

Un mal formado Request Message debería resultar en el PDP con un Decision Message con el apropiado Error Code.

**Decision (DEC) PDP -> PEP**

Lo utiliza el servidor PDP en respuesta a los mensajes REQ, es decir, cuando el cliente PEP realiza una solicitud el servidor PDP devolverá el mensaje DEC con el campo Flag del encabezado del mensaje en 1 y los objetos que sean necesarios para responder a la solicitud del cliente. Puede ocurrir que el servidor PDP envíe este mensaje sin una solicitud previa, en este caso el campo Flag del encabezado del mensaje quedará en 0. El contenido del objeto Handle va a depender del estado que haya sido inicializado por el cliente. El mensaje se integra por los siguientes objetos.

```
<Decision> = <Common Header>
             <Client Handle>
             <Decision(s)> | <Error>
             [<Integrity>]

<Decision(s)>::= <Decision> |
                  <Decision(s)> <Decision>

<Decision>::= <Context>
               <Decision: Flags>
               [<Decision: Stateless Data>]
               [<Decision: Replacement Data>]
               [<Decision: ClientSI Data>]
               [<Decision: Named Data>]
```

La representación anterior nos permite observer que el mensaje DEC es utilizado tanto para responder a una solicitud en forma positiva (objeto Decision) o informar que la solicitud ha sido rechazada por algún error (objeto Error). En caso de una decisión correcta, ésta puede esta compuesta por varios objetos Decision, pero en cualquiera de los casos en donde se incorpore este objeto, es obligatorio que se incorpore el objeto Context y el subtipo <Decision: Flags>, antes de cualquiera de los otros subtipos

**Report State (RPT) PEP -> PDP**

Este mensaje es usado pode el PEP para comunicar al PDP el éxito o falla en la decisión enviada por el PDP, o para reportar un cambio de estado de la decisión que se instalo en él. También se lo utiliza para encapsular información particular de un tipo de cliente, cuando el servidor lo solicita, esto se lo realiza incorporando al mensaje el objeto ClientSI

```
<Report State>::== <Common Header>
                    <Client Handle>
                    <Report-Type>
                    [<ClientSI>]
```

```
                    [<Integrity>]
```

**Delete Request State (DRQ) PEP -> PDP**

Solamente lo utiliza el cliente PEP para solicitarle al servidor PDP que elimine un estado instalado de un determinado Handle. Para informar cual es la razón de la solicitud se incorpora el objeto.

```
<Delete Request>::= <Common Header>
                     <Client Handle>
                     <Reason>
                     [<Integrity>]
```

**Synchronize State Request (SSQ) PDP -> PEP**

Es usado por el PDP para solicitar al PEP que reenvíe su información para una sincronización con el PDP, ya sea relacionado a un determinado Handle o que envíe todos los estados que actualmente posee.

```
<Synchronize State>::= <Common Header>
                        [<Client Handle>]
                        [<Integrity>]
```

**Client-Open (OPN) PEP -> PDP**

Es enviado por el PEP para abrir una conexión con el PDP. Mediante este mensaje el PEP indica al PDP el tipo de cliente que soporta, el último PDP al cual se conecto y/o alguna información especifica del tipo de cliente. Multiples Client-Open pueden ser enviados durante una comunicación en caso de que el PEP soporte varios Client-Types. Si el tipo de cliente (campo Client Type del encabezado del mensaje) es 0, se esta solicitando la inicialización de una sesión con autentificación de mensaje, es decir, se incluye el objeto Integrity.

```
<Client-Open>::= <Common Header>
                  <PEPID>
                  [<ClientSI>]
                  [<LastPDPAddr>]
                  [<Integrity>]
```

**Client-Accept (CAT) PDP -> PEP**

Es una respuesta del OPN comunicando que el PDP acepto la conexión del PEP y por lo tanto puede empezar a enviar mensajes de Request para solicitar políticas. Este mensaje retornara al cliente un Keep-Alive-Timer indicando el máximo intervalo de tiempo entre Keep-Alive Messages

```
<Client-Accept>::= <Common Header>
                    <KA Timer>
                    [<ACCT Timer>]
                    [<Integrity>]
```

**Client-Close (CC) PEP -> PDP, PDP -> PEP**

Cierra la conexión entre el PDP o el PEP. El mensaje puede ser enviado por cualquier entidad. Su función es informar a cualquier entidad del cierre de la conexión TCP a causa de un error en el mensaje o por algún evento en el cliente o en el servidor. El servidor PDP emplea este mensaje para cancelar el inicio de una sesión COPS, a causa de no trabajar con el tipo de cliente que solicita la sesión.

```
<Client-Close>::= <Common Header>
                  <Error>
                  [<PDPRedirAddr>]
                  [<Integrity>]
```

**Keep-Alive (KA) PEP -> PDP, PDP -> PEP**

Es un mensaje enviado entre entidades como forma de señalar su estado activo. En el inicio de la sesión COPS se establece el intervalo máximo de tiempo que puede estar la conexión TCP sin usar. El cliente PEP es el encargado de enviar el mensaje KA antes de que se cumpla este tiempo, el servidor PDP como respuesta vuelve a enviar el mensaje KA confirmando que la conexión TCP se encuentra activa

```
<Keep-Alive>::= <Common Header>
                [<Integrity>]
```

**Synchronize State Complete (SSC) PEP -> PDP**

Es una respuesta al SSQ, indicando que la sincronización fue finalizada.

```
<Synchronize State Complete>::= <Common Header>
                                [<Client Handle>]
                                [<Integrity>]
```

# ANEXO B: Requerimientos de Calidad

## *B.1: Requerimientos de Calidad*

Requerimientos de calidad de software tales como performance, seguridad, disponibilidad tienen gran influencia en la arquitectura de software de un sistema. Los arquitectos necesitan comprender su diseño en términos de atributos de calidad. Por ejemplo, ellos necesitan comprender cuando están diseñando un sistema con requerimientos de sistemas tiempo real, como debe responder un sistema a fallas, cual es el nivel de seguridad que debe proveer.

Los requerimientos de atributos de calidad conducen el diseño de la arquitectura del software [SEI05]. Elegir y diseñar una arquitectura para tales sistemas – que satisfaga los requerimientos de atributos de calidad- es vital.

A fin de ver cuales son esas implicaciones cuando diseñamos e implementamos SOA se presentan un conjunto de atributos de calidad que pueden ser requeridos por la aplicación que use el servicio.

1) **Interoperatividad:**

   Se refiere a la habilidad de un conjunto de entidades de comunicación para compartir información específica y operar esta de acuerdo a un grado de semántica operacional [BRO04]. Incrementar la interoperabilidad es uno de los mayores beneficios que provee SOA, más cuando consideramos Web Services [MCGO03].

   Muchos sistemas distribuidos han sido desarrollados usando diferentes lenguajes y plataformas tales como Common Object Request Broker Architecture (CORBA), Remote Method Invocation (RMI), Distributed Component Object Model (DCOM), Remote Procedure Call (RPC), y socket para comunicación. Sin embargo hasta la llegada de Web Services no había un protocolo de comunicación estándar o formato de datos de comunicación que puedan ser usados efectivamente por sistemas usando diferentes tecnologías para interoperar en un amplio rango.

2) **Fiabilidad**:

   Se refiere a la habilidad de un conjunto seguir operando sobre el tiempo [CLE02]. Varios aspectos de fiabilidad son importantes en SOA, en particular la fiabilidad de mensajes intercambiados entre la aplicación y los servicios y la fiabilidad de los servicios en si mismo. Aplicaciones desarrolladas por diferentes organizaciones pueden tener diferentes requerimientos de fiabilidad para un mismo conjunto de servicios.

   La fiabilidad de los mensajes se refiere a la posibilidad de garantizar intercambio confiable de mensajes entre la aplicación y los servicios. El uso de diferentes capas intermedias desde diferentes proveedores puede complicar el intercambio de mensajes entre aplicaciones y servicios.

Hay dos especificaciones -WS-Reliability (OASIS Consortium) y WS-ReliableMessaging (desarrollado por IBM, BEA, Microsoft, y TIBCO Software)- teniendo en cuenta estos aspectos y tratan de definir protocolos que posibiliten a los servicios asegurar la fiabilidad en el intercambio de mensajes.

La fiabilidad de los servicios se refiere a que el servicio opere correctamente y no falle o reporte fallas al usuario del servicio. La fiabilidad del servicio también se refiere a que el servicio sea obtenido desde un proveedor fiable tal que el nivel de verdad en la precisión del servicio y la fiabilidad pueda ser establecido

3) **Disponibilidad**:

Se refiere al grado en que un servicio o componente es operacional y accesible cuando se requiere usar. Desde la perspectiva del proveedor del servicio y el usuario la disponibilidad de un servicio es de preocupación para el éxito del servicio. Desde el punto de vista del usuario del servicio, si el sistema depende de un conjunto de servicios y uno de estos no esta disponible en un momento dado podría tener consecuencias en el éxito del sistema.

Los proveedores del servicio acuerdan ofrecer a los usuarios del servicio un conjunto de servicios e incluir cada uno de estos en un SLA. Un SLA define un contrato para la provisión del servicio con detalles que garanticen la disponibilidad del servicio y incluyen la penalidad en caso de que el servicio no este disponible.

4) **Seguridad:**

Es una de las mayores preocupaciones en SOA y Web Services, se puede encontrar diferentes aspectos en cuanto a la seguridad:

- Confidencialidad: el acceso a la información o servicio esta garantizado solo para usuarios autorizados.
- Autenticidad: se puede asegurar que el autor indicado es el único responsable de la información.
- Integridad: la información no es corrupta.

La seguridad es uno de los principales aspectos dentro de SOA y Web Services. Los Web Services están usando algunos aspectos de seguridad en el nivel de infraestructura de la red, por ejemplo, pueden ser configurados para usar SSL y encriptación de la información.

5) **Performance:**

Puede tener diferentes significados en diferentes contextos, en general se relaciona al tiempo de respuesta, cuantos pedidos puede procesar por unidad de tiempo, es decir, procesar un pedido en un determinado tiempo aceptable.

Performance es un importante atributo de calidad que usualmente es afectado negativamente en SOA, debido a su inclusión en sistemas distribuidos, el uso de xml en Web Services, las capas intermedias de transporte; es por ello que los diseñadores

deben ser cuidadosos en las tecnologías que usan para evitar cuestiones que afecten la performace.

6) **Escalabilidad:**

Escalabilidad es la posibilidad de una arquitectura orientada a servicios SOA funcione correctamente (sin degradar otros atributos de calidad) cuando el sistema cambia en tamaño o en volumen en orden de satisfacer las necesidades del usuario [W3C04]. Uno de los mayores aspectos de la escalabilidad es la posibilidad del sitio donde los servicios están localizados de acomodar un incremento en el número de usuarios sin degradar la performance.

7) **Usabilidad:**

Conjunto de características que influyen en el esfuerzo requerido para el uso y la evaluación individual de cada uso por parte de un conjunto de usuarios dados. Se divide en las sub-caracteríticas comprensión, facilidad de aprendizaje, operabilidad, atractivo.