Proyecto de grado

Generación automática de Arquitecturas Orientadas a Servicios (SOAs) con SoaML y especificación de QoS

Informe Final

Federico Bertolini, Sofía Pérez, María Noel Quiñones

Tutor
Andrea Delgado

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay
Diciembre 2014

Resumen

Tradicionalmente el modelado en el proceso de desarrollo de software es realizado principalmente como documentación del sistema, la cual difícilmente es actualizada luego de implementado y liberado el mismo. El problema de utilizar los modelos como mera documentación es que fácilmente divergen de la realidad y se vuelven obsoletos. Para poder aprovechar realmente el potencial de los modelos, el paradigma de Ingeniería Dirigida por Modelos (Model Driven Engineering, MDE) se basa en exprimir al máximo los beneficios de la automatización, incluyendo la verificación automática de modelos y generación automática de programas idealmente completos. El Desarrollo Dirigido por Modelos (Model Driven Development, MDD) forma parte de este paradigma y basa el desarrollo de software en modelos, utilizando como artefactos de primer orden metamodelos, modelos y lenguajes que permiten transformaciones entre éstos. Estas transformaciones convierten sucesivamente un modelo en otro modelo del mismo sistema, refinando el nivel de abstracción hasta llegar al código asociado.

La lógica de los Procesos de Negocio (PNs) se puede dividir en unidades lógicas más pequeñas llamadas "servicios" que resuelven una parte específica del problema. La Computación Orientada a Servicios (Service Oriented Computing, SOC) es un paradigma que utiliza estos servicios para soportar el desarrollo ágil de aplicaciones interoperables y masivamente distribuidas a bajo costo. Las aplicaciones desarrolladas utilizando una arquitectura orientada a servicios (Service Oriented Architecture, SOA) cuentan con una gran flexibilidad a cambios en el negocio y cambios tecnológicos.

El modelado de servicios se puede realizar en base al estándar SoaML (SOA Modeling Language) del OMG (Object Management Group). SoaML es un perfil y metamodelo UML que permite modelar servicios con elementos UML y estereotipos específicamente definidos. SoaML permite modelar aspectos funcionales relacionados con la red de servicios definidos para solucionar un determinado problema, sin incluir aspectos no funcionales. Para el modelado de características de calidad de los servicios es posible utilizar el estándar QoS (Quality of Service) del OMG, que provee elementos específicos para modelar elementos como seguridad, performance y disponibilidad, entre otros.

Desde el año 2010 el grupo COAL del Instituto de Computación de la Facultad de Ingeniería ha llevado adelante distintos proyectos de grado e investigación en las temáticas de procesos de negocio, servicios y desarrollo dirigido por modelos, incluyendo el desarrollo de *plugins* de Eclipse para modelado con SoaML basado en Papyrus (UML2), y generación de código asociado JEE y WS desde modelos SoaML. En este proyecto se utilizaron como base estos *plugins*, proveyendo una solución integral que incorpora además soporte para el modelado de QoS y su integración a los modelos SoaML y generación de código para las características de calidad modeladas.

El prototipo realizado incluye la actualización de los *plugins* de modelado SoaML y de generación de código a la versión Luna de Eclipse, así como el desarrollo de dos nuevos plugins: uno para la especificación de modelos de QoS que pueden aplicarse al diseño de cualquier sistema de software (diagramas de UML) y otro para el modelado de QoS con SoaML que incluye generación de código específico para las características no funcionales modeladas.

Palabras claves: SOA, Servicios, Modelado, SoaML, Plugins, Eclipse, Papyrus, QoS, Web Services

Contenido

1. Introducción

Este proyecto de grado se enmarca en el trabajo de investigación del grupo COAL, en la línea de Procesos de Negocio, servicios y desarrollo dirigido por modelos, así como en el trabajo con SOA y tecnologías asociadas del grupo LINS del InCo. En particular, continúa la línea de trabajo en modelado e implementación de procesos de negocio con servicios, especificados mediante el estándar SoaML del OMG y agregando la especificación de aspectos no funcionales mediante el estándar QoS también del OMG. De esta forma, desde modelos de procesos de negocio es posible trazar su implementación con servicios especificando tanto aspectos funcionales como no funcionales de los mismos, y generando el código asociado en cada caso.

1.1.Motivación

Cuando se modelan aplicaciones y servicios SOA, se utiliza frecuentemente el término calidad de servicio (Quality of Service, QoS) para referirse a la colección de restricciones y requerimientos de calidad para un servicio. Es importante que las QoS puedan ser modeladas de tal forma que dichas restricciones y requerimientos de calidad puedan ser reconocidas y entendidas por los grupos de interés. Con respecto a SOA, generalmente incluye seguridad, rendimiento y disponibilidad. Un lenguaje de modelado SOA debería ser capaz de modelar aspectos de calidad, para que las condiciones, restricciones y requerimientos en estos aspectos puedan ser especificados en los modelos [1].

Para lograr proveer un nivel aceptable de calidad, QoS debe ser integrado de forma temprana en el proceso de desarrollo de un sistema, con el objetivo de considerar los aspectos no funcionales y las restricciones de los usuarios del sistema [2]. Sin embargo, en los proyectos previos no se incluyeron aspectos no funcionales del sistema, que hacen a la completa especificación de los servicios requeridos para implementar procesos de negocio. Por lo tanto, es importante proveer una solución integral que comprenda los plugins existentes e incorpore el modelado de QoS a los modelos SoaML así como también la implementación de distintas características de calidad asociadas a los web services.

1.2.Objetivos

En esta sección se presentan los objetivos planteados para la realización del trabajo en el marco del proyecto de grado.

El objetivo general (OG) de este proyecto consiste en proveer una solución integral para el modelado de servicios con los estándares SoaML y QoS del OMG, explorando la incorporación del modelado de QoS a los modelos SoaML, y agregando generación de código para dichas características de calidad. Como base del trabajo se deben utilizar los plugins de Eclipse realizados para modelado e implementación de servicios con SoaML en proyectos anteriores.

Para conseguir el objetivo general se plantearon los siguientes objetivos específicos:

- (OE1) Estudiar el estado del arte en orientación a servicios y desarrollo dirigido por modelos, en particular en lo que refiere a modelado y características asociadas, y estándares existentes. Estudiar la teoría de metamodelado, perfiles UML, estándar XMI para intercambio de modelos, estándar SoaML para modelado de servicios, y QoS para especificación de requisitos no funcionales, MDA para desarrollo dirigido por modelo y transformaciones asociadas. Por otro

lado, estudiar el entorno Eclipse y el desarrollo de plugins, la plataforma Java EE y tecnología de Web Services, y plugins existentes para modelado SoaML y generación del código asociado.

- (OE2) Migrar las versiones existentes de los plugins para el modelado SoaML y generación de código asociado, para que funcionen con las últimas versiones de Eclipse, Papyrus y de las distintas APIs y frameworks utilizados, disponibles al momento de comenzar el proyecto.
- (OE3) Implementar el perfil QoS como plugin de Eclipse para soportar la especificación de modelos de calidad de servicio. Investigar y definir cómo modelar las características de calidad en un modelo SoaML con la especificación de QoS del OMG.
- (OE4) Evaluar las distintas alternativas para la implementación de características de calidad en aplicaciones Java. Implementar el agregado de características de calidad en la generación de código a partir de un modelo SoaML con QoS.
- (OE5) Validar el funcionamiento del producto desarrollado mediante un caso de estudio que permita probar las características de la solución elaborada y evaluar los resultados obtenidos.

1.3.Metas alcanzadas y aportes del proyecto

Con respecto al principal objetivo planteado (OG), se cumplió con las expectativas definidas en la especificación del proyecto, obteniendo como resultado un plugin de Eclipse denominado SoaML Toolkit que combina los plugins implementados en los proyectos de grado anteriores e incorpora el modelado y la generación de código de características de calidad de servicios, utilizando los estándares SoaML y QoS.

En cuanto al objetivo específico OE1, se estudiaron los elementos mencionados y a partir de dicho estudio se elaboró el documento de estado del arte que se presenta en este informe que sirvió de base para la realización del proyecto, y cuyo documento completo se adjunta como anexo.

Con respecto al objetivo OE2, se logró realizar la migración de los plugins existentes. El plugin SoaML fue desarrollado para Eclipse Helios, mientras que el plugin SoaML2Code no tenía ninguna restricción sobre la versión de Eclipse a utilizar. Por otro lado, el plugin SoaML fue desarrollado para la versión 0.7 de Papyrus. Nuestro objetivo era utilizar las últimas versiones estables de las herramientas y tecnologías necesarias, por lo que optamos primero que nada por utilizar la versión recientemente liberada de Papyrus, la 1.0.0. Ésta fue liberada junto con la última versión de Eclipse, versión 4.4 (nombrada Luna), por lo que el uso de la última versión de Papyrus estaba atado al uso de la última versión de Eclipse. Para el generador de código también se optó por utilizar las últimas versiones de los servidores JBoss y Tomcat. En el caso de JBoss se migró de la versión 5.0.1 GA a la 7.1.1 Final y en el caso de Tomcat se migró de la versión 7.0 a la 8.0.12.

Para el objetivo OE3 se estudió exhaustivamente la especificación de QoS así como también diversos textos que planteaban distintas alternativas para el modelado de características de calidad. Se consiguió definir un perfil en Papyrus que cumple con la especificación del OMG y se definió la forma en que los elementos de este perfil se relacionan con los elementos del estándar SoaML para modelado de servicios.

Se analizaron distintas opciones para incorporar características de calidad en el desarrollo de web services, que luego fueron implementadas en el generador de código, cumpliendo así con el objetivo OE4.

Con la herramienta construida se pudo modelar completamente el caso de estudio seleccionado y generar el código con el agregado de características de calidad a partir del modelo SoaML+QoS. De esta forma se cumplió con el último objetivo del proyecto.

1.4.Desarrollo del proyecto

En esta sección se presenta la coordinación, planificación y ejecución del proyecto, enunciando las distintas actividades que fueron necesarias para el desarrollo del mismo y el tiempo insumido por cada una.

1.4.1. Gestión de Configuración

Para almacenar el código fuente se utilizó un repositorio[1] facilitado por la facultad y TortoiseSVN[2] fue la aplicación cliente elegida. El repositorio está estructurado como se muestra en la Figura 1.

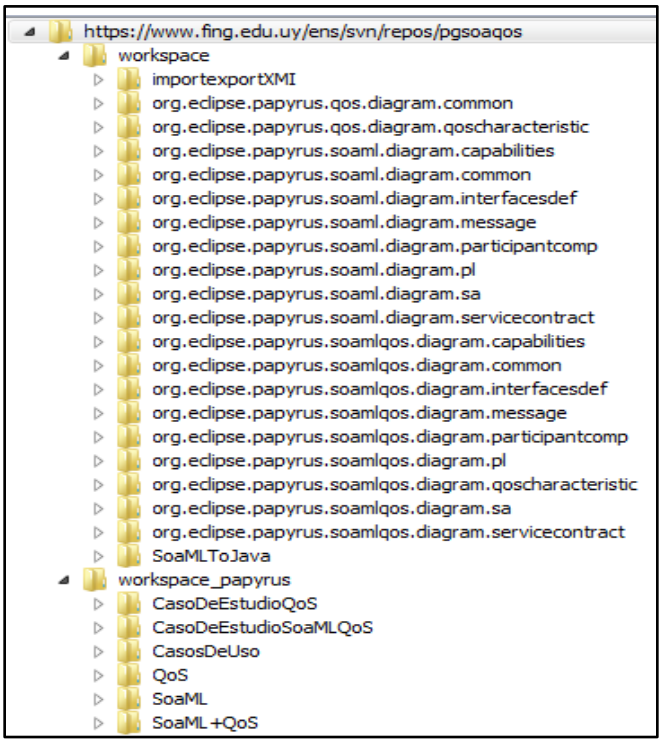

Figura 1. Estructura del repositorio

Para la documentación se optó por utilizar Google Docs ya que simplifica el trabajo en paralelo y en simultáneo, guardando las distintas versiones generadas en una carpeta compartida de Dropbox.

1.4.2. Cronograma previsto y cronograma real

El seguimiento de las tareas planificadas se establecieron mediante reuniones periódicas con la tutora. El cronograma establecido en la especificación del proyecto definía las siguientes etapas con sus actividades involucradas:

[1] https://www.fing.edu.uy/ens/svn/repos/pgsoaqos

[2] http://tortoisesvn.net/

- Estudio del estado del arte en orientación a servicios y desarrollo dirigido por modelos, en particular en lo que refiere a modelado y características asociadas, y estándares existentes.
    - estudio de la teoría de metamodelado, perfiles UML, estándar XMI para intercambio de modelos
    - estándar SoaML para modelado de servicios, y QoS para especificación de requisitos no funcionales
    - estándares MDA para desarrollo dirigido por modelo, tipos y niveles de modelos, y QVT para transformaciones
- Estudio del entorno Eclipse y desarrollo de plugins, plataforma Java EE y tecnología de Web Services, y plugins existentes para modelado SoaML y generación del código asociado.
- Definición e implementación de la solución integral para Eclipse que integre los plugins realizados y características QoS a modelos SoaML, así como la generación del código asociado.
    - Requerimientos (Casos de Uso)
    - Priorización de casos de uso, implementación de arquitectura y pruebas
    - Implementación 2dos. CU, pruebas
    - Implementación 3eros. CU, pruebas
- Caso de estudio de aplicación de modelado de servicios en SoaML con QoS con la solución Eclipse desarrollada, que podrá ser de laboratorio (ejemplo del estándar de SoaML) o real de alguna organización afín.
- Realización del informe final del proyecto de grado, correcciones y defensa.

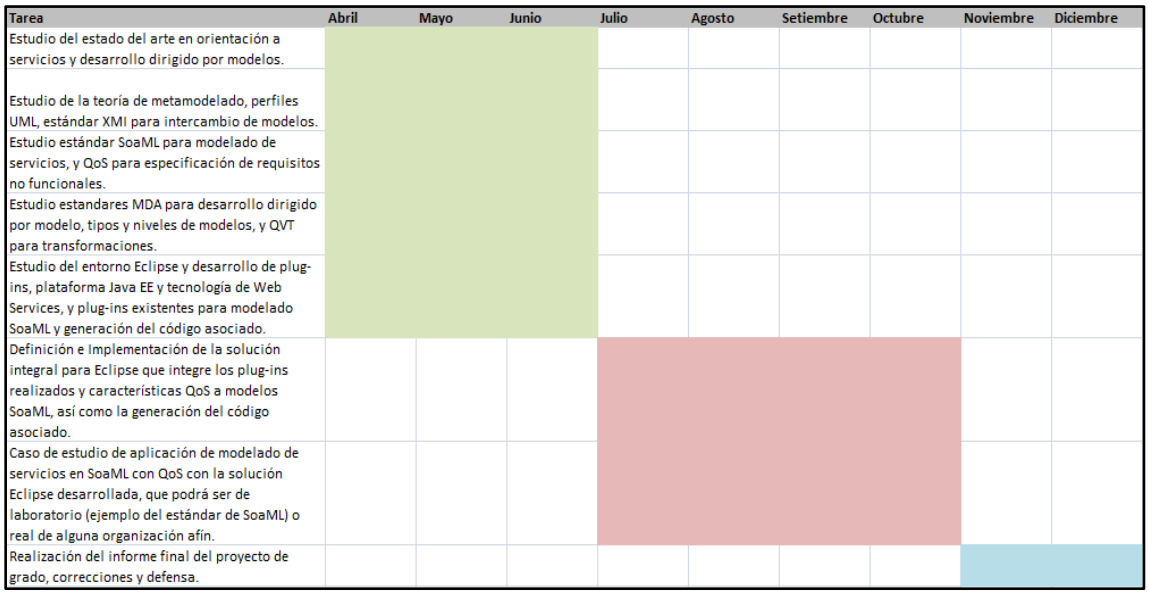

Figura 2. Diagrama de Gantt de planificación inicial del proyecto

El cronograma real se correspondió completamente con el cronograma previsto, finalizando el proyecto en el tiempo esperado con los resultados esperados.

1.5.Organización del documento

El documento se encuentra organizado en capítulos como se especifica a continuación. En el Capítulo 2 se presentan los conceptos y tecnologías investigadas a lo largo del proyecto con el objetivo de facilitar

la comprensión del informe. En el Capítulo 3 se describen los requerimientos del proyecto. En el Capítulo 4 se describe la solución propuesta en conjunto con su arquitectura, el estudio de las alternativas de desarrollo evaluadas, e implementación realizada. En el Capítulo 5, se expone el plan de verificación y los resultados obtenidos. En el Capítulo 6 se describe el caso de estudio y su realización utilizando el plugin SoaML Toolkit. Por último, en el Capítulo 7, se resume el trabajo, se presentan las conclusiones obtenidas y se comenta el posible trabajo futuro sobre el presente proyecto de grado.

Además del documento Informe Final, se pueden consultar los siguientes anexos:

- Documento del Estado del Arte
- Documento de Casos de Uso
- Documento de Arquitectura de Software
- Documento de Casos de Prueba
- Documento Técnico
- Guía de Instalación
- Manual de Usuario

2. Estado del arte

2.1.Ingeniería y Desarrollo dirigido por modelos

Antes de introducir los conceptos de ingeniería, desarrollo y arquitectura dirigida por modelos, es conveniente definir qué es un modelo. En [3] se define a un modelo como un conjunto de elementos formales, ordenados con el fin de describir algo para su posterior análisis. Cada modelo aborda un número de objetos de estudio con un grado de abstracción - cuando el grado de abstracción es alto el modelo está más cerca del lenguaje del usuario. El código fuente de un programa puede verse como un modelo del sistema de bajo nivel de abstracción escrito en un lenguaje de programación concreto (por ejemplo, Java o C#).

La ingeniería dirigida por modelos (Model Driven Engineering, MDE) es un paradigma de desarrollo de software donde los modelos son piezas fundamentales en los diferentes procesos de ingeniería de software. Dentro de este enfoque de desarrollo podemos encontrar el desarrollo dirigido por modelos (Model Driven Development, MDD), el testing dirigido por modelos y el despliegue dirigido por modelos. Las tecnologías enfocadas a la ingeniería dirigida por modelos combinan lenguajes de modelado de dominios específicos con motores de transformaciones y generadores. Dichos lenguajes formalizan la estructura de una aplicación, comportamientos y requerimientos dentro de un dominio particular [4]. La ventaja más importante de esto es que los modelos se expresan utilizando conceptos más cercanos al dominio del problema y están mucho menos relacionados a la tecnología utilizada para implementar el sistema [5]. Los generadores y motores de transformación analizan ciertos aspectos de los modelos y generan distintos tipos de artefactos como código fuente, datos de entrada de simulación y representaciones alternativas del modelo. Esto permite mantener la consistencia entre implementaciones de programas e información de análisis asociada a los requerimientos funcionales y no funcionales de los modelos [4].

El desarrollo dirigido por modelos se visualiza en [3] como la noción de que se puede construir un modelo de un sistema que luego se podrá transformar en el sistema real. Los modelos siempre fueron utilizados en la etapa de análisis y diseño de sistemas de software, y son utilizados por los programadores como referencia de lo que se desea construir. El problema de utilizar los modelos como mera documentación es que fácilmente divergen de la realidad y se vuelven obsoletos. Para poder aprovechar realmente el potencial de los modelos, MDE se basa en aprovechar al máximo los beneficios de la automatización, incluyendo la verificación automática de modelos y generación automática de programas [5]. El modelado es apropiado para formalizar conocimiento, y por lo tanto, podemos definir la sintaxis y semántica de un lenguaje a través de la construcción de un modelo del lenguaje de modelado: un metamodelo [3].

El OMG define una arquitectura de cuatro niveles de modelado (ver Figura 3) que consiste en una jerarquía de niveles de modelos, donde cada nivel (salvo el superior) corresponde a una instancia del nivel inmediatamente superior. El nivel M0 contiene los datos del usuario – los objetos de datos que manipula el software. En el nivel M1 se encuentra un modelo de los datos de usuario (modelo de M0); en este nivel se encuentran los modelos de usuario que describen los sistemas. El nivel M2 contiene los metamodelos, que son modelos de los modelos que se encuentran en M1. El metamodelo más conocido

es el lenguaje UML. Finalmente, en el nivel M3 se encuentran los meta-metamodelos como MOF, que son modelos de metamodelos [6]. MOF se describe también a sí mismo por lo cual M3 es el nivel más general.

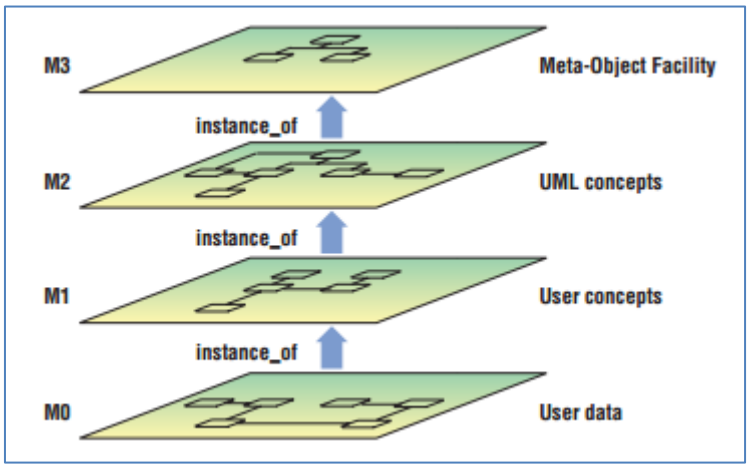

Figura 3. Arquitectura de metamodelos de [6]

2.2.Arquitectura dirigida por modelos

La Arquitectura Dirigida por Modelos (Model-Driven Architecture, MDA) [7] es un estándar del OMG que implementa el paradigma MDD. MDA provee un enfoque para especificar sistemas independientemente de las plataformas de implementación, especificar estas plataformas, elegir una de éstas y transformar la especificación del sistema a una especificación para la plataforma seleccionada. Los tres objetivos principales de MDA son la portabilidad, interoperabilidad y reusabilidad.

2.2.1. Tipos de modelos

En [7] se especifican tres tipos de modelos principales: modelo independiente de la computación (CIM), modelo independiente de la plataforma (PIM) y modelo específico a una plataforma (PSM).

El CIM es una representación del sistema desde un punto de vista enfocado en el ambiente del sistema y sus requerimientos, donde la estructura y los detalles de procesamiento del sistema están ocultos o todavía no fueron determinados. Este tipo de modelo es ocasionalmente llamado modelo de dominio y en él se utiliza un vocabulario familiar para los profesionales del dominio en cuestión. Estos modelos juegan un rol importante como nexo entre los expertos en el dominio y sus requerimientos, y los expertos en el diseño y construcción de los artefactos que cubren esos requerimientos. Ejemplos podrían ser modelos de Casos de Uso o de Procesos de Negocio.

El PIM es una representación del sistema desde un punto de vista enfocado en la operación de un sistema, ocultando todos los detalles necesarios de una plataforma en particular. Se pueden utilizar lenguajes de modelado de propósito general o lenguajes específicos al área en el que el sistema va a ser utilizado. Ejemplos serían modelos de clases de diseño UML o modelos de servicios SoaML.

El PSM es una representación del sistema atada a una plataforma en particular. En estos modelos se combinan las especificaciones del PIM con los detalles de cómo el sistema utiliza la plataforma particular.

2.2.2. Transformaciones

Uno de los conceptos más importantes en MDA es la transformación de modelos, lo que implica la conversión de un modelo de un sistema a otro modelo del mismo. En MDA existen transformaciones de PIM a PSM, de PSM a código fuente y de PIM a código fuente, siendo los dos primeros tipos de transformaciones los más habituales. En la Figura 4 se muestra la secuencia común de transformación de modelos en MDA.

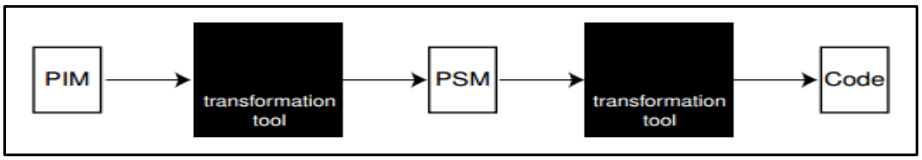

Figura 4. Ejecución de transformaciones en MDA de [8]

Los mapeos (*mapping*) en MDA proveen especificaciones para transformaciones de un PIM a un PSM de determinada plataforma [7]. Por ejemplo, se puede construir un PIM de un sistema y definir mapeos de transformación a modelos PSM con tecnología Java y .NET que finalmente serán transformados a código Java y .NET respectivamente.

Este enfoque permite centrarse en la construcción y mejora contínua de modelos PIM, ya que a partir de éstos se genera el código fuente del sistema mediante una sucesión de transformaciones. Una gran ventaja de esto es que si se quiere cambiar la tecnología del sistema alcanza con aplicar una nueva transformación sobre el PIM existente y así generar el PSM para la nueva plataforma. De esta forma se logra la reutilización de modelos y la portabilidad de sistemas.

2.2.3. Estándares de MDA

El OMG ha adoptado varias tecnologías que juntas habilitan el enfoque dirigido por modelos, por ejemplo Unified Modeling Language (UML), lenguaje de metamodelado Meta Object Facility (MOF), perfiles UML y lenguaje para intercambio de modelos XML Metadata Interchange (XMI). A continuación se describen MOF, XMI y perfiles UML por ser los principales utilizados en este proyecto. En el Anexo correspondiente al Documento del Estado del Arte se pueden ver más detalles al respecto.

2.2.3.1. Meta Object Facility (MOF)

MOF provee un framework de gestión de metadatos y un conjunto asociado de servicios de metadatos, que posibilitan el desarrollo e interoperabilidad de sistemas dirigidos por modelos y metadatos. Este estándar es la base para la definición de metamodelos en la familia de lenguajes MDA del OMG [9].

MOF introdujo los conceptos de metamodelos formales y de modelos de metadata PIM, así como también los mapeos de PIMs a PSMs. La alineación entre MOF y UML se completó con MOF 2.4 y UML 2.4, al compartir el mismo metamodelo para la definición de UML y MOF, usando restricciones OCL para definir el subconjunto del metamodelo relevante para MOF. Esta unificación en los conceptos de modelado de UML y MOF trae aparejados tres beneficios principales:

- Reglas más simples para modelar metadatos (solamente se necesita entender un subconjunto del modelado de clases UML sin anotaciones o conceptos de modelado adicionales).
- Varios mapeos existentes de MOF ahora aplican también a una amplia gama de modelos UML, incluyendo perfiles.

- Amplia gama de soporte de herramientas para metamodelado (cualquier herramienta de modelado UML puede usarse para modelar metadatos).

2.2.3.2. Perfiles UML

Los perfiles UML permiten la extensión de metaclases UML para adaptarlos a diferentes propósitos; esto incluye la habilidad de adaptar el metamodelo UML para plataformas o dominios diferentes. Los perfiles proveen un mecanismo directo para adaptar un metamodelo existente con elementos que son específicos a un dominio, plataforma o método específico. No es posible quitar ninguna de las restricciones que aplican a UML utilizando un perfil, pero es posible agregar restricciones que son específicas a éste [10].

El elemento principal de extensión es el estereotipo (*Stereotype*), y estos se definen dentro de un perfil. Un estereotipo define una extensión para una o más metaclases (*Metaclass*), y habilita la utilización de terminología o notación específica adicionalmente a las utilizadas en las metaclases extendidas. Un estereotipo es una especie de metaclase limitada que no puede ser utilizada por sí misma, y siempre debe ser usada en conjunto con una de las metaclases que extiende [10].

Los perfiles UML son paquetes (*Profile* es una especialización de *Package*) compuestos de estereotipos, y son aplicados a otros paquetes; esto significa que un perfil puede ser aplicado a otro perfil [10].

2.2.3.3. XML Metadata Interchange (XMI)

El estándar XMI [11] define los siguientes aspectos involucrados en la descripción de objetos en XML:

- La representación de objetos en términos de elementos y atributos XML.
- El mecanismo estándar de vincular objetos dentro del mismo archivo o entre distintos archivos.
- La validación de documentos XMI usando XML Schemas.
- Identidad de objetos.

XMI describe soluciones a los puntos listados anteriormente especificando reglas de producción para crear documentos y esquemas XML que comparten objetos de forma consistente [11].

Un esquema XML provee una vía para que un procesador XML pueda validar la sintaxis y alguna de las semánticas de un documento XML. El estándar XMI provee reglas para generar esquemas de cualquier modelo basado en MOF transmisible por XMI [11].

El proceso de producción de documentos XML está definido en XMI como un conjunto de reglas de producción, que cuando son aplicadas a un modelo resultan en un documento XML. Se puede reconstruir un modelo a partir de un documento XML aplicando las reglas de forma inversa [11].

2.3.Computación y Arquitectura Orientada a Servicios

El término “orientación a servicios” implica la división de los diferentes problemas en problemas más pequeños a resolver. La lógica requerida para resolver un problema grande puede ser construida de mejor forma y más fácilmente si es dividida en partes más pequeñas relacionadas entre sí [12]. En otras palabras, la orientación a servicios implica encapsular ciertas funcionalidades de negocio en servicios autónomos que podrán ser compartidos para que otros servicios puedan utilizarlos y combinarlos para crear otros servicios.

La Computación Orientada a Servicios es definida en [13] como un paradigma que utiliza servicios para soportar el desarrollo rápido de aplicaciones evolutivas, interoperables y distribuidas masivamente a bajo costo. Estos servicios son entidades autónomas e independientes de la plataforma, que pueden ser descriptas, publicadas y descubiertas para que otros servicios las utilicen. SOC impulsa la composición de aplicaciones distribuidas mediante el uso de servicios con un bajo acoplamiento.

La Arquitectura Orientada a Servicios, inspirada en el paradigma SOC, es definida en [12] como un modelo en el que la lógica del sistema es dividida en unidades lógicas más pequeñas (servicios) que pueden ser distribuidas de forma independiente.

En [13] se define SOA como una forma de diseñar un sistema de software que provea servicios a aplicaciones de usuarios finales u otros servicios distribuidos en una red, a través de la publicación de interfaces fáciles de encontrar.

SOA no está atada a ninguna tecnología [14], aunque actualmente la estrategia más común de implementación es el uso de *web services*, mediante la utilización de estándares como HTTP, SOAP, WSDL y UDDI [15].

Como se mencionaba anteriormente, un sistema basado en SOA está compuesto principalmente por servicios. El propósito fundamental de un servicio es el de representar una unidad de trabajo reutilizable. Cada servicio es una porción de funcionalidad expuesta con tres características esenciales: su autonomía, la independencia de la plataforma y la capacidad de ser localizado, invocado y combinado con otros servicios de forma dinámica [14]. Asimismo, el sistema está compuesto por consumidores de servicios, que son clientes de las funcionalidades provistas por los servicios (por ejemplo, aplicaciones de usuario final, sistemas u otros servicios) y por infraestructura de SOA, que conecta consumidores de servicios con los servicios [15].

2.4.Modelado de servicios con SoaML

SoaML [16] es un lenguaje de modelado de servicios, el cual ofrece un *framework* completo e intuitivo para representar artefactos SOA en UML, como ser participantes, puertos, servicios, contratos, etc. SoaML está implementado como un perfil UML, lo que permite su utilización en cualquier herramienta de modelado UML.

2.4.1. Elementos de SoaML

A continuación, se definen los principales elementos del metamodelo de SoaML según [16].

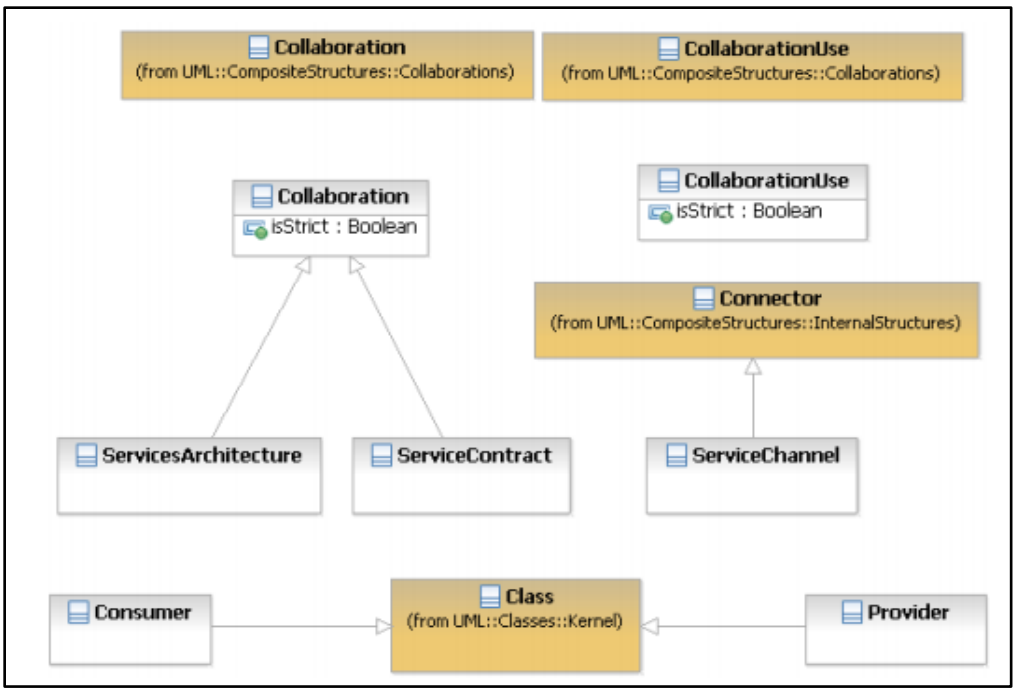

Figura 5. Metamodelo de Contratos de Servicio y Arquitecturas de Servicio de [16]

2.4.1.1. *Collaboration*

Una Collaboration representa un patrón de interacción entre roles. Esta interacción puede ser definida informalmente o puede representar acuerdos formales o requerimientos que deben ser cumplidos estrictamente. Collaboration extiende a Collaboration de UML, con el agregado de la propiedad *isStrict*, la cual indica si ésta representa un patrón estricto de interacción.

2.4.1.2. *ServicesArchitecture*

Una ServicesArchitecture es la visión de alto nivel de una SOA. Describe de qué manera un conjunto de participantes interactúan entre sí por un propósito, proporcionando y consumiendo servicios expresados como contratos de servicios.

2.4.1.3. *ServiceContract*

Un ServiceContract es la especificación del acuerdo entre proveedores y consumidores de un servicio. Ésta incluye qué información, productos, valores y obligaciones serán intercambiados entre ellos.

2.4.1.4. *ServiceChannel*

Un ServiceChannel provee una vía de comunicación entre solicitudes de consumidores y servicios de proveedores.

2.4.1.5. *Consumer*

Un Consumer modela la interfaz provista por el consumidor de un servicio. El consumidor del servicio es generalmente el que inicia la interacción. Las interfaces de los consumidores son utilizadas como tipo de un ServiceContract y están atadas a los términos y condiciones de dicho contrato de servicio.

2.4.1.6. Provider

Un Provider modela la interfaz provista por el proveedor de un servicio. El proveedor del servicio es el que generalmente da respuestas ante una interacción con un consumidor. Las interfaces de los proveedores son utilizadas como tipo de un ServiceContract y están atadas a los términos y condiciones de dicho contrato de servicio.

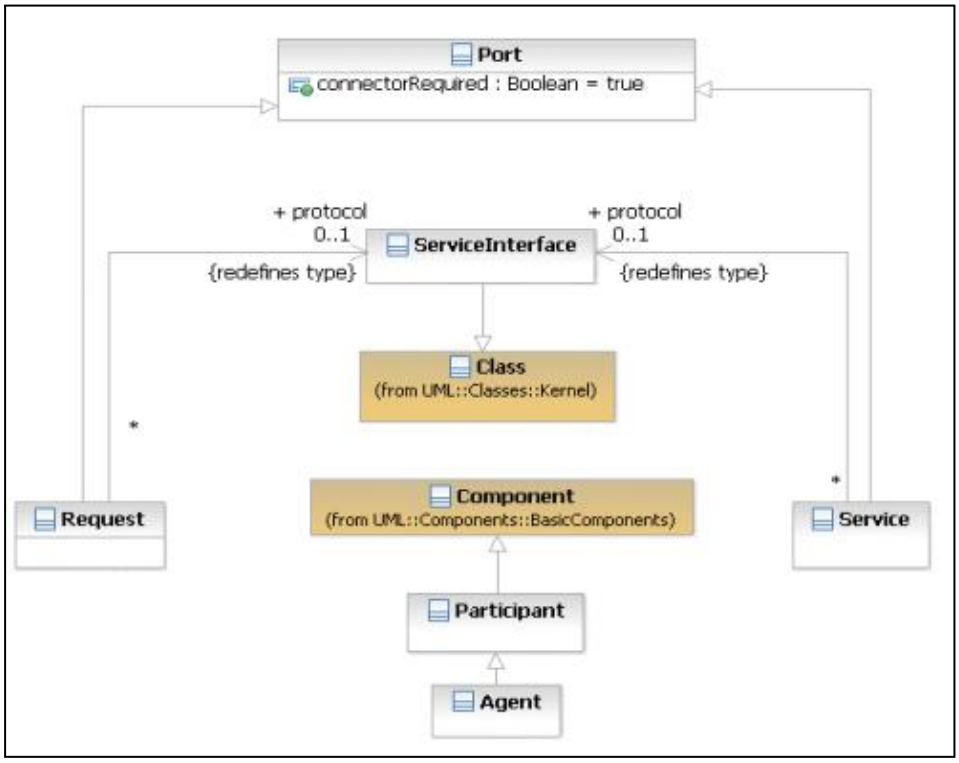

Figura 6. Metamodelo de Interfaces de Servicios y Participantes de [16]

2.4.1.7. Participant

Un Participant es una persona, organización o sistema que provee y/o consume servicios a través de sus puertos de servicios. Éste implementa cada una de las operaciones de servicios provistas y dicha implementación debe ser consistente con las operaciones, protocolos y restricciones especificadas por la ServiceInterface.

2.4.1.8. Agent

Un Agent es una especie de entidad autónoma que puede adaptarse a su entorno e interactuar con éste. Describe un conjunto de instancias de agentes que tienen características, restricciones y semántica en común. Los agentes en SoaML son también participantes, que proveen y consumen servicios.

2.4.1.9. Port

Los participantes proveen o consumen servicios a través de puertos. Un puerto es la parte de un participante en la que se produce la interacción para consumir o proveer un servicio. Un puerto en el que se ofrece un servicio se llama Puerto de Servicio, mientras que un puerto en el que se consume un servicio se llama Puerto de Solicitud. Este elemento extiende a Port de UML.

2.4.1.10. ServiceInterface

Una ServiceInterface define la interfaz y responsabilidades de un participante de proveer o consumir un servicio. Es usada como tipo de un puerto de Servicio o de Solicitud.

2.4.1.11. Request

Request es una extensión de Port, la cual especifica una característica de un Participant que representa un servicio que el Participant necesita y consume de otros participantes. El servicio es definido por una ServiceInterface.

2.4.1.12. Service

Service es una extensión de Port, la cual especifica una característica de un Participant que representa un servicio que el Participant provee y ofrece para ser consumido por otros participantes. El servicio es definido por una ServiceInterface.

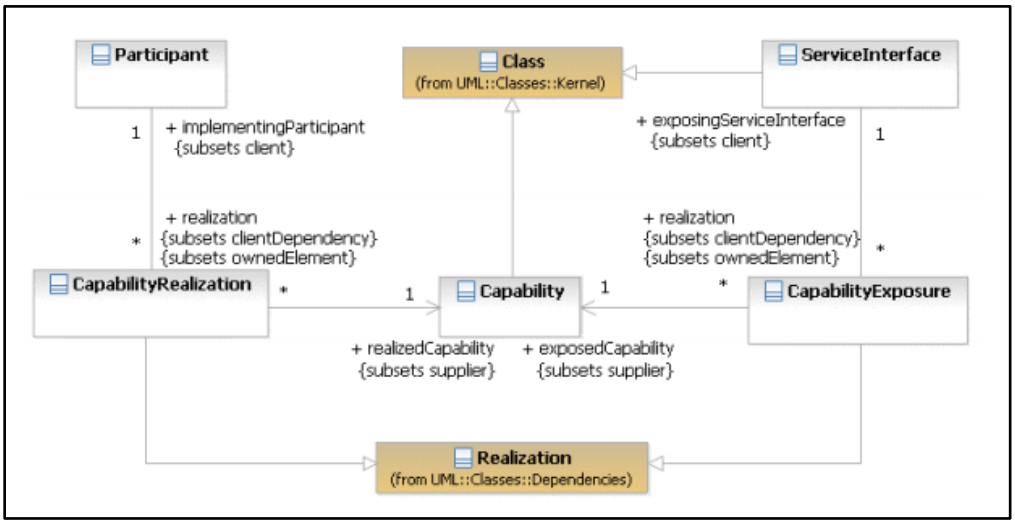

Figura 7. Metamodelo de Funcionalidades de [16]

2.4.1.13. Capability

Una Capability especifica un conjunto de funcionalidades que un servicio puede ofrecer. Se utiliza para identificar los servicios necesarios y para organizarlos en catálogos.

2.4.2. Diagramas de SoaML

A continuación se explicarán los distintos diagramas que se pueden definir en SoaML según [16] y también cómo se modelan los distintos elementos del lenguaje para definir cierta arquitectura orientada a servicios.

2.4.2.1. Diagrama de Arquitectura de Servicios

El diagrama de Arquitectura de Servicios (Service Architecture) es una vista de alto nivel de abstracción en donde se define un conjunto de servicios y se muestra cómo los distintos participantes cumplen diferentes roles interactuando entre sí para alcanzar ciertos objetivos.

En la Figura 8 se puede ver un ejemplo de un diagrama de Services Architecture y cómo se relacionan los elementos entre sí.

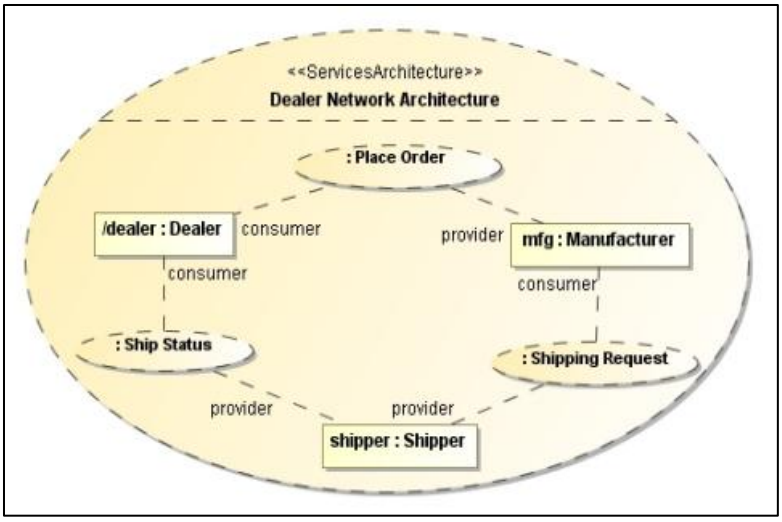

Figura 8. Ejemplo de Arquitectura de Servicios de [16]

2.4.2.2. Diagrama de Contrato de servicios

El diagrama de Contrato de Servicios (Service Contract) define los términos, las condiciones y las interfaces con las cuales los participantes que interactúan deben estar de acuerdo para que el servicio sea ejecutado. La base del Service Contract es también una colaboración UML que se centra en las interacciones que intervienen para proporcionar un servicio.

Un Participant juega un rol como el proveedor o consumidor de los servicios especificados por el Service Contract. Cada Rol es definido como una Interface o ServiceInterface. En el Service Contract se definen las relaciones entre un conjunto de roles definidos por las Interfaces y/o ServiceInterfaces.

En la Figura 9 se puede ver un ejemplo de diagrama de Service Contract, donde se define el contrato *Purchasing Service* compuesto por otros dos contratos *Place Order* y *Returns Service*. Además se muestran los roles definidos para el contrato (*buyer* y *seller*) que a su vez son consumidores y proveedores de los servicios definidos en los contratos incluidos dentro del mismo.

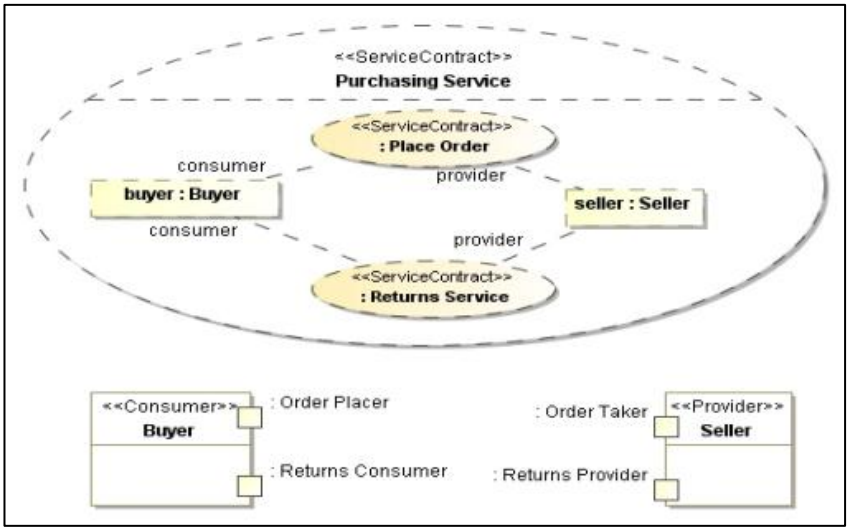

Figura 9. Ejemplo de Contrato de Servicio de [16]

2.4.2.3. Diagrama de Participantes como Clase

El diagrama de Participantes como Clase se utiliza para definir los distintos participantes y los puertos desde los cuales proveen o consumen servicios. Un Participant representa una parte o componente que provee y/o consume servicios. En el ámbito de negocios un Participant puede ser una persona, organización o sistema.

Un Participant tiene puertos los cuales utilizan el estereotipo Service y Request, y son el punto de interacción donde los servicios son ofrecidos o consumidos respectivamente. Un Participant que ofrece un servicio tiene un puerto Service cuyo tipo es una ServiceInterface de tipo Provider. Este participante implementa dicha interfaz de servicio. En cambio, un Participant que consume un servicio tiene un puerto Request cuyo tipo es una ServiceInterface de tipo Consumer. Este participante utiliza la interfaz de servicio provista por otro participante.

En la Figura 10 se puede ver un ejemplo de diagrama de Participantes como Clase, en donde el Participant *Dealer* tiene un puerto *Request* de tipo interfaz *Shipment status*. Esto quiere decir que el participante utiliza dicha interfaz. Por otro lado, el Participant *Shipper* tiene un puerto *Service* también de tipo interfaz *Shipment status*, lo que quiere decir que el participante provee dicha interfaz.

Figura 10. Ejemplo de diagrama de Participantes como Clase de [16]

2.4.2.4. Diagrama de Participantes como Componente

El diagrama de Participantes como componente sirve para especificar los participantes y las operaciones que involucran los servicios. Este diagrama también se puede utilizar para mostrar la reutilización de servicios existentes que se tienen que adaptar al contexto en el cual se quiere que funcione. La adaptación se hace definiendo un adaptador (Adapter), el cual se encargará de realizar cualquier cambio requerido en los datos, protocolo o proceso. En particular, se utiliza para unificar servicios *bottom-up* y *top-down*, en donde el *bottom-up* está acoplado a la tecnología que lo soporta y el *top-down* es más general y menos acoplado. El Adapter define la conexión entre los dos. En la Figura 11 se puede ver un diagrama de Participantes como Componente.

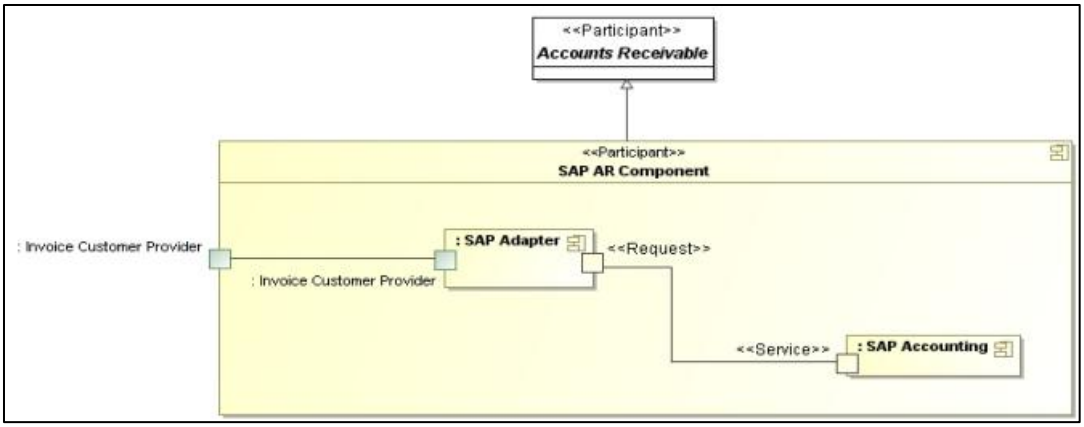

Figura 11. Ejemplo de diagrama de Participantes como Componente de [16]

2.4.2.5. Diagrama de Interfaces

El diagrama de interfaces permite modelar las distintas interfaces y las interfaces de servicios con las operaciones que proveen para realizar los servicios. También permite modelar las relaciones de dependencia entre las interfaces existentes. En la Figura 12 se puede ver un ejemplo diagrama de Interfaces.

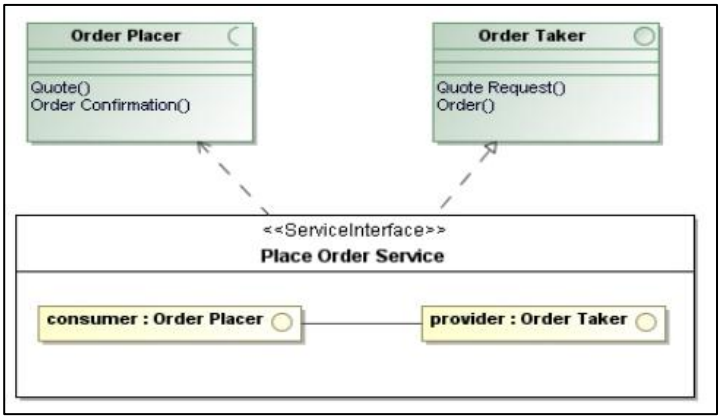

Figura 12. Ejemplo de diagrama de Interfaces de [16]

2.4.2.6. Diagrama de Capacidades

Un diagrama de Capacidades (Capability) es útil para expresar una arquitectura de servicios en términos de las capacidades lógicas de los servicios sin necesidad de conocer los participantes. Aunque los consumidores de servicios no deben preocuparse por cómo se implementa un servicio, es importante ser capaz de especificar el comportamiento de un servicio o la capacidad que realiza o implementa un ServiceInterface.

2.4.2.7. Diagrama de Mensajes

El diagrama de Mensajes tiene elementos que sirven para modelar los distintos tipos de datos. Se pueden definir *Message Type*, *Attachment* y *Property*. El tipo *MessageType* se utiliza para especificar la información intercambiada entre participantes que consumen y proveen un servicio, y puede ser definido como de tipo *Class*, *DataType* o *Enumeration.* El tipo *Attachment* es un componente de un mensaje y está adjunto a él. El tipo *Property* es una característica estructural y puede ser designado

como un identificador de propiedad, una propiedad que se puede utilizar para distinguir o identificar instancias del clasificador que las contiene. En la Figura 13 se puede ver un ejemplo de diagrama de Mensajes.

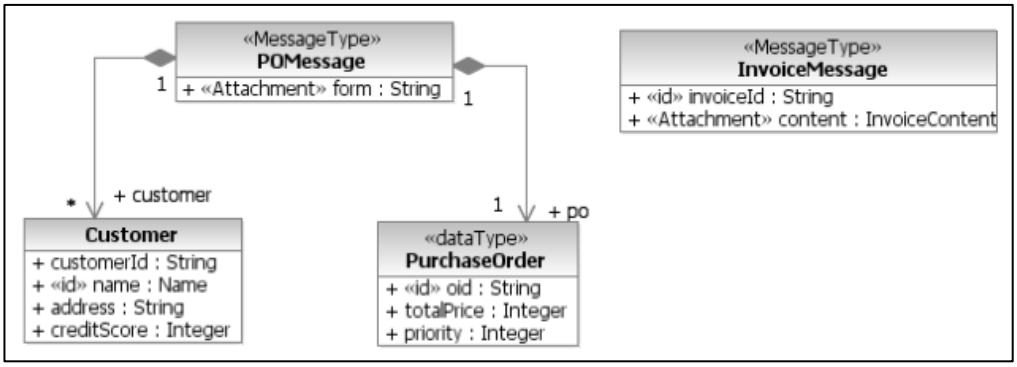

Figura 13. Ejemplo de diagrama de Mensajes de [16]

2.5.Calidad de servicios (QoS)

La calidad de servicio (*Quality of Service, QoS*) se puede definir como un conjunto de características perceptibles expresado en un lenguaje amigable con parámetros cuantificables que pueden ser subjetivos u objetivos [17]. QoS comprende atributos no funcionales importantes tales como las medidas de rendimiento (por ejemplo, tiempo de respuesta), atributos de seguridad, integridad transaccional, fiabilidad, escalabilidad y disponibilidad [13].

También el término calidad de servicio se utiliza a menudo para hacer referencia a la colección de requerimientos de calidad de un servicio. Podría pasar que diferentes servicios provean la misma funcionalidad para los consumidores, por lo tanto las especificaciones de QoS para estos servicios son esenciales a la hora de determinar el servicio más adecuado [18].

2.5.1. Atributos de calidad en SOAs

El diseño de la arquitectura de un sistema se ve directamente influenciado por los requerimientos de calidad y otros aspectos no funcionales, los cuales son importantes satisfacer para alcanzar los objetivos de negocios de la organización [19].

En [20] se especifican factores de calidad de servicios web, con la definición y explicación de los sub-factores. Los factores de calidad descritos en esta especificación son: valor de negocio, medición de nivel de servicio, interoperabilidad, procesamiento de negocio, gestionabilidad y seguridad. Estos factores pueden categorizarse en dos grupos: grupo de calidad de negocio y grupo de calidad de sistema. El primero solamente está compuesto por el factor calidad de nivel de negocio; el segundo está compuesto por una parte de calidad variante y otra de calidad invariante. La parte de calidad variante incluye factores de calidad cuyos valores pueden cambiar dinámicamente en tiempo de ejecución mientras el servicio se está usando. De forma contraria, la parte invariante refiere a los factores de calidad cuyos valores son determinados cuando el desarrollo del servicio es completado.

A continuación describiremos algunos de los factores y subfactores de calidad especificados en [20], y algunos atributos de calidad que según [19] hay que tener en cuenta a la hora de diseñar e implementar una arquitectura SOA.

2.5.1.1. Interoperabilidad

La interoperabilidad es la capacidad de una colección de entidades de compartir información específica y operar con dicha información bajo acuerdos semánticos operacionales [19].

En [20] se destaca la importancia de la interoperabilidad en los servicios web, dado que el servidor y el cliente pueden estar desplegados en plataformas diversas. La calidad de interoperabilidad es definida como un factor de calidad que evalúa si un sistema compuesto por servicios web está implementado utilizando los estándares correctos y de forma afín a la especificación.

2.5.1.2. Fiabilidad

La fiabilidad es la capacidad que tiene un sistema de mantenerse operativo en el tiempo. Cuando se habla de fiabilidad de servicios, se refiere a que los servicios no fallan o en caso de hacerlo reportan las fallas al usuario del servicio [19].

2.5.1.3. Disponibilidad

La disponibilidad es el grado en que un sistema o componente está operativo y accesible cuando es requerido para su uso [19]. Asimismo, la disponibilidad es uno de los subfactores de Seguridad en [20], ya que una de las formas de ataque a servicios web publicados en Internet es la denegación de servicio, que apunta a dejar al servicio inactivo.

2.5.1.4. Usabilidad

La usabilidad es una medida de la calidad de la experiencia del usuario en la interacción con la información o servicios. Para mejorar la usabilidad en un sistema con una SOA, los proveedores de servicios deben considerar la granularidad en los datos, el desarrollo de servicios para soportar la usabilidad (por ejemplo, un servicio para cancelar un pedido o deshacer el último pedido), y funcionalidades que permitan el restablecimiento del contexto de un servicio en caso de pérdida de conexión (por ejemplo, en un carro de compras online) [19].

2.5.1.5. Seguridad

La seguridad en los sistemas de software es asociada con cuatro principios [19]:

1. Confidencialidad: Solamente sujetos autorizados pueden acceder a la información o servicio.
2. Autenticidad: Se puede confiar en que el autor o emisor es el responsable de la información.
3. Integridad: La información no está corrupta.
4. Disponibilidad: La información o servicio están disponibles de forma apropiada.

En una SOA, a menudo los mensajes contienen datos en algún formato de texto (XML por ejemplo) y esos datos podrían contener información personal. Una buena práctica es la encriptación de mensajes para preservar la privacidad de los datos. También, los servicios pueden tener un acceso restringido según el usuario del servicio por lo que se debería tener un mecanismo de autorización que permita configurar y otorgar permisos para un usuario específico, grupo de usuarios o rol [19].

2.5.1.6. Desempeño (Performance)

El desempeño está asociado al tiempo de respuesta (cuánto tarda el procesamiento de un pedido), rendimiento (cuántos pedidos pueden ser procesados por unidad de tiempo) y capacidad (cuánta

demanda puede soportar el sistema cumpliendo con los requerimientos de tiempo de respuesta y rendimiento) [19].

2.5.1.7. Escalabilidad

La escalabilidad es la habilidad que tiene una SOA de funcionar correctamente cuando el sistema cambia de tamaño o en volumen con el fin de satisfacer las necesidades de los usuarios, sin que se vean degradados otros atributos de calidad [19].

2.5.1.8. Extensibilidad

La extensibilidad es la facilidad con la que las funcionalidades de los servicios pueden extenderse sin afectar otros servicios [19].

2.5.1.9. Adaptabilidad

La adaptabilidad es la facilidad con la que un sistema puede cambiar para adaptarse a modificaciones en los requerimientos [19].

2.5.1.10. Capacidad de testing

La capacidad de testing es el grado en que un sistema o servicio facilita el establecimiento de criterios de prueba y la realización de las mismas para determinar si se ha cumplido con esos criterios [19].

2.5.1.11. Auditabilidad

La auditabilidad es el grado en que un sistema apoya auditorías, ya sea financieras o legales, mediante el mantenimiento de registros de datos del sistema [19].

2.5.1.12. Operabilidad y Capacidad de despliegue

La operabilidad y capacidad de despliegue refieren a la capacidad de los sistemas basados en SOA de operar en un ambiente de operaciones automatizadas y con recuperación automática [19].

2.5.1.13. Modificabilidad

La modificabilidad es la capacidad de realizar cambios en un sistema de forma rápida y rentable [19].

2.5.1.14. Valor de negocio

El valor de negocio de un servicio web es el valor económico resultante de aplicar servicios web en un negocio y la calidad de éste proporciona una perspectiva del negocio que permite seleccionar correctamente los servicios, evaluando el valor de negocio de los servicios web [20].

2.5.2. Modelos de QoS para servicios

En esta sección presentaremos dos modelos de calidad de servicios: un modelo conceptual de calidad para servicios web propuesto por OASIS y otro de medición de la ejecución de procesos de negocio.

2.5.2.1. Quality Model for Web Services

En el 2005 el comité técnico Web Service Quality Model de la organización OASIS [21] publicó un modelo conceptual de calidad para servicios web [22], el cual consiste de tres componentes: factor de calidad (*Quality Factor*), actividad de calidad (*Quality Activity*) y rol de calidad (*Quality Associate*). Este modelo tiene como objetivo contribuir a que las personas involucradas en el ciclo de vida de los

servicios web entiendan cómo describir y evaluar la calidad de éstos. El factor de calidad refiere a un grupo de elementos que representan las propiedades funcionales y no funcionales de los servicios web; el componente rol de calidad refiere a los roles, tareas de las organizaciones o personas relacionadas a los servicios web; y la actividad de calidad refiere a las acciones realizadas por los roles para mantener la estabilidad de la calidad de los servicios web. El modelo de calidad de servicios web ilustrado en la Figura 14 muestra a los componentes de calidad y sus relaciones. A continuación se describe brevemente cada uno.

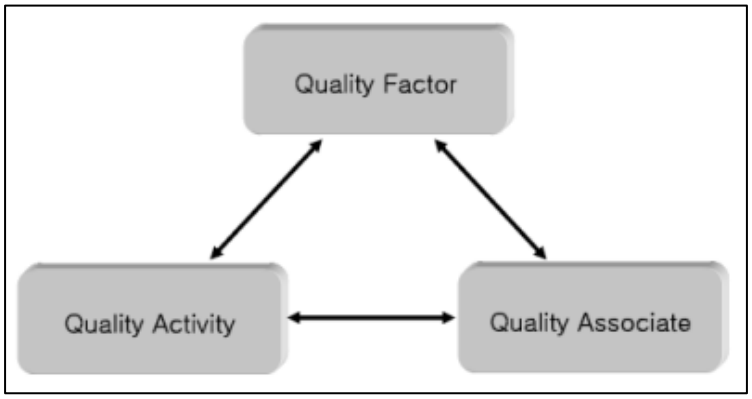

Figura 14. Modelo de Calidad de Servicios Web de [22]

2.5.2.1.1. Factores de calidad de servicios web

La calidad de los servicios web puede ser considerada en tres capas: capa de nivel de negocio, capa de nivel de servicio y capa de nivel de sistema; cada una de éstas tiene uno o más sub-factores de calidad. La capa de nivel de negocio es la calidad de representar el valor de negocio percibido por el usuario cuando utiliza los servicios web. La capa de nivel de servicio es la calidad mensurable de performance de los servicios web percibida por el usuario al utilizar los servicios web. Este factor de calidad incluye los problemas de performance como estabilidad, escalabilidad y tiempo de respuesta. Por último, la capa de nivel de sistema puede ser dividida en la capa de interoperabilidad y la capa de gestión y seguridad. La primera determina si los servicios web desarrollados en distintos ambientes y por distintos desarrolladores pueden interoperar de forma correcta, y la segunda indica la calidad de gestión y el nivel de las acciones de neutralización de ataques o accesos no autorizados.

2.5.2.1.2. Roles de calidad de servicios web

Se le llama rol de calidad a la función que cumple una persona que participa en algún paso del ciclo de vida de un servicio web. A continuación describiremos brevemente algunos de los roles especificados en el modelo:

- ***Stakeholder:*** Es quien solicita el desarrollo de un servicio web a un desarrollador, especificando los requerimientos de calidad con los que debe cumplir el servicio. Utiliza el modelo de calidad para testear si los servicios web cumplen con los niveles de calidad especificados en los requerimientos.
- ***Desarrollador:*** Es quien considera los requerimientos de calidad y diseña la estructura necesaria para cumplir con los requerimientos.

- ***Proveedor:*** Provee los servicios web implementados por el desarrollador. La calidad de los servicios web tiene una gran importancia del lado del proveedor, ya que éste puede perder ganancias si un competidor provee servicios web de mejor calidad.
- ***Consumidor***: Es el usuario que utiliza los servicios web. De existir múltiples servicios web que ofrecen las mismas funcionalidades, el consumidor elegirá el que ofrezca la mejor calidad de servicio.

En la Figura 15 se muestran todos los roles de calidad y cómo interactúan entre sí.

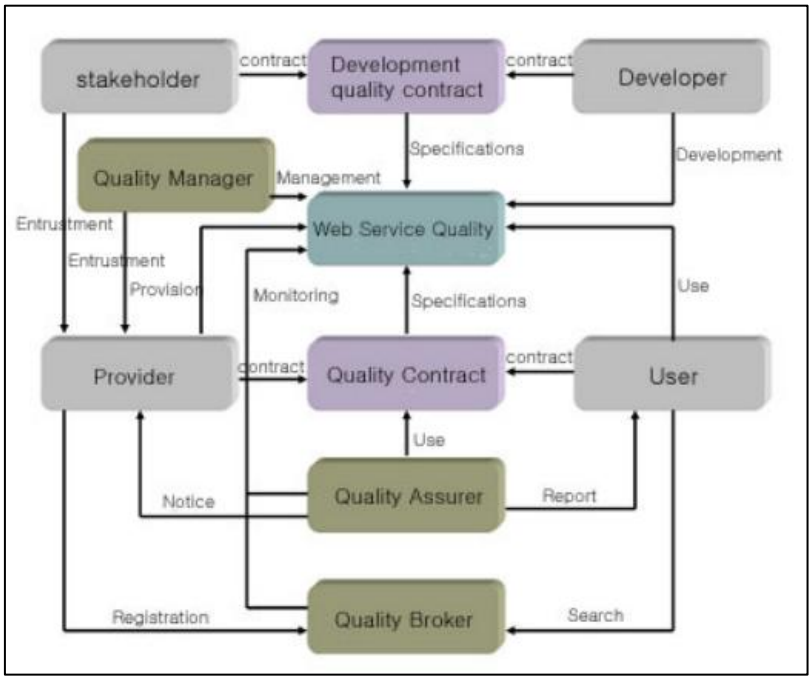

Figura 15. Roles de calidad de [22]

2.5.2.1.3. Actividades de calidad de servicios web

El componente Actividades de Calidad de este modelo consiste de varias actividades para mantener la estabilidad de los contratos de calidad de servicios web entre los roles de calidad. Existen distintos tipos de actividades, entre las que se encuentra la de tipo Contrato, la cual es la más importante y es la cooperación entre roles con los detalles de calidad de desarrollo, calidad de uso y calidad de gestión. Otras actividades son la búsqueda de detalles de calidad y búsqueda del servicio web con mejor calidad, desarrollo de servicios, publicación de detalles y niveles de calidad en un agente QoS y notificación de incumplimientos en los niveles de calidad requeridos.

El desarrollo de servicios web incluye el diseño, implementación, testing unitario e integración de servicios. Para asegurar que los servicios web son desarrollados con los niveles de calidad requeridos, los desarrolladores deben considerar la calidad en cada una de las tareas del desarrollo. Un stakeholder debe evaluar los servicios web que están siendo desarrollados e inspeccionarlos adecuadamente. Para esto, se realiza un contrato de calidad de desarrollo entre el desarrollador y el stakeholder, que debe contener los detalles de calidad que se deben alcanzar en el desarrollo y debe contener una lista de verificación para utilizarse en las inspecciones.

2.5.2.2. Modelo de Medición de la Ejecución de Procesos de Negocio (BPEMM)

El modelo de medición de ejecución de procesos de negocio [23] (*Business Process Execution Measurement Model,* BPEMM) es un modelo que integra medidas para procesos de negocios implementados con servicios. La medición de la ejecución de procesos de negocios provee una base para analizar el comportamiento real de una organización; ayuda a detectar deviaciones en los comportamientos planificados y a descubrir oportunidades de mejora en los procesos de negocio. Este modelo es parte de un framework llamado MINERVA [24], el cual aplica desarrollo dirigido por modelos y computación orientada a servicios para la mejora continua de procesos de negocio en las organizaciones.

Las medidas de BPEMM tienen tres elementos principales:

1. Objetivo: es la meta definida para la organización, proyecto o proceso, y es definida desde varios puntos de vista y modelos.
2. Pregunta: describe cómo será evaluado cada objetivo desde el punto de vista de una característica de calidad.
3. Medida: es un conjunto de datos objetivos o subjetivos, que es utilizado para responder cada pregunta de forma cuantitativa. Se utilizan tres tipos de medidas:
   3.1. Medida Base: es la medida de un atributo, sin dependencias con otras medidas, y cuyo enfoque de medición es un método de medición.
   3.2. Medida Derivada: es una medida derivada de medidas base y otras medidas derivadas, cuyo enfoque de medición es una función de medición.
   3.3. Indicador: es una medida derivada de otras medidas, cuyo enfoque de medición es un modelo analítico con un criterio de decisión asociado.

Este modelo agrupa las medidas de ejecución de acuerdo a tres vistas específicas: procesos de negocio genéricos, ejecución eficiente (*Lean*), y ejecución de servicios. La vista de procesos de negocio genéricos incluye la duración de actividades, duración del proceso completo, costos, roles involucrados y la calidad percibida del usuario. La vista Lean define medidas de ejecución utilizadas para recolectar información para la detección de derroches en la ejecución de procesos de negocio. Apunta a encontrar actividades o caminos en el proceso que puedan guiar a una optimización y mejora del proceso de negocio. La vista de ejecución de servicio contiene medidas asociadas a la ejecución de los servicios que implementan el proceso de negocio, teniendo en cuenta requerimientos QoS. En este proyecto nos es de mayor interés la vista de ejecución de servicios, ya que las medidas definidas en esta vista están fuertemente ligadas al modelado de características QoS.

Para calcular las medidas de cada servicio invocado, la información de cada invocación debe ser registrada en un log. El servidor de aplicaciones que ejecuta el servicio puede registrar el tiempo, el origen y las credenciales de cada invocación del servicio. A continuación en la Tabla 1 se muestran las medidas definidas en BPEMM para el tiempo de respuesta de servicio a modo de ejemplo.

Tabla 1. Medidas de Tiempo de Respuesta para la ejecución de servicios de [23]

Objetivo	**O1**	**Garantizar el tiempo de respuesta promedio del servicio en L1 segundos**
Pregunta	**P1**	**¿Cuál es el tiempo de respuesta promedio actual del servicio?**
Medidas	M1 (base)	Tiempo de invocación del servicio desde la actividad en el PN (IT = timestamp)
	M2 (base)	Tiempo de habilitación del servicio (ET = timestamp)
	M3 (base)	Tiempo de inicio del servicio (ST = timestamp)
	M4 (base)	Tiempo de terminación del servicio (CT = timestamp)
	M5 (base)	Tiempo de falla del servicio (FT = timestamp)
	M6 (base)	Tiempo de respuesta desde el servicio a la actividad en el PN (AT = timestamp)
	M7 (derivado)	Tiempo de procesamiento del servicio (SPoT = CT – ST)
	M8 (derivado)	Tiempo de latencia del servicio (SLaT = ST – ET)
	M9 (derivado)	Tiempo de respuesta del servicio (SRpT = SPoT + SLaT)
	M10 (derivado)	Tiempo de contestación desde el PN (SAnT = AT – IT)
	M11 (indicador)	Tiempo de procesamiento vs. Tiempo de latencia (STI = SLat/SPoT) Criterio de decisión = Index DC
	M12 (indicador)	Tiempo de respuesta promedio en todos los casos del PN (ASRpT = ∑SRpT/Tiempo total de ejecuciones en todos los casos del PN) Criterio de decisión = Index DC
	M13 (indicador)	Tiempo de contestación promedio en todos los casos del PN (ASAnT = SAnT/Tiempo total de ejecuciones en todos los casos del PN) Criterio de decisión = Index DC
Criterio de decisión	Index DC	R1: 0<=TTI<= L1 = ''LOW'' = GREEN; R2: L1<= TTI < L2 = ''MEDIUM'' = YELLOW; R3: L2<= TTI = ''HIGH'' = RED

En [23] se pueden ver todas las tablas de las medidas de ejecución definidas, que serán utilizadas en el capítulo del caso de estudio en que usamos BPEMM para especificar las QoS de los servicios ya modelados en SoaML para la implementación de un proceso de negocio seleccionado.

2.6.Modelado de calidad de servicios con QoS

La especificación *Quality of Service & Fault Tolerance* [17] define un conjunto de extensiones UML para representar conceptos de calidad de servicio y tolerancia a fallas, e integra estas extensiones en dos frameworks generales: framework de modelado QoS y framework de modelado de tolerancia a fallas. En este proyecto trataremos únicamente el framework general para modelado QoS, el cual provee la habilidad de asociar los requerimientos y propiedades de calidad a elementos del modelo, y así introducir aspectos no funcionales a los modelos UML.

El framework de modelado QoS define un metamodelo y un perfil UML. El metamodelo especifica un lenguaje abstracto de lenguajes de modelado de conceptos QoS, los cuales son representados como metaclases. El perfil define extensiones al metamodelo con el propósito de adaptarlo a una plataforma o dominio específico, sin cambiar su semántica [17].

Tanto el metamodelo como el perfil QoS se dividen en tres paquetes: el paquete *QoSCharacteristics,* que incluye los elementos de modelado que describen las características QoS, el paquete *QoSContraints,* que incluye los elementos de modelado que describen los contratos y restricciones QoS y el paquete *QoSLevels*, que incluye los elementos de modelado para la especificación de los niveles y transiciones QoS [17]. Este informe se centrará en los paquetes QoSCharacteristics y QoSContraints; por mayor información sobre el paquete QoSLevels referirse al documento anexo Estado del Arte.

2.6.1. Elementos de QoS

A continuación se definen los principales elementos del metamodelo de QoS según [17].

2.6.1.1. QoS Characteristic

Una *QoS Characteristic* representa una característica cuantificable de servicios y describe aspectos no funcionales. Por ejemplo, los atributos de calidad definidos en la sección 2.5.1 pueden ser modelados como elementos QoS Characteristic. Este elemento extiende la metaclase *Classifier* de UML.

2.6.1.2. QoS Dimension

Una *QoS Dimension* define una medida para cuantificar una QoS Characteristic. Por ejemplo, las medidas definidas en BPEMM [23] pueden ser modeladas como elementos QoS Dimension.

Una QoS Dimension contiene tres propiedades, las cuales se describen a continuación.

- ***Direction***: Indica de qué manera deben compararse dos valores distintos de una misma QoS Dimension. Éste es un enumerado cuyo valor puede ser creciente, decreciente o indefinido. Cuando el atributo toma el valor creciente, cuanto mayor es el valor, mejor es la calidad. En cambio, si el atributo toma el valor decreciente, cuanto menor es el valor, mejor es la calidad. Por ejemplo, si estamos midiendo el tiempo de respuesta de un sistema, será mejor aquel que tenga menor tiempo, por lo tanto la dirección será decreciente.
- ***Unit***: Indica la unidad de los valores de la dimensión. Por ejemplo, si se está midiendo el tiempo de respuesta, entonces la unidad podrá ser milisegundos.
- ***StatisticalQualifier***: Representa el tipo de clasificación estadística (valor máximo, valor mínimo, varianza y desviación estándar, entre otros). Por ejemplo, si se está midiendo el tiempo de respuesta, podría decirse que como máximo el servicio responderá en 1000 ms, es decir, el atributo statisticalQualifier será “valor máximo”.

Este elemento extiende la metaclase *Feature* de UML.

2.6.1.3. QoS Category

Una *QoS Category* agrupa características de calidad, permitiendo la categorización de éstas. En [17] se define un conjunto de categorías QoS, entre las que se encuentra Desempeño, la cual agrupa las características Latencia y Rendimiento. Este elemento extiende la metaclase *Package* de UML. En la Figura 16 se puede ver la relación entre los elementos QoS Category, QoS Characteristic y QoS Dimension dentro del paquete QoS Characteristics.

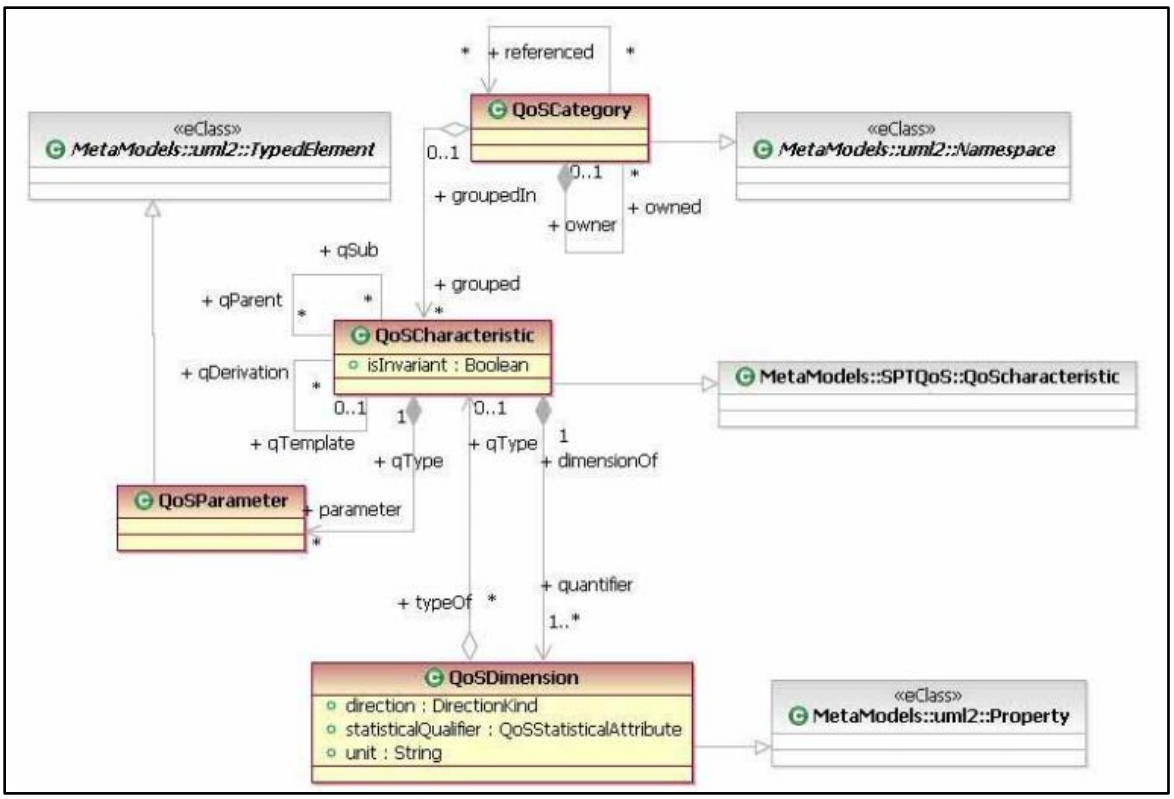

Figura 16. Diagrama de QoS Characteristics de [17]

2.6.1.4. QoS Value

Una *QoS Value* es una instancia de una QoS Characteristic y define los valores específicos para cada medida modelada con una QoS Dimension. Cuando se asocia un QoS Value a un elemento del modelo, se está caracterizando a ese elemento con los valores de calidad definidos. Por ejemplo, si se quiere modelar el tiempo de respuesta de un servicio, se le podrá asociar un QoS Value que instancie la característica Tiempo de Respuesta y así definir el tiempo máximo que el servicio podrá satisfacer. Este elemento extiende la metaclase *InstanceSpecification* de UML. En la Figura 17 se muestra el elemento QoS Value y sus relaciones con los demás elementos dentro del paquete QoS Values.

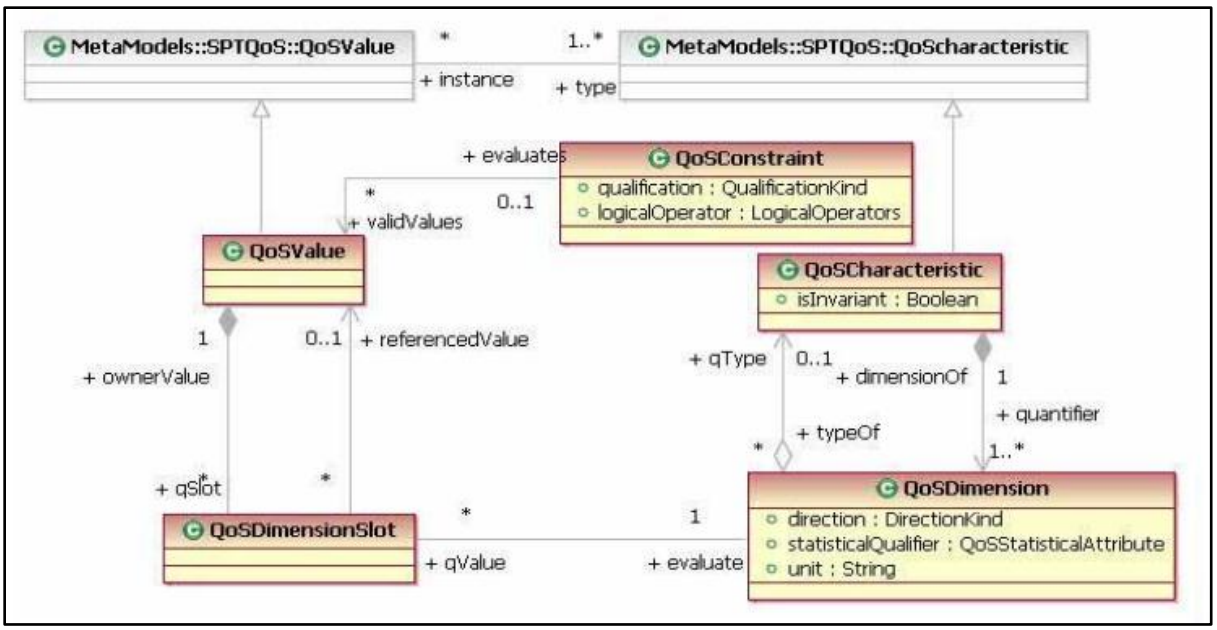

Figura 17. Diagrama de QoS Values [17]

2.6.1.5. QoS Constraint

Una *QoS Constraint* permite caracterizar elementos del modelo con aspectos no funcionales. En [17] se especifican dos formas para modelar restricciones de calidad en los elementos del modelo: como una *Constraint* UML en la que se especifica una expresión OCL, o como una *Dependency* que asocia un elemento QoS Value con otro elemento del modelo. Debido a estas formas distintas de agregar restricciones de calidad, este elemento extiende las metaclases *Dependency* y *Constraint* de UML. El elemento QoS Constraint es un estereotipo abstracto, del cual existen tres especializaciones definidas como dependencias entre elementos QoS Value y otros elementos del modelo:

- ***QoS Offered***: Se utiliza para modelar las características de calidad ofrecidas. Cuando el proveedor define una dependencia QoS Offered entre uno de los servicios que provee y un valor de calidad, es él quien debe satisfacer dicha característica, y cuando es el consumidor quien la define, él debe satisfacerla al invocar el servicio.
- ***QoS Required***: Se utiliza para modelar las características de calidad requeridas. Cuando el proveedor define una dependencia QoS Required entre uno de los servicios que provee y un valor de calidad, el consumidor debe cumplir con dicha característica para garantizar la calidad ofrecida por el proveedor. Por ejemplo, el proveedor podría requerir una frecuencia máxima de invocaciones al servicio para satisfacer el tiempo de respuesta ofrecido. Cuando el consumidor define una dependencia QoS Required, es el proveedor del servicio el que deberá satisfacer los requerimientos de calidad del consumidor.
- ***QoS Contract***: Se utiliza para modelar las características de calidad acordadas entre un proveedor y un consumidor en un contrato de servicios. Los consumidores de servicios pueden tener diferentes requerimientos de calidad de servicio, por ejemplo distintos anchos de banda de bajada de datos. Por otro lado, los proveedores de servicios normalmente ofrecen distintos niveles de calidad (suscripciones preferenciales o premium) [25]. Antes de que un consumidor

se suscriba al proveedor, necesita establecer un contrato como un acuerdo mutuo con el proveedor, detallando los niveles de garantía de varias características QoS. Una vez creado el contrato, ambas partes deben atenerse a éste. Por ejemplo, el consumidor no debe excederse en la cantidad de solicitudes, y el proveedor debe cumplir con los niveles de performance acordados [25].

En la Figura 18 se pueden ver los elementos mencionados en esta sección.

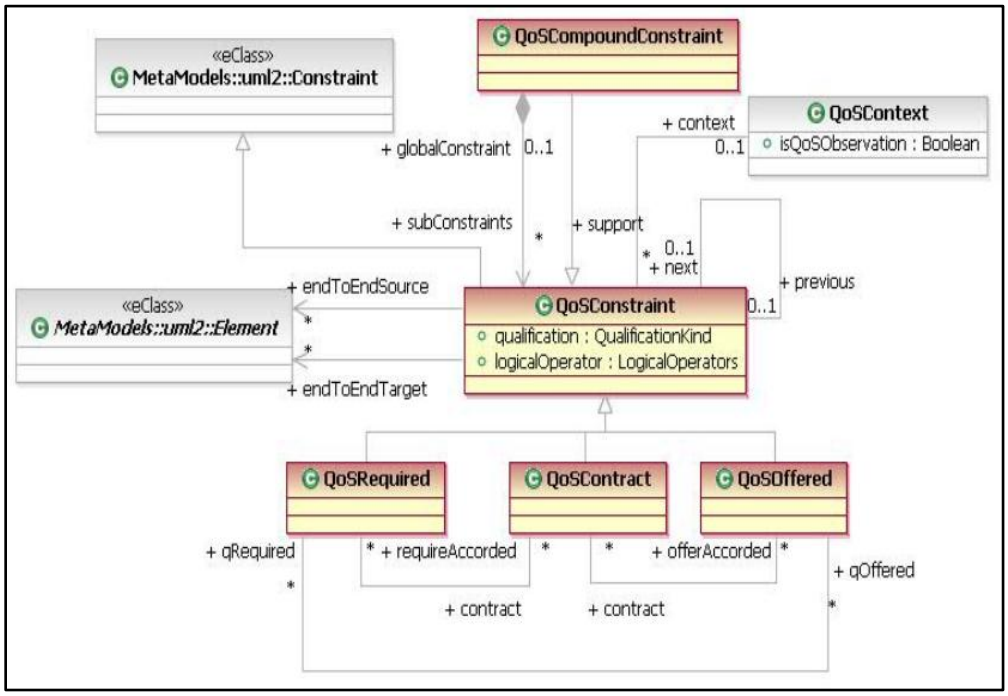

Figura 18. Diagrama de QoS Constraints de [17]

2.6.2. Diagramas de QoS

A continuación se explicarán los distintos diagramas que se pueden definir en QoS según [17].

2.6.2.1. Diagrama de Categorías

El Diagrama de Categorías permite definir dependencias entre categorías. Para esto, se pueden definir elementos QoS Category que podrán ser agrupados dentro de otros elementos QoS Category y también podrán incluirse unos a otros. Esto se puede ver en la Figura 19.

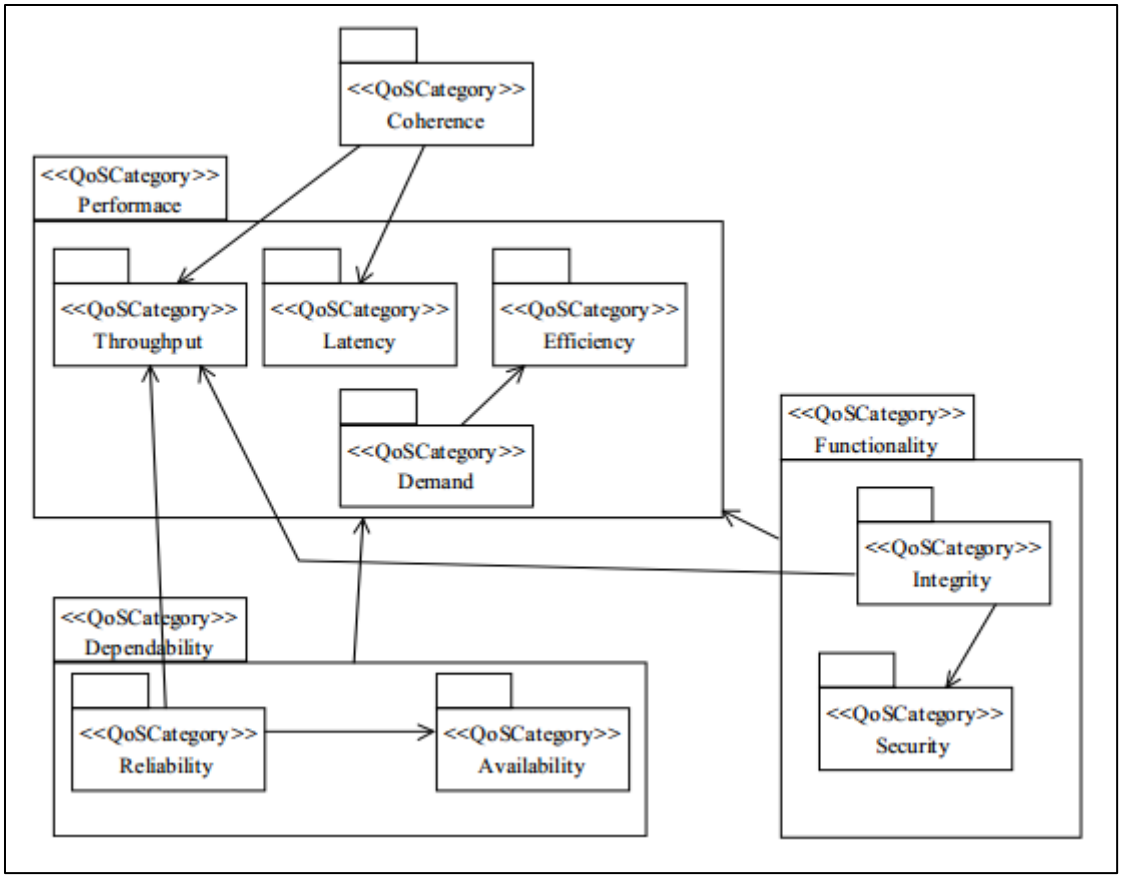

Figura 19. Ejemplo de Diagrama de Categorías de [17]

2.6.2.2. Diagrama de Características de Calidad

El Diagrama de Características de Calidad permite definir una taxonomía de características de calidad, especificando las dimensiones de cada característica y definiendo las relaciones entre éstas. En la Figura 20 se puede ver un ejemplo de un diagrama de este tipo.

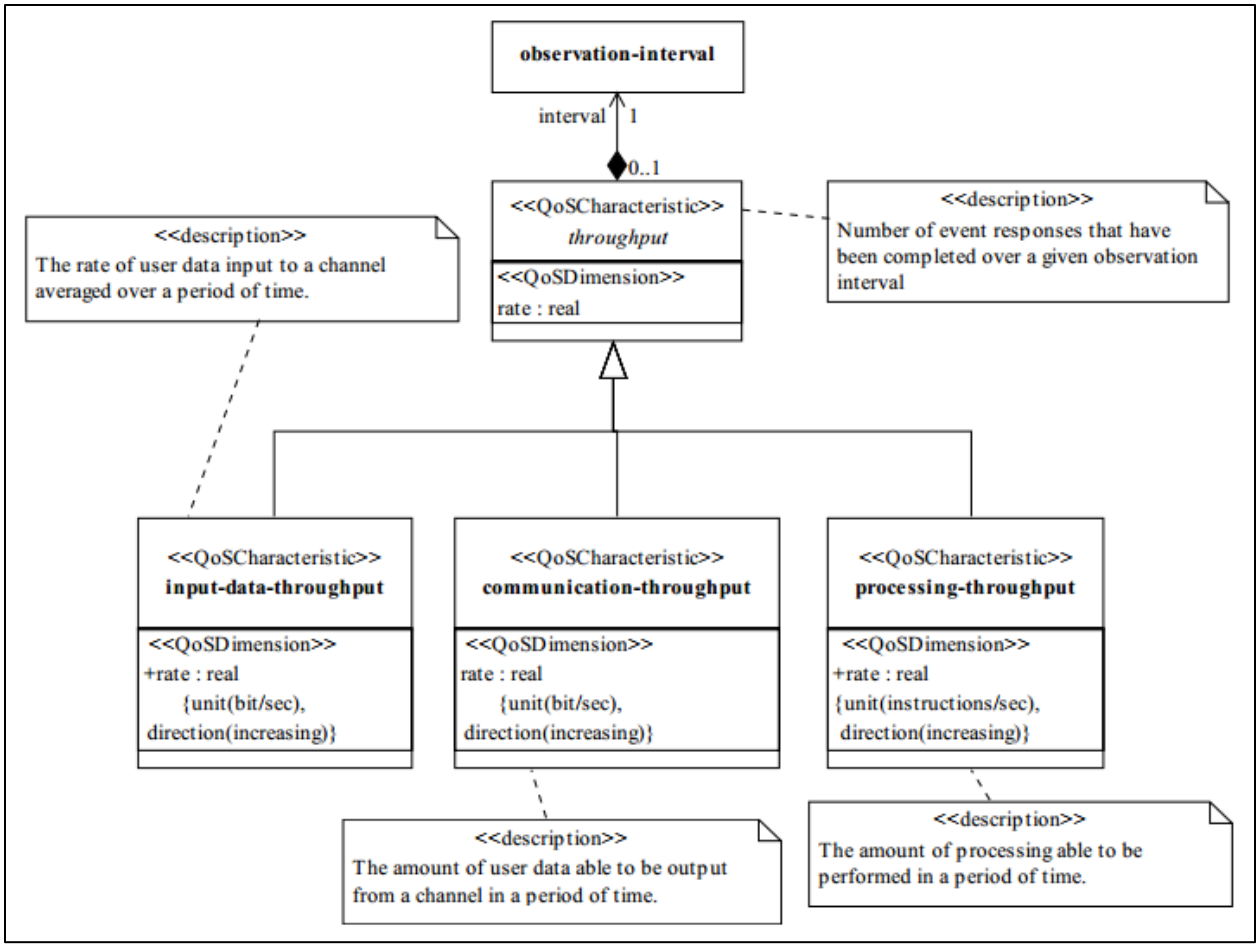

Figura 20. Ejemplo de Diagrama de Características de Calidad de [17]

2.6.2.3. Diagrama de Restricciones de Calidad

Los Diagramas de Restricciones de Calidad son diagramas UML de cualquier tipo, en el cual se aplican restricciones de calidad a los elementos del modelo. En estos diagramas se modelan los aspectos no funcionales del sistema, especificando los valores de las características de calidad que son ofrecidos, requeridos y negociados por los participantes. En la Figura 21 se puede ver el extracto de un diagrama de restricciones de calidad de [17].

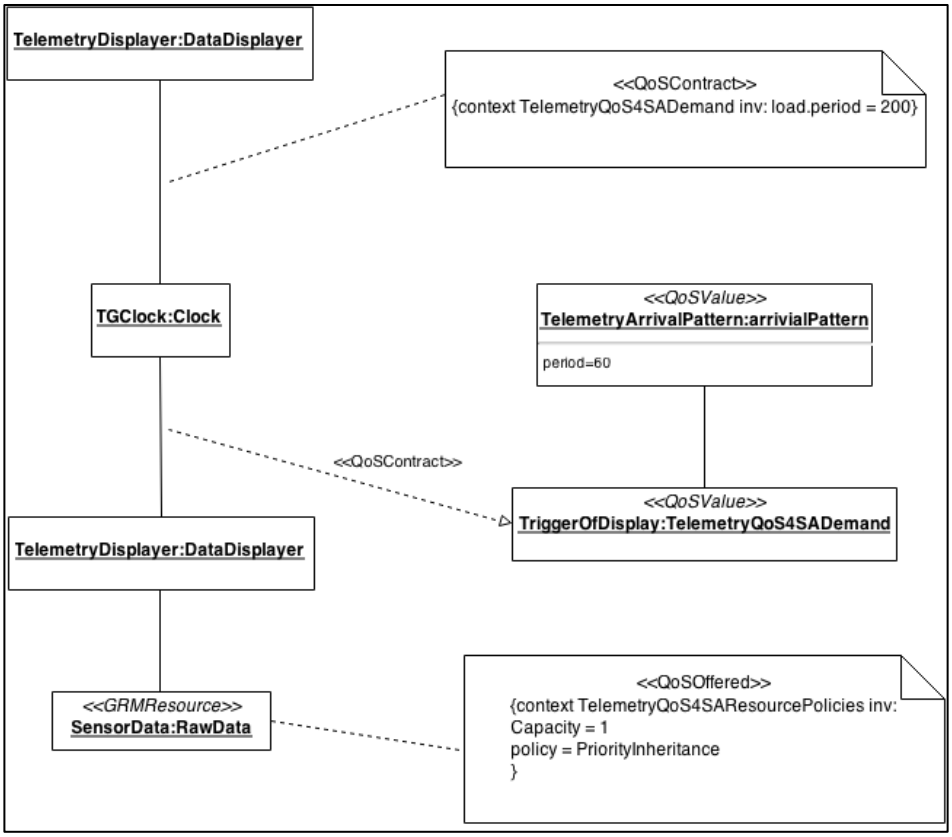

Figura 21. Ejemplo de Diagrama de Restricciones de Calidad de [17]

2.7.Modelado de SoaML con QoS

La especificación QoS [17] define un conjunto de elementos que pueden ser utilizados para modelar las características de calidad de un sistema, pero no especifica cómo incluir estas características en modelos de dominios específicos, como lo son los modelos SoaML. En [18], Schot investiga distintas opciones para modelar características de calidad en arquitecturas SOA en dos niveles de modelos: PIM y PSM.

Para los modelos PIM, utiliza la especificación SoaML [16] para el modelado de servicios y el estándar QoS para el modelado de características de calidad. Para modelar las características de calidad que ofrece un servicio, decide asociar elementos *QoSValue* al puerto de tipo *Service* utilizando una dependencia *QoSOffered*. De forma análoga, se asocian elementos *QoSValue* al puerto de tipo *Request* mediante dependencias *QoSRequired* para modelar que un consumidor espera ciertos niveles de calidad del servicio que va a utilizar. Para modelar las características de calidad acordadas en un contrato entre el proveedor y consumidor de un servicio, se asocian elementos *QoSValue* a elementos *ServiceContract* utilizando dependencias *QoSContract*. En la Figura 22 se puede ver un ejemplo de estas asociaciones.

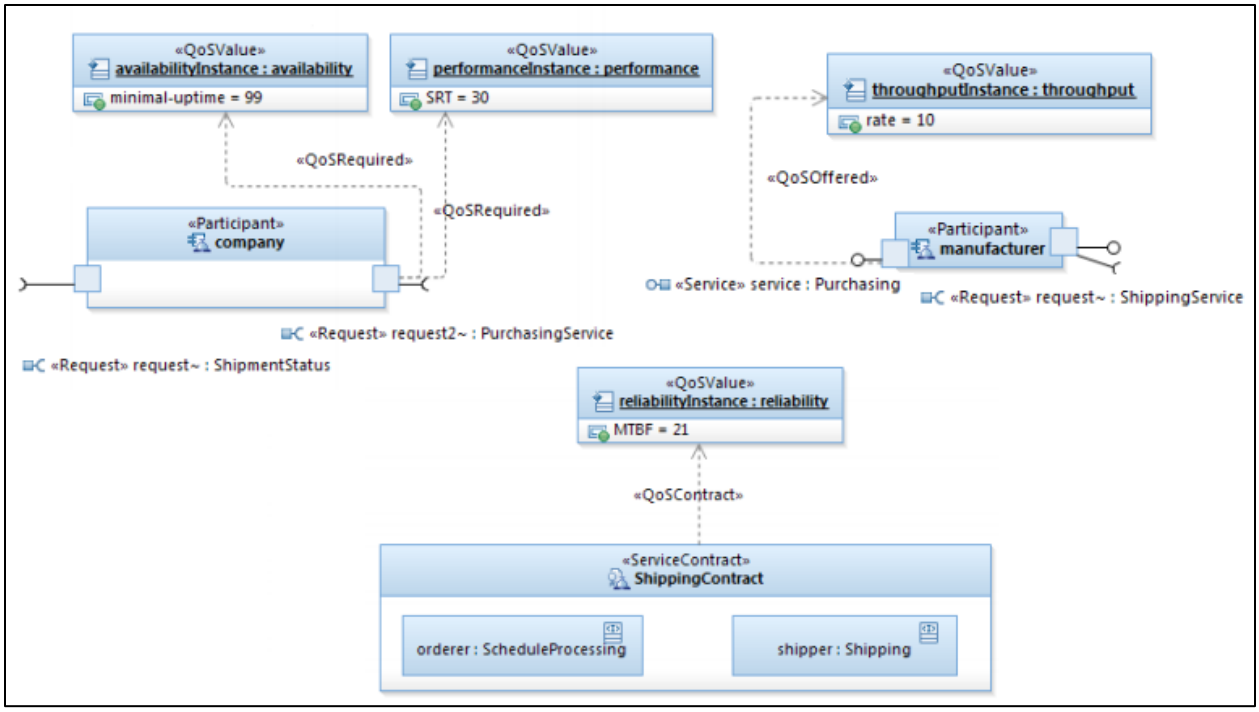

Figura 22. Ejemplo de características QoS aplicadas a modelos SoaML de [18]

A nivel de PSM, Schot [18] decide mapear las características QoS en modelos SoaML de dos formas distintas:

- Mapear los elementos QoSValue asociados a puertos de servicio y de pedido a políticas expresadas con el estándar WS-Policy [26], definiendo una gramática simple para modelar las aserciones de las políticas.
- Mapear los elementos QoSValue asociados a contratos de servicio a documentos WS-Agreement [27].

En la sección 2.8.2.4 se describen en detalle estos estándares. Los dos tipos de mapeos se describen en [18], indicando qué elementos del modelo origen deben ser transformados a los elementos del modelo destino. Los documentos WS-Agreement que se generan no están completos, ya que los modelos SoaML con características QoS no contienen toda la información que se requiere en el acuerdo.

2.8.Herramientas y tecnologías asociadas

En esta sección presentamos las herramientas empleadas para la construcción de plugins y los desarrollos de grupos anteriores sobre los que se basa este proyecto. También se presentan las tecnologías Web Services que son relevantes para el proyecto.

2.8.1. Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma que es utilizado para distintas tareas, principalmente para desarrollar software. La arquitectura de Eclipse es un micro-núcleo que emplea módulos para proporcionar toda su funcionalidad, y permite añadir nuevas funcionalidades fácilmente [28]. Es una arquitectura basada en plugins. Un plugin es una unidad mínima de funcionalidad que permite escribir nuevas extensiones incrementando de este modo las funcionalidades

provistas por el ambiente. Para esto, cada plugin Eclipse puede ofrecer puntos de extensión que permiten la posibilidad de agregar nuevas funcionalidades. En la Figura 23 se presenta un esquema de la arquitectura descripta.

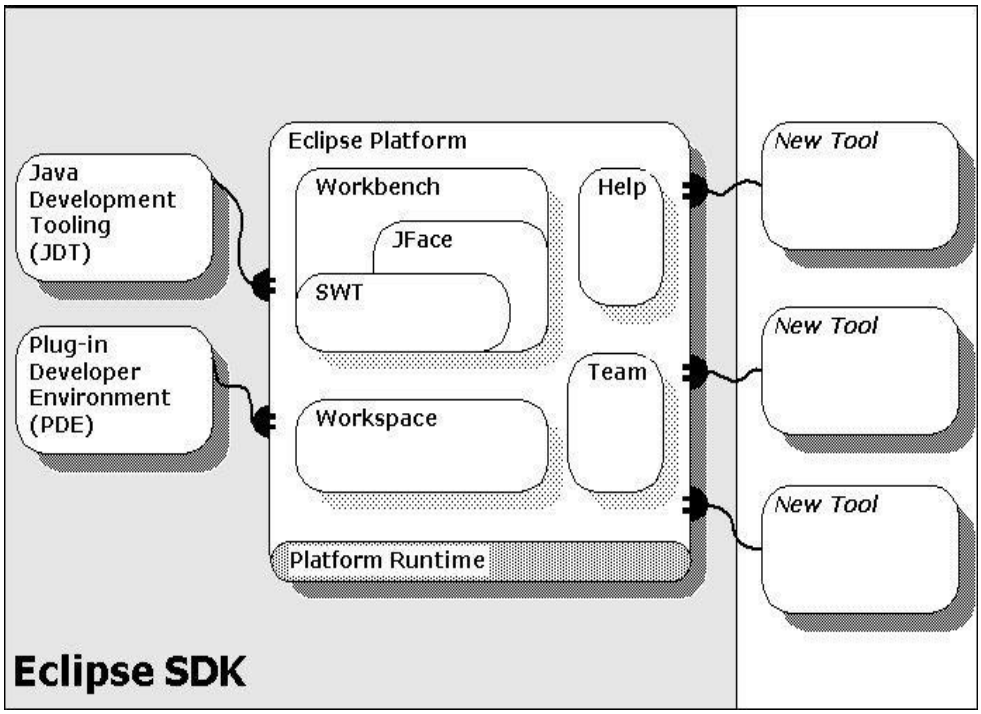

Figura 23. Arquitectura de Eclipse de [28]

2.8.2. Tecnología Web Services

Un servicio web es un sistema de software diseñado para apoyar la interoperabilidad entre diferentes aplicaciones, ejecutado en una variedad de plataformas, el cual tiene una interfaz descrita en un formato procesable por máquina (WSDL). Otros sistemas interactúan con el servicio web utilizando mensajes SOAP, los cuales son generalmente transportados utilizando HTTP con una serialización XML [29]. Existe también otro enfoque para servicios web, el enfoque REST, que es una técnica de arquitectura de software para sistemas distribuidos y es utilizada para describir cualquier interfaz web que utilice XML y HTTP [30].

La arquitectura tradicional de servicios web implica varias tecnologías, algunas de las cuales se pueden ver en la Figura 24.

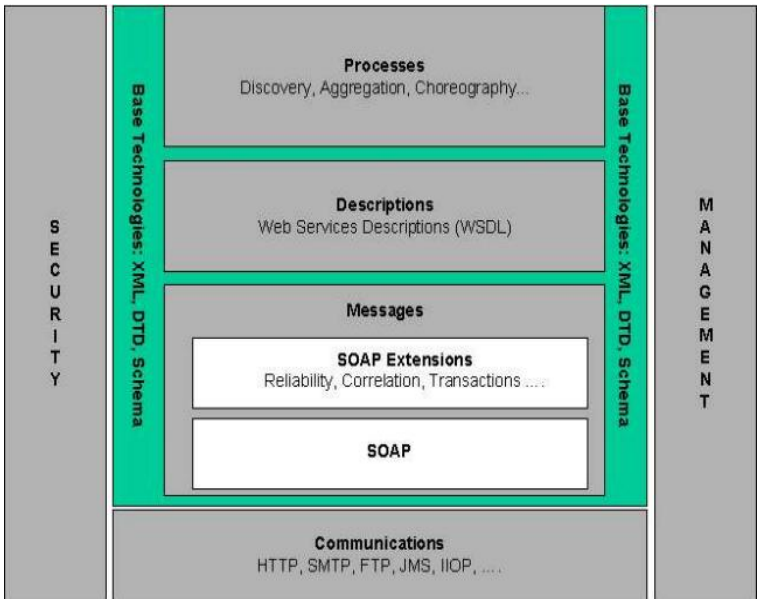

Figura 24. Arquitectura de Web Services de [29]

A continuación se describen dos de los componentes más importantes referentes a un servicio web. Por mayor información referirse al Anexo Documento de Estado del Arte.

2.8.2.1. SOAP

Protocolo Simple de Acceso a Objetos, SOAP por su sigla en inglés, es un protocolo basado en XML que permite la interacción de múltiples objetos en diferentes procesos y la transmisión de información compleja. Los datos pueden ser transmitidos a través de distintas tecnologías mostradas en la capa Communications de la Figura 24 (HTTP, SMTP, FTP, etc.). SOAP especifica el formato de los mensajes, los cuales están compuestos por un *envelope* (sobre), cuya estructura está formada por los elementos *header* (cabecera) y *body* (cuerpo) [31].

2.8.2.2. WSDL

WSDL (Web Services Description Language) es una especificación estándar basada en XML que se utiliza para describir servicios web. En éste se definen los formatos del mensaje, tipos de datos, protocolos de transporte y formatos de serialización de transporte que deben ser utilizados entre el consumidor y el proveedor. También especifica las ubicaciones de red en las que puede ser invocado, y puede proporcionar alguna información sobre el patrón de intercambio de mensajes que se espera [28].

2.8.2.3. Implementación de WS

Para la implementación de servicios web existen dos enfoques, *Top Down* y *Bottom Up*. La primera consiste en crear el servicio web a partir de un archivo WSDL. En cambio, en la segunda, se comienza por el código y se genera a partir de éste el WSDL correspondiente [32]. Una comparación de estos dos enfoques se puede ver en la Tabla 2.

Tabla 2. Ventajas y desventajas de los enfoques Top Down y Bottom Up de [32]

	Top Down	Bottom Up
Ventajas	1. Permite el desarrollo independiente y paralelo entre cliente y servicio. 2. Cuando se crean nuevos tipos, éstos pueden ser reutilizados por otros servicios. 3. No se generan dependencias con la implementación.	1. Forma rápida de exponer funcionalidades vía WS. 2. No requiere conocimiento de XMLSchema, WSDL.
Desventajas	1. Requiere conocimiento de XMLSchema y WSDL.	1. El esquema es embebido en el WSDL. 2. Los cambios en la interfaz son más difíciles de manejar. 3. En el WSDL hay dependencias con la implementación (paquetes y namespaces).

2.8.2.4. Especificación de calidad de servicios para WS

En esta sección presentaremos algunos estándares que forman parte de la familia de especificaciones de tecnologías basadas en servicios web.

2.8.2.4.1. WS-Policy

El estándar WS-Policy [26], publicado por W3C, es utilizado para expresar políticas de Web services basándose en sus propiedades no funcionales. Éste define un framework y un modelo para expresar requerimientos y características generales de entidades en un sistema basado en servicios web, en formato XML.

El modelo está compuesto por tres elementos principales: aserción de política (*Policy Assertion*), alternativa de política (*Policy Alternative*) y política (*Policy*). Una aserción de política representa un requerimiento, una capacidad o una propiedad. Por ejemplo, una aserción puede requerir que se utilice cierto tipo de encriptado en los mensajes, o puede describir ciertas características de calidad del servicio expuesto. Una alternativa de política contiene una lista no ordenada de aserciones de políticas. Una política es una lista no ordenada de alternativas de políticas, la cual es aplicada a una entidad que puede ser un mensaje, recurso, operación o *endpoint*. Cuando se aplican a un sistema basado en servicios web, las políticas son utilizadas para expresar condiciones en la interacción entre proveedores y consumidores. Por ejemplo, un consumidor puede decidir si utilizar un servicio o no dependiendo de las políticas expuestas por el proveedor [26].

Una expresión de políticas es la representación en XML de una política, la cual puede ser normal o compacta. En la forma normal se enumeran todas las alternativas y todas las aserciones dentro de éstas; en la forma compacta se utiliza la menor cantidad de expresiones posibles [26]. En la Figura 25 se muestra un ejemplo de una expresión en forma normal de una política. Esta política contiene dos alternativas: si la primera alternativa es seleccionada, el cuerpo del mensaje tiene que ser firmado; si la segunda alternativa es seleccionada, el cuerpo del mensaje tiene que ser encriptado.

```
<wsp:Policy
   xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
   xmlns:wsp="http://www.w3.org/ns/ws-policy" >
  <wsp:ExactlyOne>
    <wsp:All>
       <sp:SignedParts>
          <sp:Body/>
       </sp:SignedParts>
    </wsp:All>
    <wsp:All>
       <sp:EncryptedParts>
          <sp:Body/>
       </sp:EncryptedParts>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Figura 25. Ejemplo de una expresión de política [26]

2.8.2.4.2. WS-Agreement

La especificación WS-Agreement [27] define un lenguaje y un protocolo para publicar las capacidades de los proveedores de servicios, crear acuerdos y monitorear el cumplimiento de los acuerdos en tiempo de ejecución.

Un acuerdo entre un proveedor y un consumidor de servicios especifica los requerimientos del consumidor y las garantías de disponibilidad de recursos y/o calidad de servicio del proveedor. Por ejemplo, un acuerdo puede proveer garantías sobre el tiempo máximo de respuesta y de la disponibilidad del servicio, así como también, garantías en la disponibilidad de recursos mínimos tales como memoria, CPU, almacenamiento, etc. [27]. Como se puede ver en la Figura 26, un acuerdo está compuesto por varias partes y es expresado en formato XML.

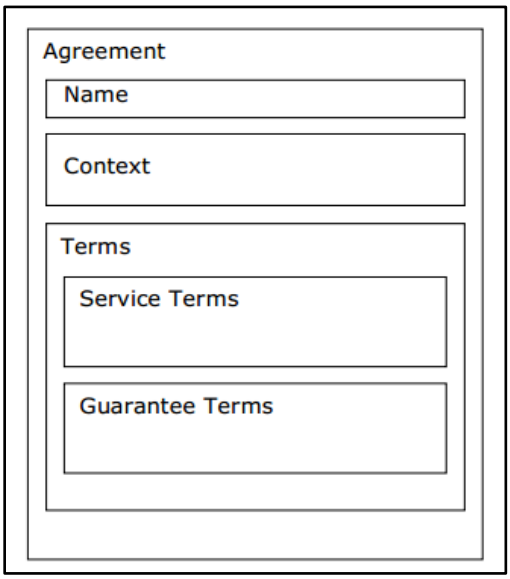

Figura 26. Estructura de un acuerdo

La etiqueta *Agreement* encapsula enteramente el acuerdo y su atributo *AgreementId* es un identificador obligatorio, que ayuda a identificar la versión del acuerdo vigente. La etiqueta *Name* dentro de *Agreement* es un nombre opcional del acuerdo.

En la etiqueta *Context* se define el alcance del acuerdo, el cual incluye las partes involucradas. Adicionalmente, el contexto contiene metadata sobre el acuerdo, como la duración del acuerdo.

Un término expresa derechos u obligaciones definidas para un participante. Cada término en un acuerdo tiene un tipo, que puede ser “término de servicio” o “término de garantía”. Los términos de servicio proveen la información necesaria para instanciar o identificar a un servicio perteneciente al acuerdo y al cual aplican los términos de garantía. Los términos de garantía especifican los niveles de servicios acordados por los participantes. Los sistemas de gestión pueden utilizar los términos de garantía para monitorear el servicio y hacer cumplir el acuerdo.

Los términos de descripción de servicio (*ServiceDescriptionTerm* o SDT) son un componente fundamental de un acuerdo. La provisión de este servicio puede estar condicionada a restricciones en tiempo de ejecución y objetivos de niveles de servicio adicionales sobre cómo debe funcionar el servicio; los SDTs definen la funcionalidad que será distribuida bajo un acuerdo.

El elemento ServiceProperties es utilizado para definir propiedades expuestas y medibles asociadas al servicio, como por ejemplo tiempo de respuesta y rendimiento. Estas propiedades son utilizadas para expresar los objetivos de niveles de servicio.

Los términos de garantía (*GuaranteeTerm*) definen la garantía sobre la calidad de servicio. Un acuerdo puede garantizar límites en la disponibilidad de recursos tales como memoria, CPU o almacenamiento. Una expresión de garantía también puede incluir condiciones en factores externos, tales como la hora del día u otras condiciones que el consumidor del servicio debe cumplir. Por ejemplo, el tiempo promedio del tiempo de respuesta del servicio de un banco puede ser garantizado si la tasa de pedidos está dentro de un umbral especificado durante días de semana.

2.8.2.4.3. WS-Security

La especificación WS-Security [33] propone un conjunto de extensiones SOAP que pueden ser utilizadas para asegurar la integridad y confidencialidad de los mensajes.

La especificación provee tres mecanismos principales: tokens de seguridad como parte del mensaje, integridad de mensajes y confidencialidad de mensajes. Un token de seguridad es un conjunto de declaraciones (*Claims*) hechas por una entidad, como por ejemplo nombre, identidad, clave, privilegios, etc. Las firmas son utilizadas para verificar el origen y la integridad de los mensajes, asegurando que cualquier modificación en los mensajes pueda ser detectada. Para mantener la confidencialidad de los mensajes SOAP se utilizan mecanismos de encriptación.

El formato y semántica de los tokens no están definidos en esta especificación, sino que son definidos en documentos de perfiles asociados. Por ejemplo, en el perfil Username Token [34] se especifica el formato de tokens utilizados para enviar un usuario y contraseña en los mensajes SOAP. En la Figura 27 se muestra un ejemplo de un mensaje SOAP con un token que especifica un nombre de usuario y contraseña.

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="...">
   <S11:Header>
      ...
      <wsse:Security>
         <wsse:UsernameToken>
            <wsse:Username>Zoe</wsse:Username>
            <wsse:Password>IloveDogs</wsse:Password>
         </wsse:UsernameToken>
      </wsse:Security>
      ...
   </S11:Header>
   ...
</S11:Envelope>
```

Figura 27. Ejemplo de UsernameToken

La especificación WS-SecurityPolicy [35] define un conjunto de aserciones de políticas de seguridad basadas en las características de seguridad de WS-Security y expresadas con el framework WS-Policy. La especificación define distintos tipos de aserciones: de integridad, de confidencialidad, de elementos requeridos, y de tokens de seguridad (por ejemplo Kerberos y X.509). En la Figura 28 se muestra un ejemplo de una política en la que se requiere que el emisor del mensaje envie la contraseña en un UsernameToken.

```
<wsp:Policy>
    <sp:SupportingTokens>
        <wsp:Policy>
            <sp:UsernameToken sp:IncludeToken=
            "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient"
        </wsp:Policy>
    </sp:SupportingTokens>
</wsp:Policy>
```

Figura 28. Ejemplo de aserción de UsernameToken

2.8.2.4.4. Q-WSDL

Q-WSDL es una extensión liviana de WSDL propuesta en [36] para permitir la descripción de características QoS de un servicio web, tales como rendimiento, confiabilidad, disponibilidad, seguridad, etc. En la especificación se introduce el metamodelo WSDL que es derivado del esquema XML de WSDL, al que se le agregan las características QoS y se lo llama Q-WSDL. En el documento Anexo Estado del Arte se describe más en detalle esta extensión.

2.8.3. Plataforma Java EE

La plataforma Java EE es una colección de especificaciones que definen una infraestructura con componentes estandarizados y servicios para desarrollar aplicaciones distribuidas, en una arquitectura multicapa.

Java EE tiene como objetivo principal permitir que el desarrollador se centre en el diseño e implementación del sistema, delegando a la infraestructura del servidor de aplicaciones Java EE las cuestiones de más bajo nivel ajenas a la aplicación. Para nuestro proyecto utilizamos la versión Java EE 7.

Arquitectura JEE

La plataforma Java EE utiliza un modelo de aplicación distribuida de varios niveles. La lógica de aplicación se divide en distintos componentes según su función y los componentes de la aplicación que constituyen una aplicación Java EE son instalados en diferentes máquinas dependiendo del nivel del ambiente JEE al que pertenecen.

La Figura 29 muestra dos aplicaciones multinivel Java EE divididas en los niveles que se describen a continuación:

- Los componentes a nivel de cliente (*Client-tier*) se ejecutan en la máquina cliente.
- Los componentes de nivel Web (*Web-tier*) se ejecutan en el servidor Java EE.
- Los componentes de nivel de negocio (*Business-tier*) se ejecutan en el servidor Java EE.
- El sistema de información de la empresa (EIS, por su sigla en inglés) se ejecuta en el servidor EIS.

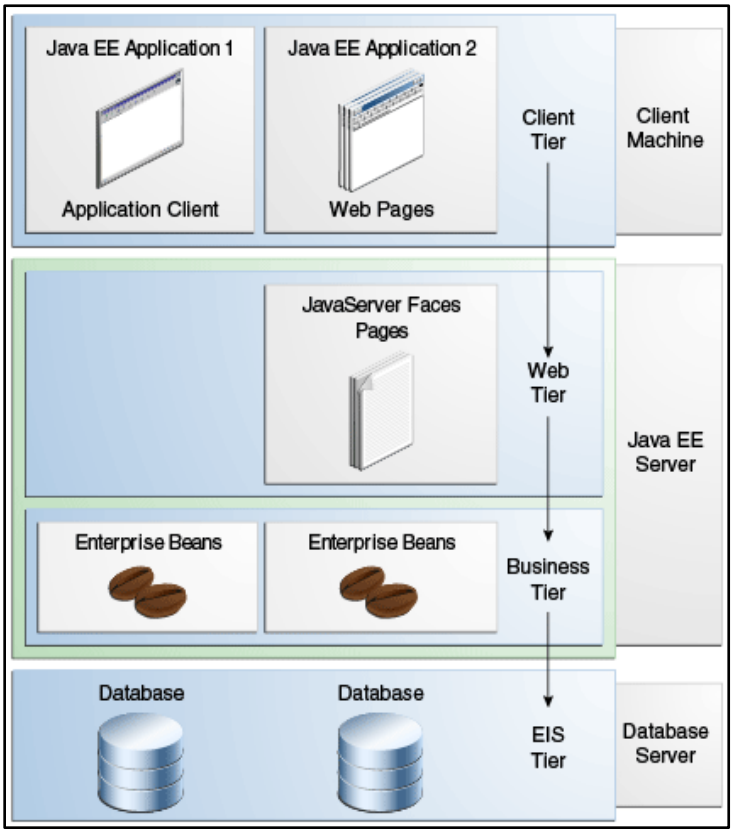

Figura 29. Aplicaciones multinivel de [37]

Una aplicación Java EE consiste de las capas de software mostradas en la Figura 29, y es considerada generalmente como aplicación de tres niveles debido a que se distribuyen en tres lugares: las máquinas cliente, la máquina servidor Java EE y la base de datos o máquinas existentes en el extremo posterior.

2.8.4. Desarrollos previos

En esta sección se describen los plugins de Eclipse realizados por proyectos de grado que trabajaron en SoaML y generación de código en años anteriores, y que utilizamos en este proyecto como base para el desarrollo del plugin integrado SoaML Toolkit.

2.8.4.1. Plugin SoaML para Eclipse

El plugin de Eclipse SoaML, realizado en el marco de un proyecto de grado en el año 2010, es un plugin que implementa el estándar SoaML (la versión beta 2 de la especificación) y permite la generación de diagramas de forma gráfica y la importación y exportación de los modelos generados en formato XMI para permitir la interoperabilidad con otras herramientas.

Actualmente existen muy pocas herramientas que ofrezcan el modelado de servicios con el estándar SoaML [16], la mayoría de las cuales son comerciales por lo que este plugin al encontrarse en el contexto de Eclipse y del proyecto Papyrus, provee a la comunidad una herramienta para el modelado de servicios con SoaML como soporte para desarrollos orientados a servicios, lo cual fue de gran aporte.

La solución que se implementó para la realización de este plugin extiende las funcionalidades del plugin Papyrus para el modelado UML, permitiendo de esta forma reutilizar tanto los diagramas que se implementaron para el mismo, como otras funcionalidades provistas a nivel de interfaz gráfica. Se aprovecharon también ciertas facilidades que Papyrus ofrecía para la definición de editores de perfiles de UML y mecanismos de personalización de los mismos. El plugin SoaML provee la mayoría de los diagramas definidos por el estándar SoaML para el modelado de: Arquitectura de Servicios, Participantes, Servicios y su especificación mediante contratos de servicio, interfaces, operaciones con parámetros de entrada y salida. Puntualmente brinda siete diagramas: diagrama de Arquitectura de Servicios, diagrama de Contratos de Servicios, diagrama de Participantes como clases y como componentes, diagrama de Mensajes y diagrama de Capacidades. En la Figura 30 se puede ver el esquema de la arquitectura Eclipse con las extensiones para el plugin SoaML.

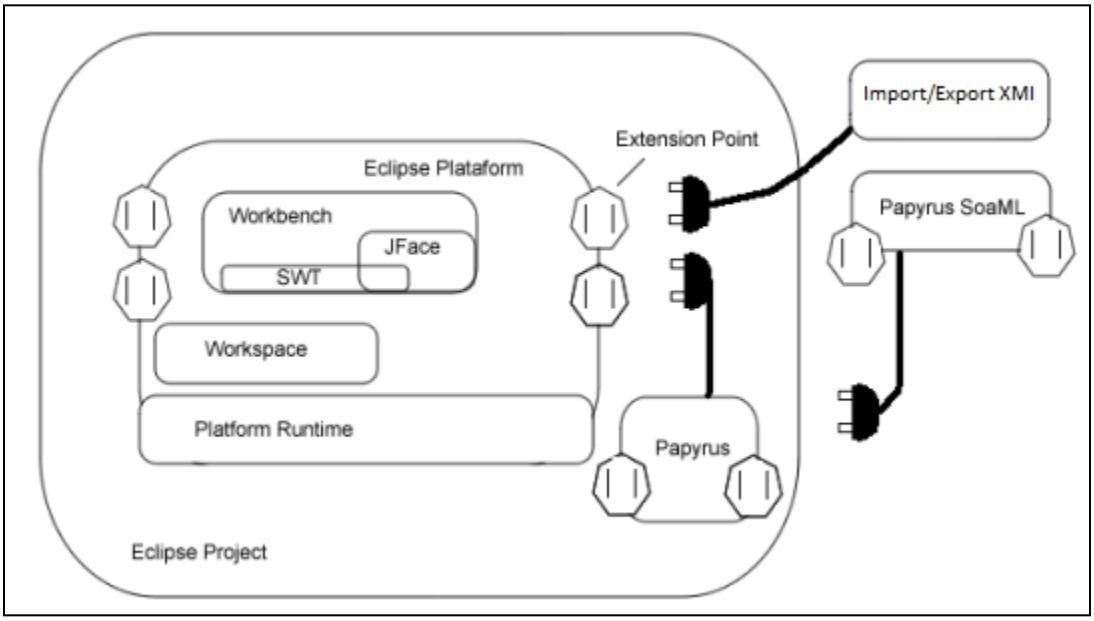

Figura 30. Esquema de la arquitectura Eclipse con las extensiones para el plugin SoaML

2.8.4.2. *Plugin SoaML2Code para Eclipse*

El plugin de Eclipse denominado SoaML2Code, que fue desarrollado en un proyecto de grado en el año 2012, es una extensión del plugin SoaML. Éste permite que a partir de un modelo SoaML representado por los diagramas realizados en el plugin SoaML, genere código ejecutable en lenguaje Java que represente dicho modelo, y exponga los servicios provistos en el modelo como Web Services.

La solución implementada utiliza Apache CXF, el cual es un framework para servicios de código abierto que ayuda a construir y desarrollar servicios, incluyendo entre sus principales características el soporte a múltiples estándares de Web Services, además de soporte para una variedad de lenguajes de programación en los clientes y una gran variedad de protocolos de transporte.

El plugin toma en cuenta para la generación de código la mayoría de los elementos que forman parte del estándar SoaML y que son representados por los diagramas definidos por el plugin SoaML. Los servicios son expuestos como Web Services SOAP bajo la API de implementación Java para la creación de Web Services JAX- WS RI y JAX-WS + Spring, permitiendo a los usuarios de la comunidad agregar fácilmente alguna otra implementación que deseen.

La herramienta es de fácil utilización y al encontrarse en el contexto de Eclipse provee a la comunidad de un plugin para la generación de código ejecutable que permite la exposición de servicios a partir de modelos SoaML, funcionando correctamente de manera independiente de la versión de Eclipse que se esté utilizando y brindando una herramienta de distribución gratuita y de código abierto como soporte para desarrollos orientados a servicios. En la Figura 31 se puede observar cómo interactúa el plugin SoaML2Code con el plugin SoaML y Eclipse.

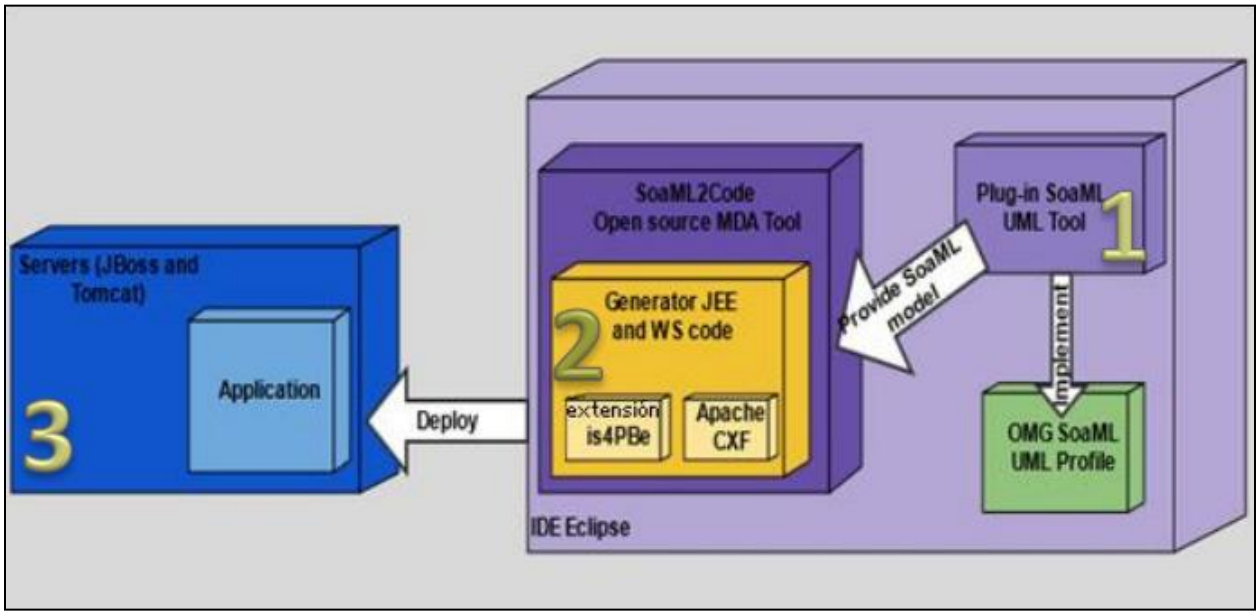

Figura 31. Interacción de componentes para la generación en plugin SoaML2Code

3. Requerimientos

En esta sección se describen los requerimientos planteados para la construcción de un plugin de modelado de características de calidad en modelos SoaML y la extensión del generador de código existente para incluir la generación de características de calidad.

3.1.Descripción resumida de los requerimientos

El plugin *SoaML Toolkit* debe proveer una solución integral para Eclipse que integre los plugins realizados y características QoS a modelos SoaML, así como la generación del código asociado. Para esto se identificaron varios casos de uso. En la Figura 32 se muestran los casos de uso de modelado SoaML para los que no fue necesario realizar ninguna modificación.

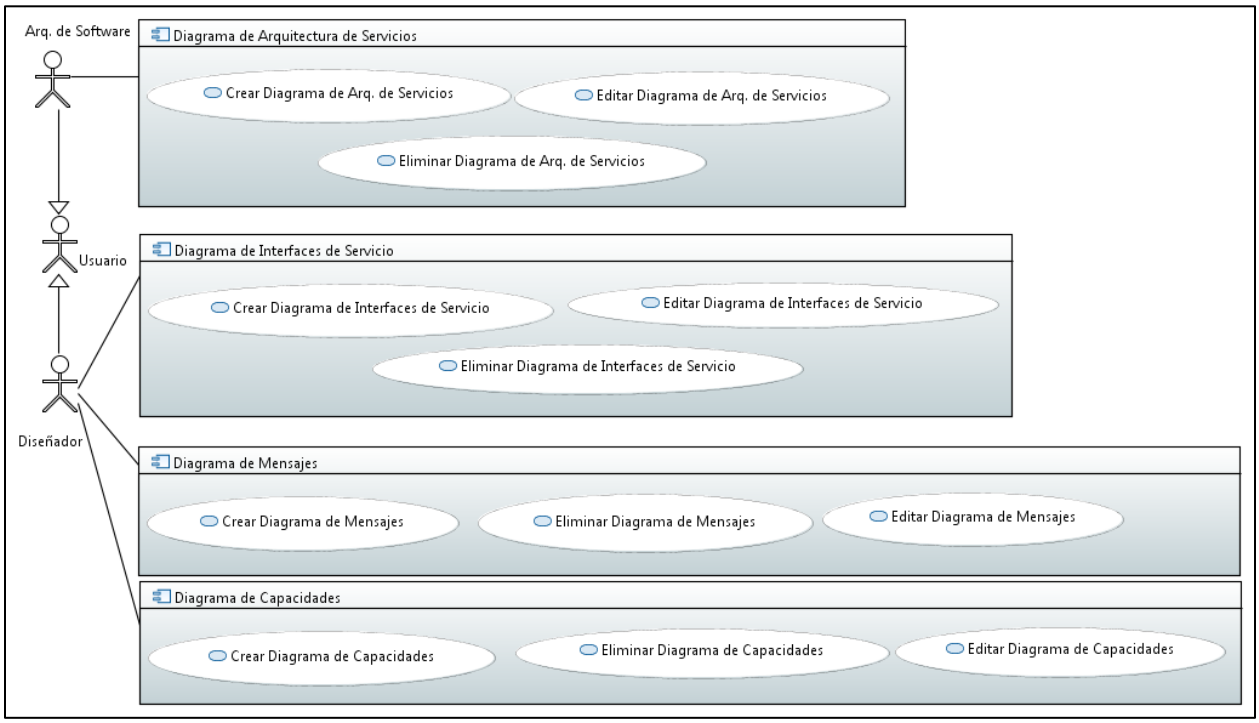

Figura 32. Casos de uso sin modificar

En la Figura 33 se ilustran los casos de uso de modelado SoaML y generación de código que requirieron modificaciones para agregar el modelado y generación de código de características de calidad.

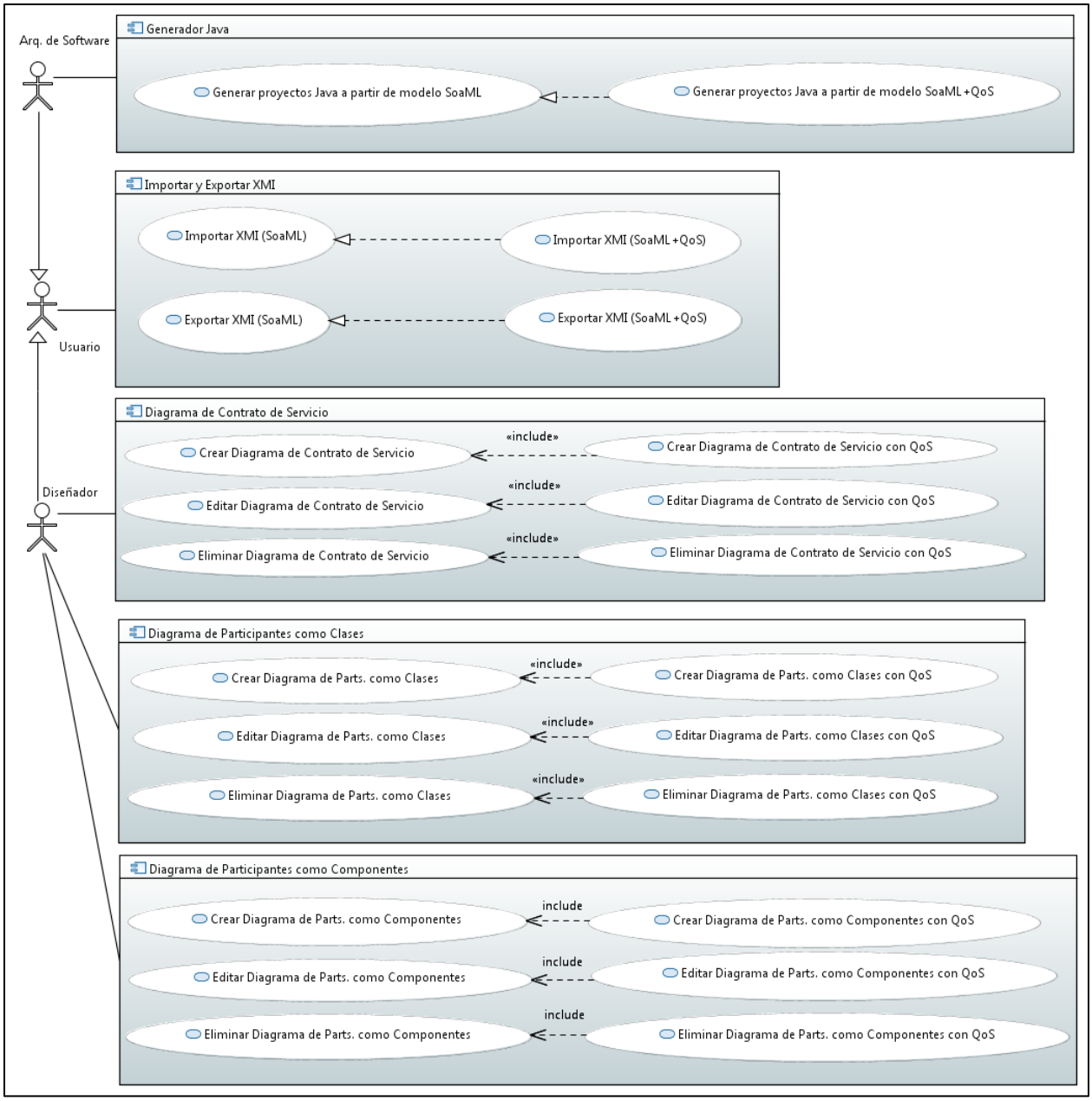

Figura 33. Casos de uso modificados

Finalmente, en la Figura 34 se presentan los casos de uso nuevos que fueron agregados para permitir la creación de modelos QoS, y la importación de modelos SoaML y QoS en modelos SoaML+QoS.

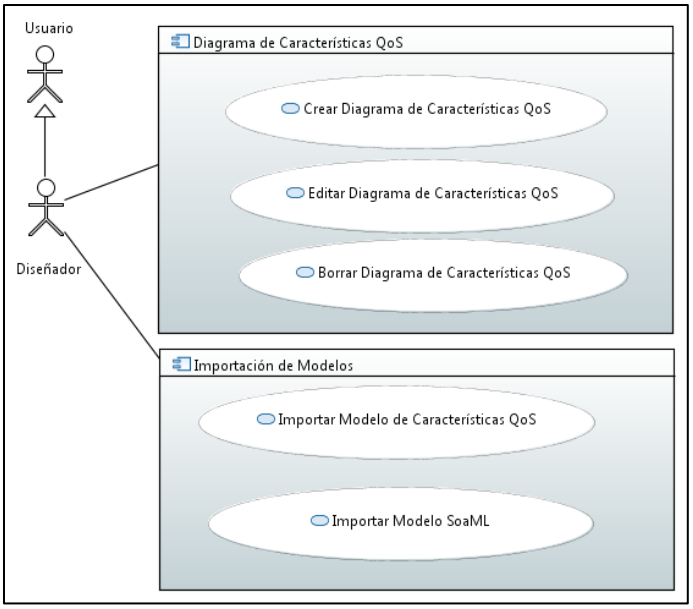

Figura 34. Casos de uso nuevos

Para describir brevemente los requerimientos, se agruparon los casos de uso identificados en la creación, modificación y eliminación de los diagramas y en la generación de proyectos Java. La descripción detallada de los casos de uso se puede ver en el Anexo Documento de Casos de Uso. A continuación se presenta la descipción general o de alto nivel para que se pueda entender qué hace cada uno:

3.1.1. Modelado

3.1.1.1. Crear diagrama de Características QoS

Permite al usuario crear un diagrama de tipo QoS Characteristic agregando elementos al lienzo desde la paleta de herramientas. Este diagrama permite definir las distintas características de calidad, las dimensiones que cuantifican dichas características así como también la categoría a la cual pertenece y las asociaciones que pueden haber entre distintas características. Se podrá agregar elementos tales como características, dimensiones y categorías así como también asociaciones y comentarios.

3.1.1.2. Editar diagrama de Características QoS

Permite al usuario editar un diagrama de tipo QoS Characteristic existente agregando, borrando y modificando elementos del diagrama.

3.1.1.3. Eliminar diagrama de Características QoS

Permite al usuario eliminar un diagrama preexistente de tipo *QoS Characteristic*, esto es eliminar cada uno de los elementos contenidos en el diagrama y las referencias existentes a ellos en otros diagramas.

3.1.1.4. Crear diagrama de Contrato de Servicios con QoS

Permite al usuario crear un diagrama de tipo *Service Contract with QoS*, agregando elementos al lienzo desde la paleta de herramientas del diagrama. Se pueden agregar elementos tales como contratos de servicios, roles y usos de contratos de servicios. El usuario también puede agregar características y valores de calidad de servicios, y asociaciones entre valores de calidad y contratos de servicios.

3.1.1.5. Editar diagrama de Contrato de Servicios con QoS

Permite al usuario editar un diagrama de tipo *Service Contract with QoS* existente agregando desde la paleta de herramientas del diagrama nuevos contratos de servicio al lienzo y agregar nuevos roles, usos de contratos de servicios y asociaciones dentro de un contrato de servicios existente, así como también puede agregar y modificar características y valores de calidad de servicios, y asociaciones entre valores de calidad y contratos de servicios. El usuario puede también modificar los tipos de los elementos existentes o crear nuevos elementos para asignar como tipos de otros elementos. En todo momento el usuario puede eliminar elementos o asociaciones existentes en el diagrama.

3.1.1.6. Eliminar diagrama de Contrato de Servicios con QoS

Permite al usuario eliminar un diagrama preexistente de tipo *Service Contract with QoS*, esto es eliminar cada uno de los elementos contenidos en el diagrama y las referencias existentes a ellos en otros diagramas.

3.1.1.7. Crear diagrama de Participantes como Clases con QoS

Permite al usuario crear un diagrama de tipo *Participant Class with QoS*. Es una especificación a alto nivel en la que se pueden agregar al lienzo elementos participante y sus puertos y sus comportamientos asociados. El usuario también puede agregar características y valores de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios.

3.1.1.8. Editar diagrama de Participantes como Clases con QoS

Permite al usuario editar un diagrama de tipo *Participant Class with QoS* existente agregando nuevos elementos participante al lienzo y agregando nuevos puertos service y request a un elemento participante existente en el lienzo. Además, el usuario podrá asignar nuevos tipos a los puertos contenidos en el lienzo, crear nuevos tipos para asignar a los puertos o eliminar asignaciones de tipos existentes. El usuario también puede agregar y modificar características y valores de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios. En todo momento el usuario podrá eliminar elementos contenidos en el diagrama.

3.1.1.9. Eliminar diagrama de Participantes como Clases con QoS

Permite al usuario eliminar un diagrama preexistente de tipo *Participant Class with QoS*, esto es eliminar cada uno de los elementos contenidos en el diagrama y las referencias existentes a ellos en otros diagramas.

3.1.1.10. Crear diagrama de Participantes como Componentes con QoS

Permite al usuario crear un diagrama de tipo *Participant Component with QoS* que indica la implementación de participantes como componentes. Para esto el usuario puede agregar elementos al lienzo desde la paleta de herramientas del diagrama. Se pueden agregar componentes participante y dentro de ellos elementos tales como proveedores de servicios, puertos, asociaciones y canales de servicio entre los puertos, capacidades y sus dependencias con los participantes existentes en el diagrama, usos de arquitecturas de servicios y los correspondientes rolebindings con los proveedores de servicios asociados. El usuario también puede agregar características y valores de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios.

3.1.1.11. Editar diagrama de Participantes como Componentes con QoS

Permite al usuario editar un diagrama de tipo *Participant Component with QoS*, agregando nuevos componentes participante al lienzo y agregando nuevos elementos dentro de participantes existentes tales como proveedores de servicios, puertos, dependencias, asociaciones, canales de servicio, capacidades, usos de arquitecturas de servicios y los correspondientes rolebindings con los proveedores de servicios asociados. Además el usuario puede modificar los puertos, proveedores de servicios y capacidades, usos de arquitecturas de servicios existentes asignándoles nuevos tipos o borrando la asignación de tipos existente. El usuario también puede agregar y modificar características y valores de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios. En todo momento el usuario podrá eliminar elementos contenidos en el diagrama.

3.1.1.12. Eliminar diagrama de Participantes como Componentes con QoS

Permite al usuario eliminar un diagrama preexistente de tipo *Participant Component with QoS*, esto es eliminar cada uno de los elementos contenidos en el diagrama y las referencias existentes a ellos en otros diagramas.

3.1.1.13. Importar modelo de Características QoS

Permite al usuario importar un modelo de características QoS a un modelo SoaML+QoS, realizando una copia en profundidad de todos los elementos del modelo origen al modelo destino.

3.1.1.14. Importar modelo SoaML

Permite al usuario importar un modelo SoaML a un modelo SoaML+QoS, realizando una copia en profundidad de todos los elementos del modelo origen al modelo destino.

3.1.1.15. Importar Modelo desde archivo con formato XMI

Permite al usuario importar un modelo SoaML o SoaML+QoS desde un archivo XML con formato XMI, de forma de que se generen en el árbol del modelo los elementos contenidos en el archivo importado con sus estereotipos asociados.

3.1.1.16. Exportar Modelo a archivo con formato XMI

Permite al usuario exportar un modelo SoaML o SoaML+QoS con sus elementos y estereotipos asociados desde el editor a un archivo XML con formato XMI.

3.1.2. Generación de código

3.1.2.1. Generar proyectos Java a partir de modelo SoaML+QoS

Permite al usuario (arquitecto del sistema) generar proyectos Java a partir de modelos SoaML+QoS. El modelo puede estar en formato XMI (2.0 o 2.1) o en formato UML, y se permite generar distintos tipos de proyectos que exponen los servicios web especificados en el modelo y los clientes de estos servicios. También permite generar contratos WS-Agreement a partir de las características de calidad asociadas a los contratos especificados en el modelo, y permite generar archivos WSDL con la descripción de los servicios, y con políticas WS-Policy a partir de las características de calidad asociadas a los puertos de servicio en el modelo.

4. Solución propuesta

En esta sección se presenta la forma en la que se llevó a cabo el desarrollo de los editores QoS y SoaML+QoS y el generador de código JEE y WS partiendo de modelos SoaML con QoS.

4.1.Descripción de la solución

La solución propuesta extiende los plugins SoaML y SoaML2Code para lograr la incorporación de características de calidad al modelado con SoaML y a la generación de código JEE y WS.

Para el modelado se desarrollaron dos nuevos plugins: QoS Modelling Tool, que permite el modelado de características de calidad QoS, y SoaML+QoS Modelling Tool, que se utiliza para realizar modelos SoaML con características de calidad QoS. Adicionalmente se realizó el upgrade del plugin SoaML a la nueva versión de Papyrus, la cual tiene gran cantidad de cambios de arquitectura. Esto permite que la solución pueda utilizarse en la última versión de Eclipse (Luna) y posiblemente posteriores si los cambios no impactan en el core utilizado.

El plugin Import/Export XMI fue modificado para incorporar la importación y exportación de modelos SoaML+QoS.

En cuanto a la generación de código de QoS agregado a modelos SoaML, se incorporó al plugin SoaML2Code la generación de políticas WS-Policy en los archivos WSDL para la descripción de las características de calidad y para control de acceso a los servicios, y la generación de documentos WS-Agreement que describen las características de calidad asociadas a los contratos de servicios.

En resumen, el plugin SoaML Toolkit está compuesto por: QoS Modelling Tool, SoaML+QoS Modelling Tool, SoaML Modelling Tool, Import/Export XMI y SoaML2Code. En la Figura 35 se puede ver un diagrama que muestra las interacciónes entre los distintos componentes de la solución propuesta. Por un lado tenemos en (1) los plugins de modelado, los cuales proveen los modelos que son utilizados por el generador de código en (2). Por último, el proyecto generado en (2) es desplegado en los servidores de aplicación de (3). Además, el plugin Import/Export XMI permite el intercambio de modelos SoaML y SoaML+QoS en formato XMI, facilitando la interoperabilidad con otras herramientas.

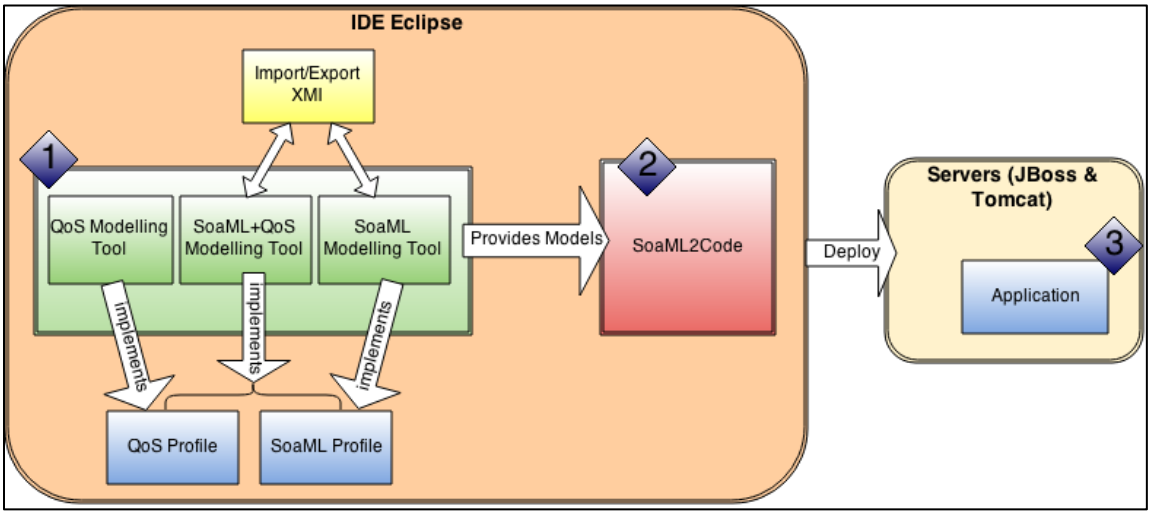

Figura 35. Interacción entre componentes de la solución

Se desarrollaron todos los plugins de modo que puedan ser instalados y utilizados de forma totalmente independiente. Para facilitar su instalación desde Eclipse, se proveen los paquetes de instalación *SoaMLToolkit_update*, que contiene lo necesario para instalar el plugin completo, y SoaMLModelling_update, SoaMLQoSModelling_update, SoaMLToJava_update, ImportExportXMI_update y QoSModelling_update, los cuales permiten instalar los plugins mencionados anteriormente de forma independiente. En el Anexo Guía de Instalación se describe esto más detalladamente.

A continuación se describe en detalle la solución provista, incluyendo la evaluación de alternativas de implementación y la descripción de la arquitectura.

4.2.Arquitectura de la solución

La arquitectura fundamental de Eclipse está basada en plugins, lo cual permite integrar elementos (nuevas funcionalidades) al sistema, enriqueciéndolo con nueva características. Cada plugin debe extender ciertos puntos de extensión básicos del core Eclipse (relacionados al workbench, vistas, perspectivas, etc.), y a su vez puede ofrecer nuevos puntos de extensión para que otros plugins puedan reutilizar o extender ciertas funcionalidades.

Todos los tipos de diagramas que ofrece Papyrus están implementados en distintos módulos (archivos jar) que se encuentran en la carpeta plugins de Eclipse (carpeta en el directorio raíz de Eclipse que contiene los plugins para extender la funcionalidad del editor). En la arquitectura propuesta, se siguió este mismo camino, por lo que cada nuevo tipo de diagrama fue distribuido en un módulo distinto, que debe agregarse a la carpeta plugins de Eclipse. Es indispensable, para el correcto funcionamiento de los nuevos diagramas, que en el Eclipse donde sean agregados se encuentre instalado el plugin Papyrus. A su vez, el plugin para la generación de código estará implementado en otro módulo independiente.

De la misma forma en que fue realizado en el proyecto del plugin SoaML, se modificaron clases y algunos archivos de configuración de los diagramas pre-existente en Papyrus. Se definió implementar dos plugins separados: por un lado el modelado de QoS, pudiendo así modelar taxonomías de características de calidad que podrán ser reutilizadas en distintos proyectos de modelado con SoaML u otros lenguajes como UML para sistemas tradicionales, y por otro, el modelado de SoaML con características de calidad QoS asociadas, manteniendo independiente el modelado de SoaML.

Se crearon por lo tanto dos plugin Papyrus que acceden al core del mismo, uno para modelado QoS y otro para modelado SoaML+QoS, los cuales facilitan las funcionalidades de aplicar los perfiles QoS y SoaML+QoS por defecto y agregar estas dos nuevas categorías de diagramas como opciones en el cuadro de diálogo al crear un nuevo modelo Papyrus.

En cuanto a la arquitectura del plugin encargado de importar y exportar archivos XMI, ésta se mantiene respecto a la versión existente del proyecto de modelado SoaML y la implementación fue modificada para que tuviera en cuenta el perfil del archivo de origen y así diferenciar la importación o exportación según si es SoaML o SoaML con QoS.

Para la arquitectura del plugin de generación de código se respetó la diseñada por el proyecto SoaML2Code, que incluye:

- Un parser que se encarga de realizar el parseo de modelos SoaML (archivo UML o XMI) para luego crear los objetos Java correspondientes.
- Un procesador de reglas que se encarga de procesar los objetos Java y aplicar reglas que definirán la invocación a los generadores de código de WS y Java EE.
- Un generador encargado de generar el código WS y Java EE que representa al modelo SoaML de partida.

En la Figura 36 se puede ver un esquema de los subsistemas involucrados y sus dependencias, donde los elementos resaltados en azul corresponden a los subsistemas desarrollados en este proyecto y los resaltados en verde son los subsistemas que ya existían que requirieron modificaciones para la inclusión de modelos SoaML+QoS. El resto son subsistemas ya existentes que no sufrieron modificaciones.

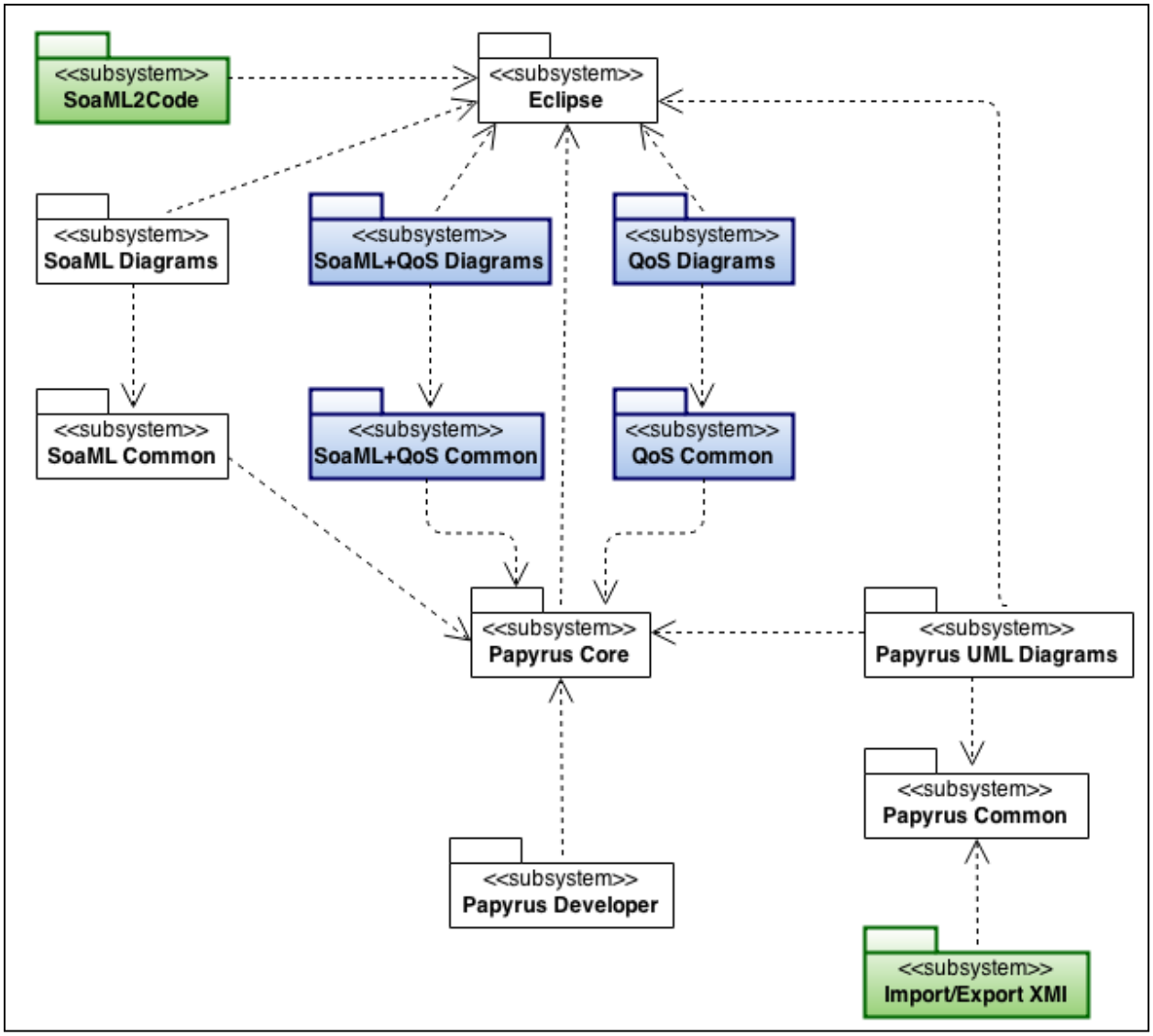

Figura 36. Subsistemas de la arquitectura Papyrus extendida

4.3.Alternativas de implementación

En esta sección se analizan las distintas opciones que se consideraron en determinados problemas a resolver, se comparan las ventajas y desventajas de cada opción y se describe la decisión tomada.

4.3.1. Arquitectura de plugins de modelado

Uno de los principales objetivos del proyecto es la extensión del plugin de modelado SoaML para permitir agregar aspectos de calidad al modelado. Para poder cumplir con este objetivo, el usuario debe poder crear diagramas con taxonomías de características de calidad, y poder asociar estas características a los servicios y contratos definidos con el perfil SoaML.

Se evaluaron tres enfoques distintos para cumplir con el objetivo:

1. Modificar el plugin de modelado SoaML, agregando un tipo de diagrama de características de calidad, y agregando ciertos elementos del perfil QoS en la paleta de los diagramas existentes.
2. Idem al punto anterior, con el agregado de la construcción de un plugin independiente que permita el modelado de características de calidad (modelo QoS). En el modelo SoaML permitir la importación de modelos QoS.
3. Mantener intacto el plugin de modelado SoaML, y construir dos plugins independientes: uno para el modelado de características de calidad (modelo QoS), y otro para el modelado de SoaML con características de calidad (modelo SoaML+QoS).

La evaluación de estas alternativas se realizó utilizando los criterios tiempo de desarrollo relativo y valor agregado al producto. La Tabla 3 resume la comparación de las tres alternativas de acuerdo a los dos criterios de evaluación, donde se ve que el valor agregado de la alternativa es proporcional al tiempo de desarrollo requerido.

Tabla 3. Comparación de alternativas de Arquitectura de plugin de modelado

	Tiempo de desarrollo	**Valor agregado**
1. Modificar plugin SoaML	Bajo	Bajo
2. Modificar plugin SoaML y construir plugin QoS	Medio	Medio
3. Construir plugin QoS y plugin SoaML+QoS	Alto	Alto

La alternativa más simple es la modificación del plugin existente para que permita el modelado QoS. Sin embargo, en este enfoque el modelado QoS está fuertemente atado al modelado SoaML: si un usuario está interesado en crear únicamente un diagrama de características de calidad, deberá crear un modelo SoaML.

La segunda alternativa ofrece un poco más de flexibilidad en el modelado, permitiendo al usuario crear diagramas de características de calidad de forma totalmente independiente a los modelos SoaML. Además, se ofrece la reutilización de un mismo modelo de calidad en distintos modelos SoaML, ya que el usuario podría importar un modelo QoS a un modelo SoaML. Claramente, esta alternativa requiere más tiempo de desarrollo, ya que además de modificar el plugin existente, se crea un nuevo plugin de modelado.

La última es la alternativa que ofrece al usuario la mayor flexibilidad de modelado: permite la creación de modelos QoS, de SoaML y de SoaML con QoS de forma totalmente independiente. Además se ofrece

la importación de modelos QoS a modelos SoaML+QoS, permitiendo la reutilización de taxonomías de características de calidad. También se pueden importar los elementos de un modelo SoaML en un modelo SoaML+QoS, permitiendo agregar características de calidad a un modelo SoaML sin necesidad de crear el modelo desde cero. Si bien es la alternativa que ofrece el mayor valor agregado, es la que requiere más tiempo de desarrollo.

Se decidió darle más importancia al valor agregado que al tiempo de desarrollo requerido, y por lo tanto se optó por la construcción de un plugin de modelado QoS y otro plugin de modelado SoaML+QoS, sin realizar modificaciones al plugin de modelado SoaML.

4.3.2. Categorización de características de calidad en la generación de código

Para el modelado de SoaML con QoS nos basamos en el estudio realizado por Schot [18], descripto en la sección 2.7, en el que se define que las restricciones de calidad se asocian a los puertos de los participantes a través de dependencias QoSOffered y QoSRequired.

En el generador de código decidimos generar cosas distintas según el tipo de característica de calidad, como por ejemplo, control de acceso mediante una política Username Token, definición de políticas en el WSDL para describir aspectos de calidad del servicio, o registro de tiempos de procesamiento en los clientes de servicios web. El problema que se nos planteó fue cómo determinar qué es lo que se debe generar para cada restricción QoS en el modelo. Para este problema se analizaron las siguientes soluciones posibles:

1. Utilizar el nombre de la dimensión del elemento QoSValue para determinar qué es lo que se debe generar. Se puede proveer un modelo externo al plugin, que contiene las características de calidad que el generador sabe implementar.
2. Además de tener el elemento QoSCharacteristic, en la paleta de los diagramas se incluyen elementos que son instancias de QoSCharacteristic; cuando el usuario inserta una de estas instancias al diagrama, se está insertando un elemento del tipo QoSCharacteristic con determinado nombre. El generador utiliza el nombre del elemento para determinar qué tipo de característica es.
3. Cuando el usuario inserte un elemento QoSCharacteristic a un diagrama, desplegar una pantalla con la lista de características que el generador sabe implementar; el generador utiliza el nombre del elemento para determinar qué tipo de característica es.
4. Agregar una propiedad en el elemento QoSCharacteristic, cuyo tipo es un enumerado con los distintos tipos de características que el generador sabe implementar.
5. En el wizard de generación de código, agregar permitirle al usuario crear un mapa de asociaciones entre las características definidas en el modelo con los tipos de características que el generador sabe implementar.
6. En la paleta de los diagramas se incluyen elementos que son especializaciones de QoSChatacteristic, donde cada elemento es una característica que el generador sabe implementar. El generador se basa en el tipo del elemento para determinar qué tipo de característica es.

Para la evaluación de estas alternativas se tomaron en cuenta los siguientes criterios: adecuación al estándar QoS, usabilidad, tiempo de investigación necesario y facilidad de extensión de nuevas

características en el generador. En la Tabla 4 se muestra la comparación de las alternativas según los criterios de evaluación.

Tabla 4. Comparación de alternativas de categorización de características de calidad en la generación de código

	Adecuación al estándar	**Usabilidad**	**Tiempo de investigación**	**Facilidad de extensión**
1. Utilizar el nombre de la característica	Alta	Baja	No requiere	Muy fácil
2. Agregar instancias de QoSCharacteristic en la paleta	Alta	Media	Alto	Complejo
3. Desplegar lista de características	Alta	Media	Medio	Medio
4. Agregar propiedad a QoSCharacteristic	Baja	Alta	No requiere	Medio
5. Agregar mapa de asociaciones entre características del modelo y tipos de características	Alta	Medio-Alta	No requiere	Muy fácil
6. Agregar especializaciones de QoSCharacteristic en la paleta	Baja	Alta	No requiere	Complejo

La primera alternativa es la más simple de todas, no requiere tiempo de investigación y no es necesario realizar modificaciones al perfil QoS. Sin embargo, es la que presenta peor usabilidad hacia el usuario final. En primer lugar, el usuario debe conocer el nombre de cada característica que el generador entiende, y debe adaptar su taxonomía de características de calidad a esos nombres. Además, los nombres de las características que el generador sabe implementar deben definirse en un idioma (ya sea español o inglés), y el usuario puede querer realizar modelos de calidad en otro idioma. Cuando se agrega la generación de nuevas características, no es necesario realizar modificaciones en el plugin de modelado.

La segunda alternativa tampoco requiere modificaciones en el perfil QoS, y tiene la ventaja con respecto a la primera opción en que el usuario no necesita conocer y recordar exactamente el nombre de las características que el generador puede implementar. Se sigue manteniendo el problema del idioma del modelo y de que el usuario debe adaptar su taxonomía a los nombres específicos. Además, esta alternativa requiere un tiempo de investigación considerable para saber cómo se pueden agregar instancias de elementos en la paleta de Papyrus. Otra desventaja de esta alternativa es que si se agrega la generación de nuevas características, será necesario modificar el plugin de modelado para agregar las nuevas características en la paleta.

La tercera opción es similar a la segunda, ya que el generador utiliza los nombres de los elementos para determinar el tipo de generación. No se deben realizar modificaciones al perfil QoS y el usuario no necesita recordar el nombre exacto de las características que el generador puede implementar. Se sigue manteniendo el problema del idioma del modelo y de que el usuario debe adaptar su taxonomía a los nombres específicos. Además, se requiere de tiempo de investigación para determinar cómo se puede desplegar una lista de opciones en Papyrus cuando un usuario inserta un elemento a un diagrama. Otra desventaja de esta alternativa es que si se agrega la generación de nuevas características, será necesario

modificar el plugin de modelado para agregar las nuevas características en la lista de características que se despliega al insertar un QoSCharacteristic al diagrama.

La cuarta opción no requiere tiempo de investigación, y a nuestro entender es la que ofrece mayor usabilidad al usuario. Sin embargo, es necesario realizar modificaciones al perfil QoS para agregar una propiedad en el elemento QoSCharacteristic. Al agregarse la generación de nuevas características, es necesario modificar el plugin de modelado para agregar estas características al enumerado de tipos de características.

La quinta opción tampoco requiere de investigación, y cuenta con las ventajas de adaptarse al perfil QoS sin modificarlo, y la facilidad de extensión de generación de nuevas características ya que no es necesario modificar el plugin de modelado. La usabilidad de esta alternativa no es tan buena ya que el usuario debe elegir el tipo de cada característica que se especificó en el modelo cada vez que desea generar código. Además, no existe un mecanismo para controlar que la selección del usuario tenga sentido. Por ejemplo, el usuario podría mapear la característica *Service Response Time* a la implementación *WS-Security Username Token*, generando código para el control de acceso cuando en realidad tendría sentido generar lógica de medición de tiempo de respuesta.

La última opción no requiere tiempo de investigación y ofrece mayor usabilidad, pero requiere realizar cambios en el perfil QoS (agregar especializaciones de QoSCharacteristic) y requiere modificar el plugin de modelado cada vez que se agrega la generación de nuevas características.

Para decidir cuál enfoque tomar, primero descartamos las alternativas que requieren modificar el perfil QoS y luego dimos prioridad a las que requieren menos tiempo de investigación. Luego de filtrar las opciones según esos dos criterios, nos quedamos con la opción que ofrece mayor flexibilidad. De esta manera, se decidió realizar un mapa de asociaciones entre las características del modelo y los tipos de características que el generador puede implementar, según lo presentado en la opción 5.

4.3.3. Descripción de características de calidad en WSDL

Uno de los requerimientos del proyecto es agregar las características de calidad definidas en los modelos a la generación de código. Se decidió que el generador de código agregue las características de calidad en los archivos WSDL. Se plantearon dos alternativas:

1. Definir las características de calidad como políticas WS-Policy asociadas al servicio.
2. Definir las características de calidad utilizando la extensión Q-WSDL propuesta en [36].

Para la evaluación de estas alternativas se establecieron los siguientes criterios: tiempo de desarrollo, adecuación a estándares, capacidad de modelado. La Tabla 5 muestra la comparación de las alternativas según estos criterios de evaluación.

Tabla 5. Comparación de alternativas de Descripción de características de calidad en WSDL

	Tiempo de desarrollo	**Adecuación a estándares**	**Capacidad de modelado**
1. Políticas WS-Policy	Bajo	Alto	Media
2. Extensión Q-WSDL	Alto	Bajo	Alta

La especificación de características de calidad utilizando políticas WS-Policy asociadas al servicio tiene las ventajas de requerir poco tiempo de desarrollo y adecuarse a estándares que son reconocidos y utilizados a nivel mundial.

La especificación Q-WSDL propuesta por D'Ambrogio es una extensión de WSDL, que permite la descripción de características QoS de un servicio web. Si bien es muy interesante el framework que plantea para introducir aspectos no funcionales en el WSDL a nivel de mensajes, operaciones, servicios y puertos, requiere cambios en el esquema WSDL y no existen grandes organizaciones que promuevan su utilización. Además el tiempo de desarrollo es alto, ya que requiere realizar cambios en toda la estructura de los documentos WSDL.

Para elegir el enfoque a utilizar, dimos prioridad a la adecuación a estándares y al tiempo de desarrollo requerido, por lo que decidimos modelar las características de calidad en los archivos WSDL como políticas WS-Policy.

4.3.4. Importación de modelos QoS

Al utilizar el perfil SoaML+QoS el usuario puede importar modelos QoS, permitiendo la reutilización de taxonomías de características de calidad. Para realizar esta importación se detectaron dos alternativas:

1. Utilizar la funcionalidad provista por Papyrus para importar paquetes existentes en modelos externos.
2. Implementar la importación específica de elementos contenidos en un modelo QoS.

Se utilizaron los siguientes criterios de evaluación: tiempo de desarrollo y usabilidad. La Tabla 6 muestra la comparación de las alternativas según estos criterios de evaluación.

Tabla 6. Comparación de alternativas de Importación de modelos QoS

	Tiempo de desarrollo	**Usabilidad**
1. Importación de Papyrus	Alto	Baja
2. Importación implementada por nosotros	Bajo	Media

Al importar un modelo utilizando la funcionalidad de Papyrus, éste es referenciado desde el modelo SoaML+QoS, es decir, los elementos no son copiados de un modelo al otro. Esto provoca que las modificaciones realizadas sobre los elementos QoS referenciados también afecten el modelo original, resultando en una baja usabilidad. Esta alternativa requiere un tiempo de desarrollo alto ya que es necesario modificar el generador de código para parsear, no solamente el modelo SoaML+QoS, sino que también se debe parsear el modelo QoS referenciado. También es necesario modificar el plugin Import/Export XMI para que si se quiere exportar un modelo SoaML+QoS también se exporte el modelo QoS referenciado.

La importación implementada por nosotros tiene un bajo tiempo de desarrollo ya que sólo es necesario desarrollar el parseo del archivo UML para obtener los elementos QoS y copiarlos al archivo UML del modelo SoaML+QoS. La usabilidad de esta alternativa tiene algunas ventajas sobre la anterior pero cuenta con la desventaja de que las actualizaciones de la taxonomía en el modelo QoS no se realizan en el modelo SoaML+QoS donde fue importada, por lo que se deben realizar manualmente.

Como en ambos criterios de evaluación la segunda opción es mejor que la primera, se decidió implementar la importación específica de elementos contenidos en un modelo QoS.

4.4.Implementación

Esta sección describe la forma en que se implementaron los distintos plugins para brindar la funcionalidad deseada.

Como se mencionó anteriormente, se decidió implementar dos plugins de forma independiente, uno para el modelado de QoS y otro para el modelado de SoaML con QoS. Los distintos componentes, así como sus relaciones y dependencias se pueden ver en la Figura 37.

Se crearon dos nuevos plugins *"common"* distintos a los usados por los plugins pre-existentes. Uno para QoS, el cual permite aplicar el perfil QoS por defecto en el diagrama de tipo QoS y además, agregar una nueva categoría de diagramas. El otro es para SoaML+QoS, el cual, así como el anterior, permite aplicar el perfil SoaML+QoS por defecto en los nuevos diagramas de tipo SoaML+QoS y agregar una nueva categoría de diagramas.

Fue necesario generar código a partir de los perfiles, utilizando EMF, para que funcionaran correctamente los menús de Papyrus y éste mostrara las distintas opciones de diagramas que se pueden generar.

El editor de diagramas de *QoS Characteristics* fue realizado en base al editor de diagrama de clases pre-existente. Para SoaML+QoS se tomó como base lo implementado para SoaML, modificando los plugins que debían incluir características de calidad.

Como los editores de diagramas de *Service Contract*, *Participant Class* y *Participant Component* fueron creados a partir del editor de diagrama de estructura compuesta de Papyrus, ninguno contaba con el elemento *Instance Specification*, el cual es necesario para implementar el elemento *QoSValue*. Para lograr que funcione se debió agregar manualmente todo lo referente al elemento *Instance Specification*, utilizando como base el código del editor de diagrama de clases y copiándolo donde fuera necesario.

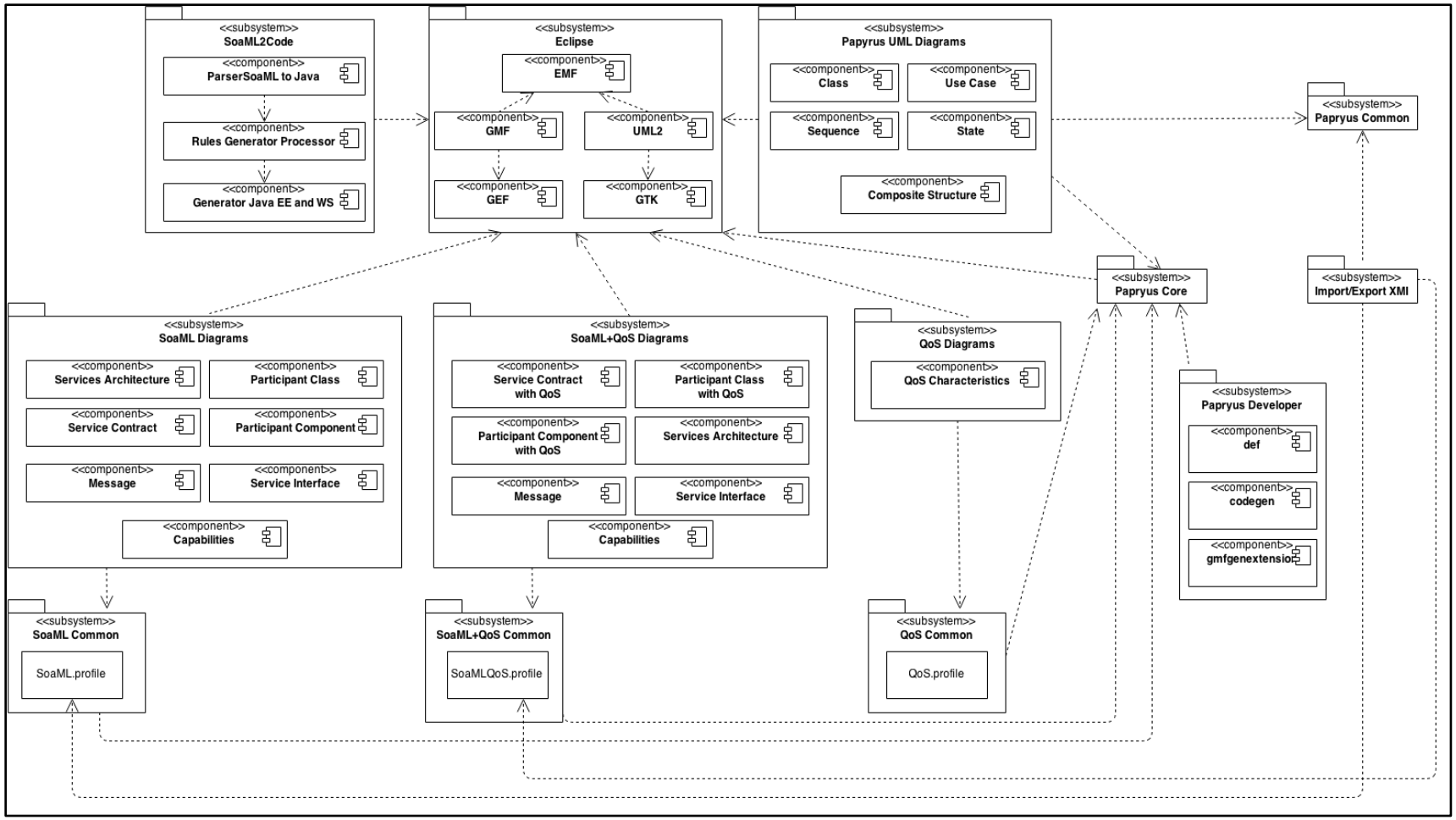

Figura 37. Diagrama de componentes de la solución

Para el generador de código se analizaron los elementos del dominio que componen el modelo SoaML con QoS y se estudió el archivo de intercambio de modelos en formato XMI. De este análisis se obtuvo el nuevo modelo de dominio que extiende el realizado en el proyecto SoaML2Code con elementos QoS. Este modelo de dominio se ilustra ver en la Figura 38, en donde los elementos resaltados en azul corresponden a los agregados en este proyecto.

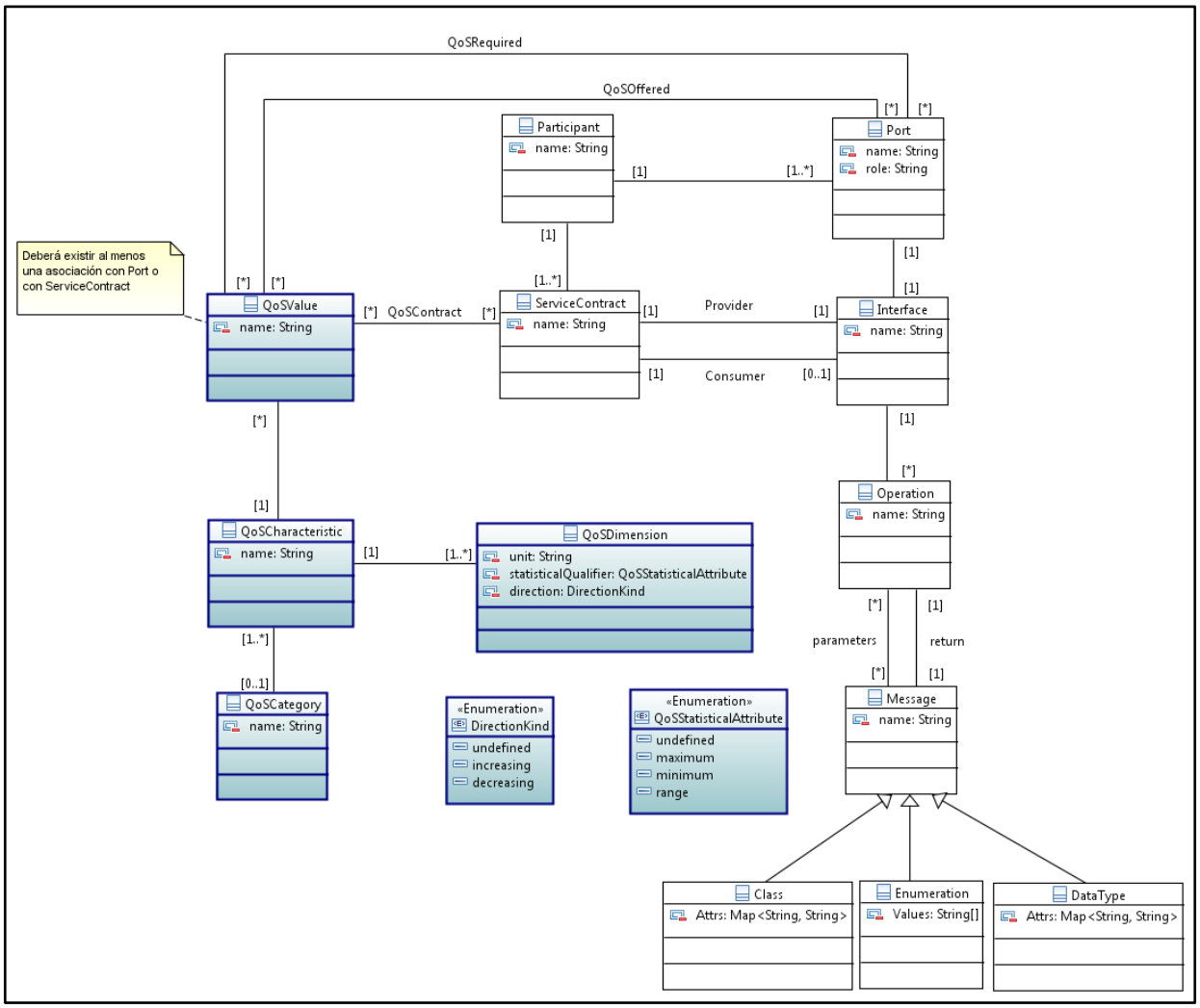

Figura 38. Modelo de dominio SoaML+QoS

El parseo existente del archivo XMI que representa el modelo SoaML se modificó para incluir los elementos de QoS que pueden aparecer o no en dicho modelo. Cabe destacar que el formato del archivo XMI a procesar debe cumplir con la estructura general presentada en la Figura 39. En particular, dentro del paquete QoS (que debe aparecer en cuarta posición respecto a los demás paquetes) deben estar todos los elementos QoS del modelo. Para generarlo de esta forma se modificó también la exportación del archivo XMI del modelo SoaML para incluir el paquete QoS y sus elementos.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI>
    <uml:Model name="PatientMASProcess" xmi:id="0">
        <packagedElement name="Participants">
            <packagedElement name="Participant1"> ... </packagedElement>
            ..
            <packagedElement name="ParticipantN"> ... </packagedElement>
        </packagedElement>
        <packagedElement name="Messages">
            <packagedElement name="Message1"/>
            ..
            <packagedElement name="MessageN"/>
        </packagedElement>
        <packagedElement name="Services">
            <packagedElement name="Service1">
                <packagedElement>
                    <ownedOperation name="Operation1">
                        <ownedParameter name="parameter"/>
                    </ownedOperation>
                </packagedElement>
                ..
                <packagedElement>
                    <ownedOperation name="OperationN">
                        <ownedParameter name="parameter"/>
                    </ownedOperation>
                </packagedElement>
            </packagedElement>
            ..
            <packagedElement name="ServiceN">
                <packagedElement>
                    <ownedOperation name="Operation1">
                        <ownedParameter name="parameter"/>
                    </ownedOperation>
                </packagedElement>
                ..
                <packagedElement>
                    <ownedOperation name="OperationN">
                        <ownedParameter name="parameter"/>
                    </ownedOperation>
                </packagedElement>
            </packagedElement>
        </packagedElement>
        <packagedElement name="QoS">
            ...
        </packagedElement>
        ..
</xmi:XMI>
```

Figura 39. Estructura general archivo XMI entrada del parser

Luego de tener el archivo parseado, resta especificar el modelo Java de correspondencia con el modelo SoaML+QoS que fue definido, es decir, la transformación de los objetos del modelo SoaML+QoS a sus correspondientes objetos Java que generan la implementación del modelo. Dicha transformación se ilustra en la Figura 40. El color azul indica que los elementos fueron incorporados en este proyecto, el verde que fueron modificados y el blanco que se mantuvieron sin modificaciones.

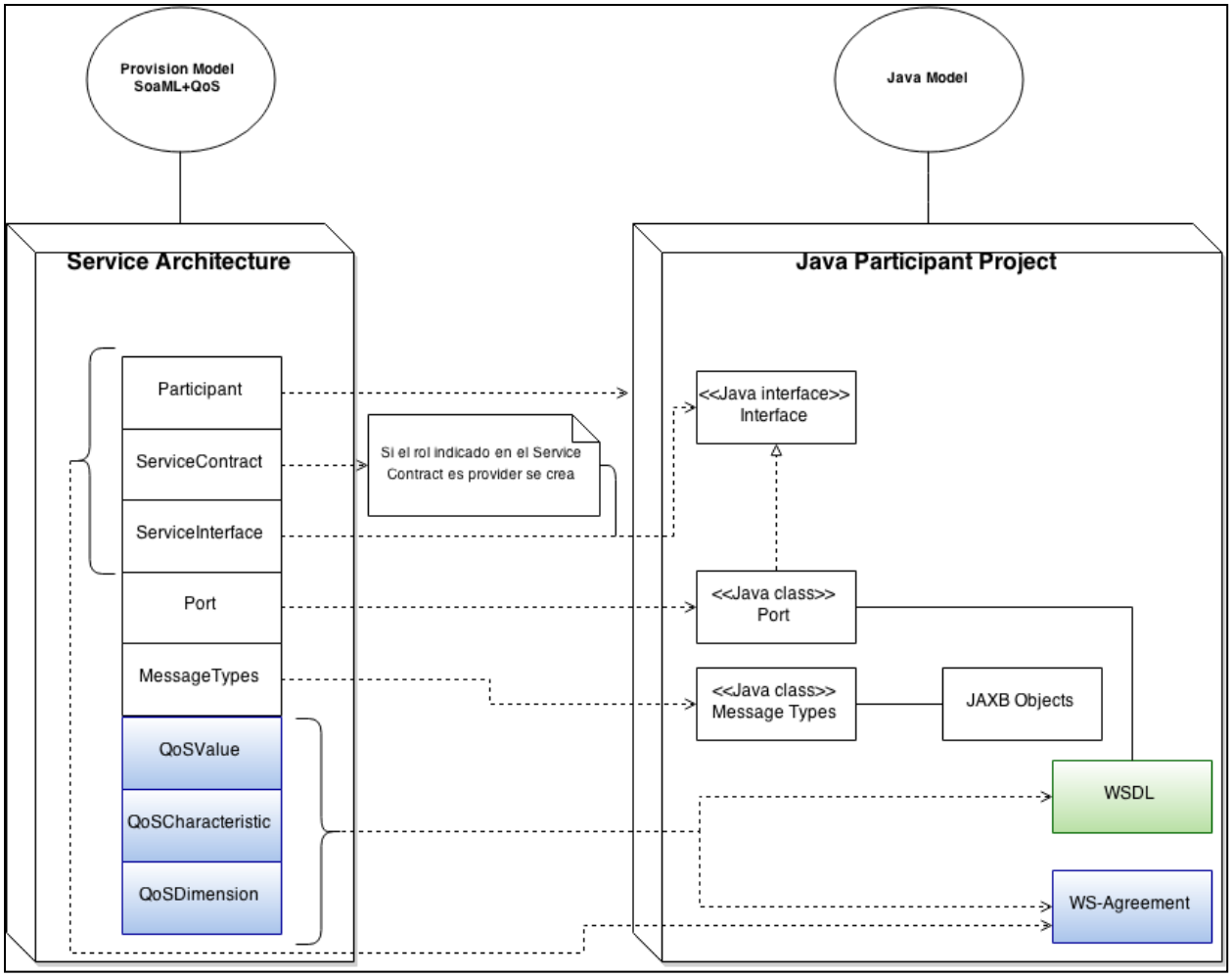

Figura 40. Implementación Java de modelo SoaML+QoS

Para la generación de código de características de calidad se implementaron cuatro opciones:

- WS-Agreement para los contratos de servicios.
- WS-Policy Generic, la cual permite la generación de políticas genéricas a partir de las características definidas.
- WS-Policy Response Time, la cual permite el registro de los tiempos de los servicios.
- WS-Security Username Token, la cual permite el control de acceso a los servicios.

En la Tabla 7 se muestra qué tipo de política se genera para los distintos casos de generación de código. En la generación de servidores EJB, los archivos WSDL contienen las políticas de calidad y de control de acceso. Las clases Java necesarias para validar el token de seguridad o la configuración para ignorar las políticas de calidad cuando se recibe un mensaje SOAP son altamente dependientes del framework y del servidor donde serán desplegados los servicios. Por este motivo y dado que el proyecto EJB genera una biblioteca que puede ser referenciada desde proyectos web con distintas implementaciones y desplegados sobre distintos servidores, decidimos no generar las clases de validación de Username Token y configuración de WS-Policy.

Tabla 7. WS-Policies y Tecnologías

QoS	EJB	JAX-WS RI JBoss	JAX-WS RI Tomcat	JAX-WS + Spring Tomcat
WS-Policy Generic	Sólo wsdl	✓	✓	✓
WS-Policy Response Time	Sólo wsdl	✓	✓	✓
WS-Security Username Token	Sólo wsdl	✓	✓	✓
WS-Agreement	✓	✓	✓	✓

El generador de código se puede extender para generar nuevos tipos de características de calidad. Para esto es necesario modificar la interfaz de usuario (ya sea agregando nuevos pasos al wizard o agregando nuevos tipos de políticas) y el back-end. En el Anexo Documento Técnico se describe en detalle los pasos a seguir para realizar estas modificaciones.

A continuación, en primer lugar presentaremos la generación realizada por cada tipo de QoS según la correspondencia con las clases Java presentada previamente. En segundo lugar presentaremos para cada columna de la Tabla 7 correspondiente a la tecnología destino de la generación, los detalles de la misma.

4.4.1. Generación WS-Agreement

Para generar documentos WS-Agreement, para cada participante generado se obtienen todos los elementos del tipo Service Contract que contengan al participante (ya sea como proveedor o consumidor) y que estén asociados a al menos un elemento QoSValue. Para cada uno de los contratos, se genera un documento WS-Agreement que contiene parte de la información del contrato y del servicio implementado, y en donde se especifican las características de calidad asociadas al contrato. El contrato para el cual el participante esté como proveedor se genera en el proyecto del servidor, y en el proyecto del cliente se genera el contrato para el cual el participante esté como consumidor. La Figura 41 muestra la transformación entre los objetos Java y las distintas partes del documento WS-Agreement generado.

Figura 41. Mapeo de objetos Java a componentes del documento WS-Agreement

4.4.2. Generación WS-Policy

Para generar WS-Policy, el archivo WSDL es modificado para incluir las distintas políticas disponibles. Para WS-Policy Generic y WS-Policy Response Time, el resultado en el archivo WSDL es el mismo. Por cada QoSDimension dentro de un QoSValue asociado al puerto de servicio, se genera una política WS-Policy cuyo nombre es el de la dimensión de calidad. Esta transformación se ilustra en la Figura 42.

Figura 42. Mapeo de objetos Java a políticas genéricas en el WSDL

4.4.3. Generación WS-Security

Cuando el usuario indica que determinada dimensión de calidad se implemente como WS-Security UsernameToken, en el WSDL se agrega una política de seguridad UsernameToken, sin importar el valor definido en el elemento QoSValue ni las propiedades de la dimensión de calidad. En la Figura 43 se observa el resultado de la generación del WSDL con WS-Security UsernameToken.

```
<wsdl:service name="ReceiveSurgerydateReservationService">
    <wsdl:port
        binding="serv:ReceiveSurgerydateReservationServiceBinding" name="ReceiveSurgerydateReservation">
        <soap:address location="http://localhost:8080/PatientServer/ReceiveSurgerydateReservationService"/>
        <wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy">
            <wsp:All>
                <sp:SupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
                    <wsp:Policy>
                        <sp:UsernameToken sp:IncludeToken=
                        "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                            <wsp:Policy>
                                <sp:WssUsernameToken11/>
                            </wsp:Policy>
                        </sp:UsernameToken>
                    </wsp:Policy>
                </sp:SupportingTokens>
            </wsp:All>
        </wsp:Policy>
    </wsdl:port>
</wsdl:service>
```

Figura 43. Ejemplo de policy Username Token

4.4.4. Generación de políticas combinadas

Si se generan distintos tipos de políticas, éstas se anidan como se muestra en la Figura 44.

```
<wsdl:service name="ReceiveSurgerydateReservationService">
    <wsdl:port
        binding="serv:ReceiveSurgerydateReservationServiceBinding" name="ReceiveSurgerydateReservation">
        <soap:address location="http://localhost:8080/PatientServer/ReceiveSurgerydateReservationService"/>
        <wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy">
            <wsp:All>
                <ServiceLatencyTime direction="decreasing" unit="ms" value="10"/>
                <sp:SupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
                    <wsp:Policy>
                        <sp:UsernameToken sp:IncludeToken=
                        "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                            <wsp:Policy>
                                <sp:WssUsernameToken11/>
                            </wsp:Policy>
                        </sp:UsernameToken>
                    </wsp:Policy>
                </sp:SupportingTokens>
            </wsp:All>
        </wsp:Policy>
    </wsdl:port>
</wsdl:service>
```

Figura 44. Ejemplo de varias policies

La generación de código depende directamente de qué tipo de proyecto se vaya a generar y qué implementaciones de características de calidad. A continuación describimos para cada tecnología los detalles de la generación realizada.

4.4.5. Generación para Dynamic Web Project - JAX-WS ri

En la versión anterior de SoaML2Code se generaba el mismo código ya sea para JBoss como para Tomcat. Con la nueva versión de JBoss utilizada hay elementos que ya no son necesarios incorporar al proyecto

generado, por lo que se optó por dividir las opciones para que el usuario pueda decidir qué servidor desea utilizar y así generar códigos diferentes.

4.4.5.1. Generación para JBoss

Con la versión 7.1.1 de JBoss ya no resulta necesario agregar las librerías de JAX-WS ya que éstas ya vienen incluidas. Tampoco es necesario agregar los servlets al archivo web.xml como se hacía anteriormente.

Las WS-Policy Generic que generamos sirven para describir las características de calidad de los servicios y no para restringir aspectos en el intercambio de mensajes. Por este motivo es que se le debe instruir al servidor que ignore esas políticas al momento de recibir una invocación. Para esto se incorpora al código del proyecto las clases *IgnorablePolicyInterceptorProvider* y *PolicyInterceptorProviderInstallerInterceptor*. La primera extiende *AbstractPolicyInterceptorProvider* y es la que se encarga de marcar la política como verificada. La segunda extiende *AbstractPhaseInterceptor<Message>* y es la que se encarga de registrar las políticas genéricas definidas junto con el interceptor que las ignora. Para que esto funcione, se debe anotar la interfaz como se muestra en la Figura 45.

```
@InInterceptors(interceptors = {"qos.PolicyInterceptorProviderInstallerInterceptor"})
```

Figura 45. Interfaz anotada con interceptor

Para que el proyecto compile se agregan al *buildpath* las librerías cxf-api-2.4.6.jar, cxf-rt-ws-policy-2.4.6.jar y cxf-common-utilities-2.4.6.jar contenidas en la instalación del JBoss. Para el runtime se crea el archivo de configuración jboss-deployment-structure.xml en donde se agregan las dependencias a librerías incluidas en JBoss. En este caso se agrega la dependencia al módulo org.apache.cxf, como se muestra en la Figura 46.

```
<dependencies>
    <module name="org.apache.cxf" />
</dependencies>
```

Figura 46. Dependencia org.apache.cxf

Si se selecciona WS-Policy Response Time lo generado es lo mismo que para WS-Policy generic, lo que difiere es lo que se genera en el cliente.

Al seleccionar WS-Security Username Token se crea la clase *ServerPasswordCallback* la cual implementa *CallbackHandler* y es quien se encarga de validar que el usuario y contraseña enviados por el cliente son correctos. Para añadir el *handler* a los servicios se crea el archivo jaxws-endpoint-config.xml que tiene el formato mostrado en la Figura 47.

```
<?xml version="1.0" encoding="UTF-8"?>
<jaxws-config xmlns="urn:jboss:jbossws-jaxws-config:4.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:javaee="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="urn:jboss:jbossws-jaxws-config:4.0 schema/jbossws-jaxws-config_4_0.xsd">
    <endpoint-config>
        <config-name>Custom WS-Security Endpoint</config-name>
        <property>
            <property-name>ws-security.callback-handler</property-name>
            <property-value>qos.ServerPasswordCallback</property-value>
        </property>
    </endpoint-config>
</jaxws-config>
```

Figura 47. Archivo jaxws-endpoint-config.xml

La clase que implementa el web service se anota para que sea invocado el *handler* como se puede ver en la Figura 48.

```
@org.jboss.ws.api.annotation.EndpointConfig(configFile = "WEB-INF//jaxws-endpoint-config.xml",
                                            configName = "Custom WS-Security Endpoint")
```

Figura 48. Anotación en clase que implementa servicio

Para que el proyecto compile se agregan al *buildpath* las librerías wss4j-1.6.5.jar y jbossws-api-1.0.0.GA.jar contenidas en la instalación del JBoss. Para el runtime se crea el archivo de configuración jboss-deployment-structure.xml en donde se agregan las dependencias a librerías incluidas en JBoss. En este caso se agrega la dependencia a los módulos org.apache.ws.security y org.jboss.ws.api, como se muestra en la Figura 49.

```
<dependencies>
    <module name="org.apache.ws.security" />
    <module name="org.jboss.ws.api" />
</dependencies>
```

Figura 49. Dependencias org.apache.ws.security y org.jboss.ws.api

Para la correcta generación de este tipo de proyecto es necesario tener instalado el server JBoss en el Eclipse y haber agregado el plugin de JAX-WS ri a la carpeta plugins de Eclipse. Para que el proyecto compile se debe crear la variable JBOSS_HOME en Eclipse apuntando a la instalación de JBoss (Ver Anexo Guía de Instalación).

4.4.5.2. Generación para Tomcat

Para que funcionen las políticas genéricas no fue necesario ninguna configuración especial.

Para WS-Security Username Token se crea la clase *ServerPasswordCallback*, la cual es igual a la utilizada para JBoss, a diferencia del *import* de la clase WSPasswordCallback. En JBoss dicha clase se obtiene del paquete *org.apache.ws.security* mientras que en Tomcat se obtiene de *org.apache.wss4j.common.ext*. También se crea la clase WSUsernameTokenValidator, la cual implementa *SOAPHandler<SOAPMessageContext>* y es la encargada de validar los los mensajes SOAP entrantes mediante la invocación a la clase WSPasswordCallback.

Por otro lado, se crea el archivo *handler.xml*, el cual configura que todos los mensajes SOAP sean interceptados por WSUsernameTokenValidator. Éste tiene el formato que se muestra en la Figura 50.

```
<?xml version="1.0" encoding="UTF-8"?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/javaee_web_services_metadata_handler_2_0.xsd">
    <handler-chain>
        <handler>
            <handler-name>WSUsernameTokenValidator</handler-name>
            <handler-class>qos.WSUsernameTokenValidator</handler-class>
        </handler>
    </handler-chain>
</handler-chains>
```

Figura 50. Archivo handler.xml

La clase que implementa el web service se anota para que sea invocado el *handler* como se puede ver en la Figura 51.

```
@javax.jws.HandlerChain(file = "./qos/handlers.xml")
```

Figura 51. Anotación HandlerChain en clase que implementa servicio

Por último, se agregan las bibliotecas de WSS4J y sus bibliotecas referenciadas necesarias para que compile y funcione el servicio en el servidor, como se puede ver en la Figura 52. WSS4J es una biblioteca que provee una implementación de la mayoría de estándares WS-Security, y es la utilizada en los proyectos generados con JAX-WS ri con Tomcat para procesar los Username Tokens.

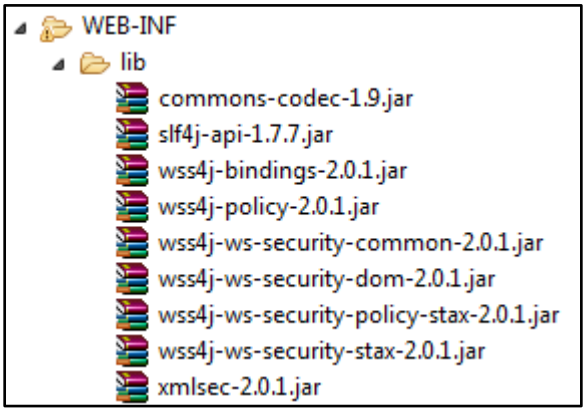

Figura 52. Bibliotecas necesarias para wss4j

4.4.6. Generación para Dynamic Web Project - JAX-WS + spring

Se actualizó la versión de Apache CXF a la 3.0.1 lo cual implicó algunos cambios en el archivo beans.xml.

Para que funcione WS-Policy Generic el servidor debe, al igual que para JBoss con JAX-WS ri, ignorar la política cuando el servicio es invocado. Para esto se modifica el beans.xml, agregando el interceptor provisto por CXF *org.apache.cxf.ws.policy.IgnorablePolicyInterceptorProvider*, como se puede ver en el ejemplo de la Figura 53.

```
<bean class="org.apache.cxf.ws.policy.IgnorablePolicyInterceptorProvider">
    <constructor-arg>
        <list>
            <bean class="javax.xml.namespace.QName">
                <constructor-arg value="ServiceLatencyTime"/>
            </bean>
        </list>
    </constructor-arg>
</bean>
```

Figura 53. Bean IgnorablePolicyInterceptorProvider

De la misma forma que para JBoss, si se selecciona WS-Policy Response Time lo generado es lo mismo que para WS-Policy generic. Lo que difiere es lo que se genera en el cliente, lo que se explicará más adelante.

Para WS-Security Username Token se crea la clase ServerPasswordCallback, la misma utilizada para el proyecto Dynamic Web Project - JAX-WS ri para Tomcat. Para añadir el handler a los servicios también se debe utilizar la clave ws-security.callback-handler pero en este caso se configura en el beans.xml, como se muestra en la Figura 54.

```
<jaxws:properties>
    <entry key="ws-security.callback-handler" value="qos.ServerPasswordCallback"/>
    <entry key="ws-security.enable.nonce.cache" value="false"/>
</jaxws:properties>
```

Figura 54. Extracto de archivo beans.xml

4.4.7. Generación del Cliente para consumo del WS

El cliente se genera de forma independiente de la tecnología seleccionada para el servidor.

Para WS-Policy Generic no se debe generar nada específico del lado del cliente ya que el servidor no espera nada e incluso ignora la política.

Para WS-Policy Response Time se registran en un archivo de log los tiempos de inicio y fin del método que invoca al servicio, lo cual permite comprobar si los tiempos de respuesta cumplen con las políticas de cada servicio. Para esto se agregó la librería *log4j2* al cliente generado junto con el archivo de configuración log4j2.xml en el cual se puede definir el archivo en el que se quieren imprimir los logs y el formato de éste, entre otras cosas. El archivo de configuración fue implementado de forma tal que el log del proyecto generado cumpla con el formato esperado por la herramienta ProM [38] de minería de procesos. En la Figura 55 se puede ver un ejemplo de log con dicho formato.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- MXML version 1.0 -->
<!-- This is a process enactment event log created to be analyzed by ProM. -->
<!-- ProM is the process mining framework. It can be freely obtained at http://www.processmining.org/. -->
<WorkflowLog
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://is.tm.tue.nl/research/processmining/WorkflowLog.xsd"
    description="CPN Tools simulation log">
    <Source program="CPN Tools simulation"/>
    <Process id="Log Servicio Caso de Estudio" description="Simulated process">
        <ProcessInstance id="1" description="Simulated process instance">
            <AuditTrailEntry>
                <WorkflowModelElement>ReceiveSurgerydateReservationService</WorkflowModelElement>
                <EventType>start</EventType>
                <Timestamp>2014-10-28 18:30:46.554</Timestamp>
                <Originator>Service</Originator>
            </AuditTrailEntry>
            <AuditTrailEntry>
                <WorkflowModelElement>ReceiveSurgerydateReservationService</WorkflowModelElement>
                <EventType>complete</EventType>
                <Timestamp>2014-10-28 18:30:47.898</Timestamp>
                <Originator>Service</Originator>
            </AuditTrailEntry>
        </ProcessInstance>
    </Process>
</WorkflowLog>
```

Figura 55. Ejemplo de log

Para WS-Security Username Token se crea la clase UsernameTokenUtil, la cual se encarga de agregar el usuario y contraseña al cabezal SOAP. Esta es invocada desde la clase main del cliente.

Para finalizar, se muestra en la Tabla 8 un resumen de las plataformas tecnológicas de destino que el plugin provee.

Tabla 8. Tecnologías destino del generador de código

Descripción de servicios web	WSDL 1.1 (SOAP 1.1)
	XML Schema 1.1
	WS-Policy
	WS-Security y WS-SecurityPolicy
Contratos de calidad de servicios web	WS-Agreement
Java EE	JDK 7 o superior
	EJB 3.1
	JAX-WS ri 2.2.8
	JAX-WS + Spring 2.0
Entorno de desarrollo	Eclipse Web Tools

4.5.Dificultades encontradas

En esta sección se describen un conjunto de problemas encontrados a lo largo de la implementación. Las dificultades relacionadas con el código fueron enfrentadas mediante estudio e investigación en los foros de Eclipse relacionados con la creación de plugins, entre otros.

4.5.1. Papyrus

Uno de los principales problemas con los que nos enfrentamos fue la migración de Papyrus de su versión 0.7 a 1.0. El grupo que desarrolló el plugin SoaML ya había manifestado el inconveniente de la poca

documentación de la arquitectura y diseño de Papyrus y nosotros coincidimos en que es una gran traba. En particular, no encontramos información detallada sobre la migración que sufrió Papyrus y la mayor parte de la documentación estaba obsoleta. Al igual que el grupo anterior, nos fue de bastante utilidad el uso del foro de Papyrus. En https://www.eclipse.org/forums/index.php?t=showposts&id=198230 se pueden ver los mensajes con los que participamos.

Dentro de la migración, algunos problemas se pueden destacar como de mayor dificultad:

- Cambios en *namespaces*. Hubo grandes cambios en ese sentido y fue muy difícil encontrar los nuevos *namespaces* ya que no está claramente documentado en ningún lado. Sobretodo tuvimos dificultad con los cambios de nombres de los puntos de extensión.
- Algunos de los cambios realizados para el plugin SoaML no estaban documentados por lo que generó dudas de si eran cosas que habían olvidado documentar o que estaban diferentes debido a la migración de Papyrus. Esto provocó que se tuviera que hacer una comparación exhaustiva de los proyectos para no pasar nada por alto.
- La aparición de diagramas en el *wizard* dependiendo del lenguaje seleccionado fue algo que nos llevó bastante tiempo lograr y no era tan sencillo como en la versión de Papyrus utilizada en SoaML.
- Lograr que se mostraran los menús de creación de diagramas también fue un gran obstáculo que necesitó de bastante dedicación.

También indentificamos un bug en el modelado de Collaborations que, por definición, deben ser representadas con líneas punteadas y Papyrus las estaba mostrando con línea contínua. Esto fue corregido para los diagramas que contienen elementos que extienden al elemento Collaborations de UML, estos son Diagrama de Arquitectura de Servicios y Diagrama de Contrato de Servicios, dentro de los plugins de modelado SoaML y SoaML+QoS.

4.5.2. Tiempo de Desarrollo

La nueva versión de Eclipse (Luna) aumentó considerablemente el uso de CPU y de memoria por lo que el desarrollo de los plugins fue bastante tedioso, en particular a la hora de usar el *debugger*. Además, al ser una versión recién liberada, todavía tiene ciertos *bugs* que si bien no son tan importantes, complican su uso.

También se gastó bastante tiempo intentando solucionar *warnings* que luego nos dimos cuenta que también se daban en las versiones anteriores de los plugins y no tenían importancia.

4.5.3. Agregar nuevos elementos UML a diagramas

Para la implementación de QoS Value en los diagramas de Participant Class, Participant Component y Service Contract era necesario el elemento *Instance Specification* pero, dado que éstos fueron implementados a partir del plugin del diagrama de estructura compuesta, no se contaba con dicho elemento. Se tuvo que copiar del plugin del diagrama de clases todas las clases y códigos relacionados a *Instance Specification* y todo lo asociado a ésta, lo cual requirió bastante tiempo de investigación e implementación.

4.5.4. Uso de QoSValue

Una limitación que encontramos en el uso de UML y por consiguiente de Papyrus, fue que no es posible agregar dependencias a operaciones. Esto limita las posibilidades de definir características de calidad diferentes para distintas operaciones dentro de un servicio. Por ejemplo, uno podría estar interesado en definir un tiempo de respuesta mayor para una operación que tiene una lógica compleja al que tiene una operación que sólo retorna información.

4.5.5. Implementación de WS-Policy para distintos servidores

En la implementación del generador de código, una de las partes más complicadas fue lograr hacer funcionar las políticas de calidad en todos los servidores utilizados con las distintas tecnologías. JBoss y Tomcat tienen grandes diferencias en cuanto a configuración e incluso en la forma en que procesan las políticas. No resultó fácil encontrar documentación que indicara paso a paso cómo configurar e implementar WS-Policy con dichos servidores, lo cual nos llevó a recurrir a distintos foros y a tener que realizar una cantidad considerable de pruebas hasta conseguir que funcionaran.

5. Verificación del plugin SoaML Toolkit

En este capítulo se resumen los resultados obtenidos de los casos de prueba ejecutados sobre las distintas versiones del plugin, incluyendo el modelador y el generador de código.

En la sección 5.1 se exponen y analizan los resultados obtenidos en lo referente a la ejecución de los casos de uso correspondientes al modelado de QoS. En la sección 5.2 se muestran los resultados obtenidos luego de la ejecución de pruebas para el modelado SoaML+QoS y en la sección 5.3 se describe la verificación realizada sobre el modelado de SoaML. En la sección 5.4 se exponen y analizan los resultados obtenidos de los casos de prueba ejecutados sobre el generador de código. Finalmente, en la sección 5.5 se realiza un resumen de la verificación realizada en el cual se cuantifican los casos de prueba realizados, corregidos y restantes.

5.1.Reporte de los casos de prueba de modelado QoS

En esta sección se exponen los resultados obtenidos en las pruebas ejecutadas sobre el modelado de taxonomía QoS.

5.1.1. Crear Diagrama de Características QoS

Esta funcionalidad permite crear un diagrama de tipo *QoS Characteristic* y modelar en éste las distintas características de calidad, las dimensiones que cuantifican dichas características, así como también la categoría a la cual pertenece y las asociaciones que pueden haber entre distintas características.

La validación se realizó creando proyectos QoS y proyectos SoaML+QoS, agregando el diagrama de tipo QoS Characteristic y agregando al lienzo los diferentes elementos disponibles en la paleta.

Se corrieron un total de 18 casos de prueba para el caso de uso. En la primera iteración ya se obtuvieron los resultados esperados.

5.1.2. Editar Diagrama de Características QoS

Esta funcionalidad permite editar un diagrama de tipo QoS Characteristic existente agregando, borrando y modificando elementos de éste.

La validación se realizó partiendo de proyectos QoS y SoaML+QoS con diagramas de características de calidad existentes y se probó de agregar, eliminar y editar todos los elementos disponibles en la paleta.

Se corrieron un total de 18 casos de prueba para el caso de uso. En la primera iteración ya se obtuvieron los resultados esperados.

5.1.3. Eliminar diagrama de Características QoS

Esta funcionalidad permite eliminar un diagrama de tipo QoS Characteristic existente y los elementos contenidos en éste.

La validación se realizó partiendo de proyectos QoS y SoaML+QoS con diagramas de características de calidad existentes y eliminando uno a uno sus elementos. Finalmente se probó eliminar el diagrama del proyecto.

Se corrieron un total de 4 casos de prueba para el caso de uso. En la primera iteración ya se obtuvieron los resultados esperados.

5.1.4. Resumen de la verificación de los casos de prueba de modelado QoS

Los casos de prueba ejecutados para verificar el correcto funcionamiento del modelado QoS dieron resultados muy favorables. Como se puede ver en la Figura 56 se obtuvo la salida esperada en los 40 casos de prueba ejecutados. Ya que no se encontraron errores en la primera iteración no fue necesario realizar cambios en la implementación del modelado QoS y por lo tanto tampoco fue necesaria una segunda iteración.

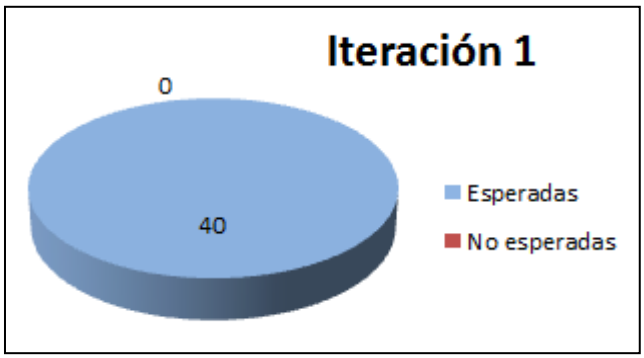

Figura 56. Resumen de verificación de casos de prueba de modelado QoS

5.2.Reporte de los casos de prueba de modelado SoaML+QoS

En esta sección se exponen los resultados obtenidos en las pruebas ejecutadas sobre el modelado de SoaML con QoS.

5.2.1. Crear diagrama de Contrato de Servicios con QoS

Esta funcionalidad permite crear un diagrama de tipo *Service Contract with QoS* y modelar en éste elementos tales como contratos de servicios, roles y usos de contratos de servicios, agregar asociaciones entre roles y especificar los roles que cumplen ciertos roles en contratos de servicios asociados. También es posible agregar valores de características de calidad de servicios, y asociaciones entre éstos y los contratos de servicios.

La validación se realizó partiendo de proyectos SoaML+QoS con características de calidad definidas, agregando el diagrama de tipo Service Contract with QoS y agregando al lienzo los diferentes elementos disponibles en la paleta.

Se realizó una iteración en donde se ejecutaron 6 casos de prueba para los cuales se obtuvieron los resultados esperados. Se observó un comportamiento no favorable en Papyrus cuando se define un elemento QoSValue arrastrando el elemento QoSCharacteristic y seleccionando una QoSDimension: los Slots definidos en la QoSDimension seleccionada no se ven dentro del elemento QoSValue en el diagrama pero sí se ven definidos dentro del elemento en el árbol del modelo. Para que aparezcan dentro del elemento QoSValue en el diagrama es necesario arrastrar los Slots desde el árbol del modelo.

5.2.2. Editar diagrama de Contrato de Servicios con QoS

Esta funcionalidad permite editar un diagrama de tipo Service Contract with QoS existente agregando, borrando y modificando elementos de éste.

La validación se realizó partiendo de proyectos SoaML+QoS con características de calidad definidas y con diagramas de contratos de servicio con características de calidad existentes y se probó de agregar, eliminar y editar todos los elementos disponibles en la paleta.

Se realizó una iteración en donde se ejecutaron 11 casos de prueba para los cuales se obtuvieron los resultados esperados.

5.2.3. Eliminar diagrama de Contrato de Servicios con QoS

Esta funcionalidad permite eliminar un diagrama de tipo Service Contract with QoS existente y los elementos contenidos en éste.

La validación se realizó partiendo de proyectos SoaML+QoS con características de calidad definidas y con diagramas de contratos de servicio con características de calidad existentes y eliminando uno a uno sus elementos. Finalmente se probó eliminar el diagrama del proyecto.

Se realizó una iteración en donde se ejecutaron 4 casos de prueba para los cuales se obtuvieron los resultados esperados.

5.2.4. Importar modelo de Características QoS

Esta funcionalidad permite importar un modelo de características QoS a un modelo SoaML+QoS, realizando una copia en profundidad de todos los elementos del modelo origen al modelo destino.

La validación se realizó utilizando un modelo SoaML+QoS e importando distintos tipos de modelos, verificando que sólo importara los modelos QoS.

En la primera iteración se realizaron 3 casos de prueba y se obtuvo un resultado no esperado. Al seleccionar un modelo que no fuera QoS se esperaba que se desplegara un mensaje de error, no permitiendo la importación. Sin embargo, se seleccionó un modelo SoaML+QoS y se importaron todos los elementos del modelo. Para la segunda iteración se volvieron a ejecutar los mismos casos de prueba, habiéndose corregido ya el error, y se obtuvieron los resultados esperados.

5.2.5. Importar modelo SoaML

Esta funcionalidad permite importar un modelo SoaML a un modelo SoaML+QoS, realizando una copia en profundidad de todos los elementos del modelo origen al modelo destino.

La validación se realizó utilizando un modelo SoaML+QoS e importando distintos tipos de modelos, verificando que sólo importara los modelos SoaML.

La verificación se realizó en la segunda iteración ya que fue un caso de uso definido luego de la primera iteración. Se realizaron 3 casos de prueba y se obtuvieron los resultados esperados.

5.2.6. Crear Diagrama de Participantes como Clases con QoS

Esta funcionalidad permite crear un diagrama de tipo *Participant Class with QoS* y modelar en éste participantes, sus puertos y sus comportamientos asociados. También es posible agregar valores de características de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a los puertos.

La validación se realizó partiendo de proyectos SoaML+QoS con características de calidad definidas, agregando el diagrama de tipo Participant Class with QoS y agregando al lienzo los diferentes elementos disponibles en la paleta.

Se realizó una iteración en donde se ejecutaron 7 casos de prueba para los cuales se obtuvieron los resultados esperados. Se observó el mismo comportamiento con la definición del QoS Value que el declarado en el caso de uso Crear diagrama de Contrato de Servicios con QoS.

5.2.7. Editar Diagrama de Participantes como Clases con QoS

Esta funcionalidad permite editar un diagrama de tipo Participant Class with QoS existente agregando, borrando y modificando elementos de éste.

La validación se realizó partiendo de proyectos SoaML+QoS con características de calidad definidas y con diagramas de participantes como clases con características de calidad existentes y se probó de agregar, eliminar y editar todos los elementos disponibles en la paleta.

Se realizó una iteración en donde se ejecutaron 14 casos de prueba para los cuales se obtuvieron los resultados esperados. Se observó el mismo comportamiento con la definición del QoS Value que el declarado en el caso de uso anterior.

5.2.8. Eliminar diagrama de Participantes como Clases con QoS

Esta funcionalidad permite eliminar un diagrama de tipo Participant Class with QoS existente y los elementos contenidos en éste.

La validación se realizó partiendo de proyectos SoaML+QoS con características de calidad definidas y con diagramas de participantes como clases con características de calidad existentes y eliminando uno a uno sus elementos. Finalmente se probó eliminar el diagrama del proyecto.

Se realizó una iteración en donde se ejecutaron 4 casos de prueba para los cuales se obtuvieron los resultados esperados.

5.2.9. Crear Diagrama de Participantes como Componentes con QoS

Esta funcionalidad permite crear un diagrama de tipo *Participant Component with QoS* y modelar en éste componentes del tipo Participant y dentro de ellos elementos tales como proveedores de servicios, puertos, asociaciones y canales de servicio entre los puertos, capacidades y sus dependencias con los participantes existentes en el diagrama, etc. También es posible agregar valores de características de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios.

La validación se realizó partiendo de proyectos SoaML+QoS con características de calidad definidas, agregando el diagrama de tipo Participant Component with QoS y agregando al lienzo los diferentes elementos disponibles en la paleta.

Se realizó una iteración en donde se ejecutaron 7 casos de prueba para los cuales se obtuvieron los resultados esperados. Se observó el mismo comportamiento con la definición del QoS Value que el declarado en el caso de uso Crear diagrama de Participantes como Clases con QoS.

5.2.10. Editar Diagrama de Participantes como Componentes con QoS

Esta funcionalidad permite editar un diagrama de tipo Participant Component with QoS existente agregando, borrando y modificando elementos de éste.

La validación se realizó partiendo de proyectos SoaML+QoS con características de calidad definidas y con diagramas de participantes como componentes con características de calidad existentes y se probó de agregar, eliminar y editar todos los elementos disponibles en la paleta.

Se realizó una iteración en donde se ejecutaron 14 casos de prueba para los cuales se obtuvieron los resultados esperados. Se observó el mismo comportamiento con la definición del QoS Value que el declarado en el caso de uso anterior.

5.2.11. Eliminar Diagrama de Participantes como Componentes con QoS

Esta funcionalidad permite eliminar un diagrama de tipo Participant Component with QoS existente y los elementos contenidos en éste.

La validación se realizó partiendo de proyectos SoaML+QoS con características de calidad definidas y con diagramas de participantes como componentes con características de calidad existentes y eliminando uno a uno sus elementos. Finalmente se probó eliminar el diagrama del proyecto.

Se realizó una iteración en donde se ejecutaron 4 casos de prueba para los cuales se obtuvieron los resultados esperados.

5.2.12. Importar modelo desde archivo con formato XMI

Esta funcionalidad permite importar un modelo SoaML o SoaML+QoS desde un archivo XML con formato XMI, de forma de que se generen en el árbol del modelo los elementos contenidos en el archivo importado con sus estereotipos asociados.

La validación se realizó importando diferentes archivos XMI de modelos SoaML+QoS con distintos diagramas. La validación de importación de archivos XMI de modelos SoaML se realizó como parte de la verificación realizada en la sección 5.3.

Se realizó una iteración en donde se ejecutaron 5 casos de prueba para los cuales se obtuvieron los resultados esperados.

5.2.13. Exportar modelo desde archivo con formato XMI

Esta funcionalidad permite exportar un modelo SoaML o SoaML+QoS con sus elementos y estereotipos asociados desde el editor a un archivo XML con formato XMI 2.1.

La validación se realizó exportando diferentes modelos SoaML+QoS con distintos diagramas. La validación de exportación de modelos SoaML se realizó como parte de la verificación realizada en la sección 5.3.

Se realizó una iteración en donde se ejecutaron 5 casos de prueba para los cuales se obtuvieron los resultados esperados.

5.2.14. Resumen de la verificación de los casos de prueba de modelado SoaML+QoS

Para la verificación del modelado SoaML+QoS se realizaron dos iteraciones. En la primera se ejecutaron 84 casos de prueba correspondientes a todos los casos de uso de modelado SoaML+QoS. Allí se detectó un error en la importación de modelos QoS por lo que fue necesaria una segunda iteración, luego de corregir el error encontrado. En la segunda iteración se ejecutaron 3 casos de prueba para verificar que se corrigió el error encontrado en la primera iteración, obteniendo el resultado esperado. Además, para la segunda iteración se agregaron 3 casos de pruebas para el caso de uso importar modelo SoaML que se definió luego de la primera iteración. En la Figura 57 se pueden ver dos gráficas que ilustran los resultados obtenidos en las dos iteraciones.

Figura 57. Resumen de verificación de casos de prueba de modelado SoaML+QoS

5.3.Reporte de los casos de prueba de modelado SoaML actualizado

Se corrieron un total de 655 casos de prueba diseñados por el equipo que implementó el plugin SoaML y se obtuvieron los resultados esperados, como se puede ver en la Figura 58. Cabe aclarar que los problemas detectados en la versión anterior sobre la falta de validación en las asociaciones entre elementos y la asignación de tipos se mantienen. Por último, es necesario destacar que con la nueva versión de Papyrus hubo un cambio importante en el uso del editor. Anteriormente, cuando uno utilizaba la tecla Suprimir para eliminar un elemento del lienzo o del modelo, ésta estaba asociada con el evento Ocultar. Es por esta razón que en los casos de prueba se escribió que al eliminar un elemento del lienzo, éste seguía existiendo en el explorador del modelo, porque en lugar de eliminar el elemento definitivamente, se estaba ocultando. Ahora dicha tecla está asociada efectivamente al evento Eliminar, por lo que los casos de prueba en donde se menciona eliminar un elemento del lienzo, éste también es eliminado del modelo. En resumen, el comportamiento es el siguiente:

- Si se elimina un diagrama que contiene elementos, éstos se mantienen en la vista del modelo aunque no estén referenciados en otro lado.
- Si se elimina un elemento de un diagrama, éste es eliminado también de la vista del modelo y de todos los diagramas en los que estuviera o fuera referenciado.

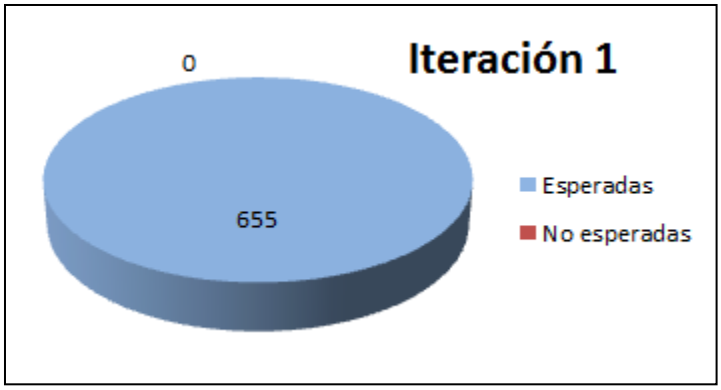

Figura 58. Resumen de verificación de casos de prueba de modelado SoaML

5.4.Reporte de los casos de prueba del generador

En esta sección se exponen los resultados obtenidos en las pruebas ejecutadas sobre el generador de código.

5.4.1. Generar proyectos Java a partir de modelo SoaML+QoS y SoaML

Esta funcionalidad permite generar proyectos Java a partir de modelos SoaML+QoS y SoaML. El modelo puede estar en formato XMI o en formato UML, y se permite generar distintos tipos de proyectos que exponen los servicios web especificados en el modelo y los clientes de estos servicios. Para modelos SoaML+QoS también se permite generar contratos WS-Agreement a partir de las características de calidad asociadas a los contratos especificados en el modelo, y permite generar archivos WSDL con la descripción de los servicios, y con políticas WS-Policy a partir de las características de calidad asociadas a los puertos de servicio en el modelo.

La validación se realizó generando distintos tipos de proyectos Java (Dynamic Web Project y EJB) con distintos frameworks (JAX-WS RI y JAX-WS+Spring) a partir de diferentes modelos SoaML y SoaML+QoS. Para los modelos SoaML+QoS se verificó la correcta generación de los documentos WS-Agreement y de los archivos WSDL con WS-Policy y WS-Security. También se verificó el correcto funcionamiento de los servicios en los distintos servidores (JBoss y Tomcat) invocándolos desde los clientes generados. Los servicios que incluían control de acceso se probaron invocándolos con distintas combinaciones de usuarios y contraseñas correctas e incorrectas.

En la primera iteración se ejecutaron 7 casos de prueba entre los que se obtuvieron 2 resultados no esperados. Se detectó un error en la generación de documentos WS-Agreement, en donde bajo ciertas condiciones se generan documentos asociados a contratos en los que el participante no posee ningún rol. También se detectó una posible mejora en la configuración de log4j: los clientes se configuraban para imprimir los logs en la ruta “C:\output.log”, lo cual puede traer problemas si el cliente no tiene permisos de escritura sobre esa ruta. Se corrigió este error modificando la generación del cliente para que el archivo de log se genere dentro de la carpeta donde se encuentra instalado el cliente.

En la segunda iteración se ejecutaron 12 casos de prueba para los que se obtuvieron los resultados esperados. Se agregaron 5 casos de prueba respecto a la iteración anterior para probar la generación de proyectos con JAX-WS RI para Tomcat con características de calidad y además la comunicación entre

clientes y servicios localizados en distintas máquinas. El único inconveniente que se detectó es cuando el proyecto servidor es generado para JAX-WS RI y Tomcat. Cuando un servicio web es invocado el servidor ejecuta correctamente el servicio, pero imprime la siguiente advertencia:

"*WARNING: onComplete() failed for listener of type [org.apache.catalina.core.AsyncListenerWrapper]*

java.lang.IllegalStateException: It is illegal to call getRequest() after complete() or any of the dispatch() methods has been called"

En la Figura 59 se ilustran los resultados obtenidos en la ejecución de casos de prueba del generador en las dos iteraciones.

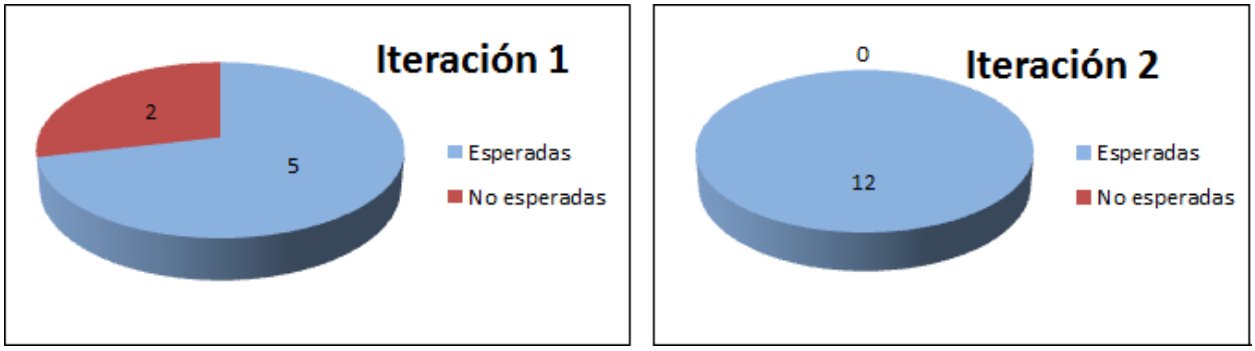

Figura 59. Resumen de verificación de casos de prueba del generador

5.5.Resumen de la verificación realizada

Se realizaron dos iteraciones para los casos de pruebas definidos para los plugins de modelado y generación de código. En la primera iteración se ejecutaron los casos de pruebas de los casos de Uso del plugin de modelado QoS, para el de SoaML con QoS y para el de SoaML actualizado. Además se ejecutaron los casos de pruebas del generador de código. En la segunda iteración se ejecutaron los casos de pruebas de dos casos de uso del plugin de modelado SoaML con QoS y los casos de pruebas del generador de código.

En la Tabla 9 se presenta un resumen de la cantidad de casos de pruebas ejecutados para cada plugin, el total de casos de pruebas con la cantidad de errores encontrados y corregidos para cada iteración. En la Figura 60 se ilustran los resultados totales para cada iteración.

Tabla 9. Resumen de ejecución de casos de prueba

	Iteración 1	**Iteración 2**
Casos de prueba de modelado QoS	40	0
Casos de prueba de modelado SoaML+QoS	84	6
Casos de prueba de modelado SoaML	655	0
Casos de prueba de generación de código	7	12
Total de casos ejecutados	**786**	**18**
Total de casos con resultados esperados	**783**	**18**
Total de casos con resultados no esperados	**3**	**0**
Errores solucionados	**3**	**0**

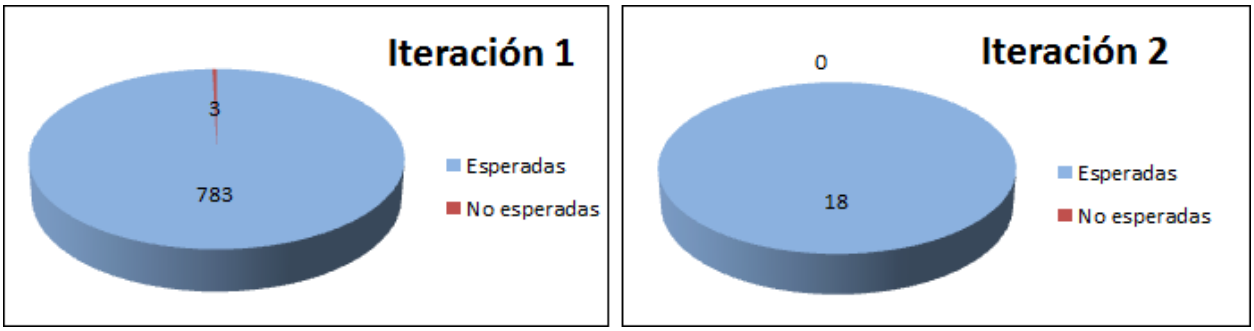

Figura 60. Resumen de la verificación

En los plugins de modelado no se encontraron grandes problemas. En el plugin de modelado de SoaML+QoS se corrigió el error encontrado en el caso de uso de Importación de modelo de Características QoS.

Como se comentó en la sección 5.3, en el modelado SoaML se mantienen los errores con respecto a asociaciones indebidas de elementos o a definiciones de tipos incorrectas. A su vez se observa comportamientos no favorables de Papyrus que no entran en el alcance de este proyecto corregirlas. También es importante aclarar que en el modelado QoS y SoaML+QoS no se validó que las asociaciones entre elementos sean conceptualmente correctas y consistentes con los perfiles SoaML y QoS, debido a que escapaba al alcance del proyecto.

En el plugin de generación de código se corrigió el error en la generación del documento WS-Agreement y se implementó una mejora sobre el archivo log generado en el cliente.

6. Caso de estudio

El caso de estudio utilizado para probar la herramienta desarrollada fue realizado en base al proceso de negocio (PN) “Admisión y Registro de Paciente para Cirugía Mayor Ambulatoria (CMA)” [39], el cual fue utilizado por la tutora, en el marco del framework MINERVA [24] y que fue también el utilizado por los grupos que realizaron los plugins SoaML y SoaML2Code. El objetivo de este caso de estudio es realizar un modelo de QoS utilizando el plugin de QoS, importarlo en el modelo de servicios SoaML ya existente de proyectos anteriores y agregar características de QoS al modelo SoaML utilizando el plugin SoaML+QoS. A partir de este modelo generar el código correspondiente.

El modelo de características de calidad QoS que será agregado a los diagramas SoaML corresponde a las características definidas en el Modelo de Medidas de Ejecución de Procesos de Negocio BPEMM (Business Process Execution Measurement Model) [23] y que define entre otras medidas, una variedad de características y medidas de calidad para servicios.

El material otorgado para la realización del caso de estudio consta de un archivo XMI correspondiente al modelo SoaML del caso de estudio, proporcionado por el grupo que implementó el plugin SoaML2Code, el cual fue importado para crear el proyecto SoaML en Papyrus.

6.1.Presentación del caso de estudio

Una breve descripción de la realidad que se representa en este caso de estudio se obtiene a partir del proyecto del plugin SoaML2Code. El modelo del proceso de negocio y el modelo SoaML de implementación de dicho proceso con servicios son dados.

El proceso de negocio “Admisión y Registro de Paciente para Cirugía Mayor Ambulatoria (CMA)”, es un proceso correspondiente con el Hospital General de Ciudad Real (España). En dicho proceso se distinguen tres participantes: el Hospital Público Local, el Paciente y el Registro Central de Salud. La interacción entre ellos se da de la siguiente forma:

- Un paciente solicita programar la CMA.
- La secretaría del Hospital solicita el registro médico del paciente al Registro Central de Salud.
- El paciente recibe la fecha y hora solicitada.

En la Figura 61 se muestra el modelo en la notación BPMN 2.0 (Business Process Model and Notation) [40] del OMG donde se describe en forma completa la realidad del PN, de la cual se extrae el fragmento analizado.

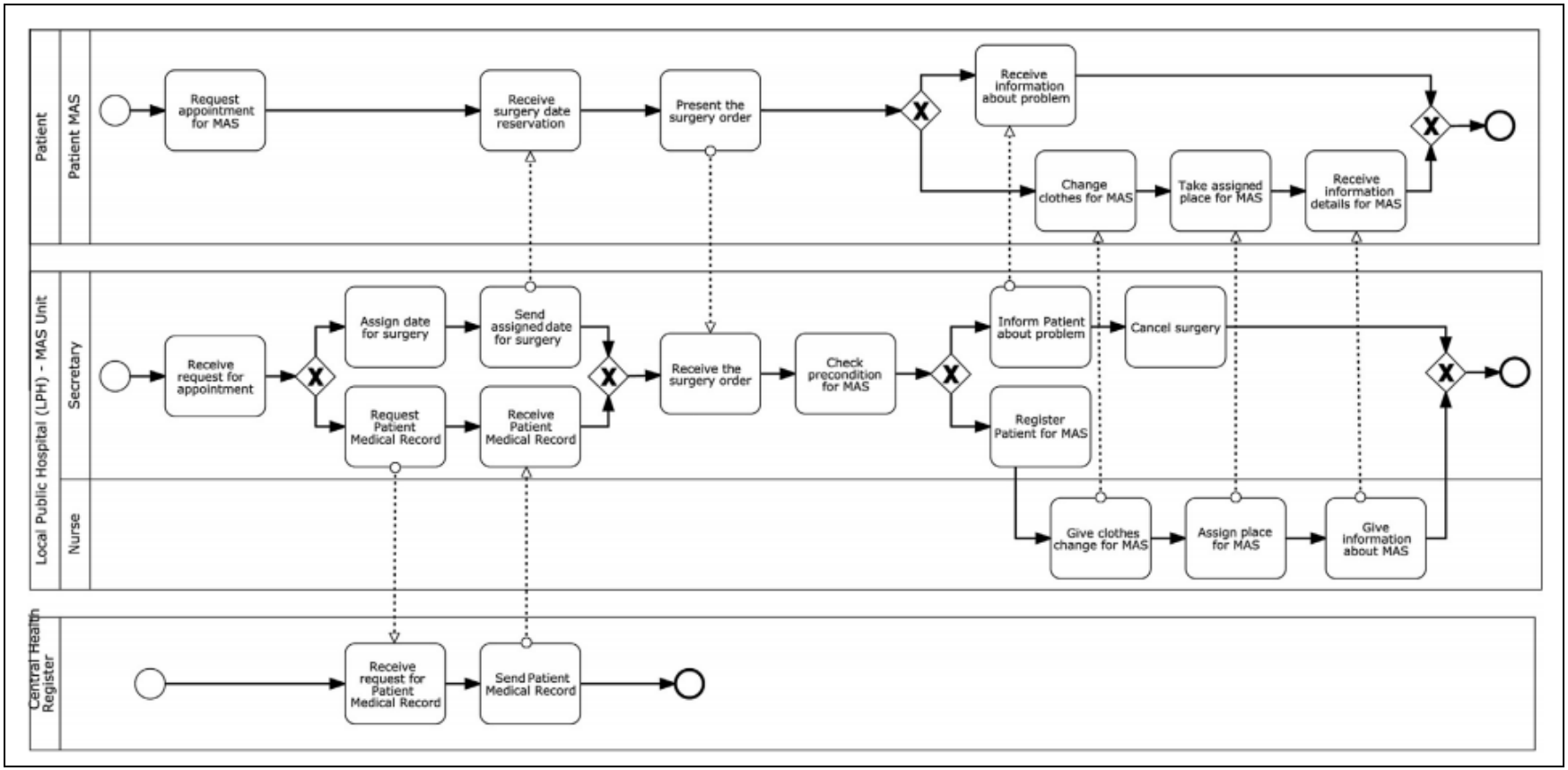

Figura 61. Modelo BPMN "Admisión y Registro de Paciente para Cirugía Mayor Ambulatoria" (CMA) [39]

Como se mencionó anteriormente, el modelo SoaML de implementación del proceso con servicios fue obtenido de la documentación provista por el proyecto de grado SoaML2Code. En la Figura 62 se puede ver el Diagrama de Participantes como Clases del cual partimos, el cual contiene los tres participantes antes mencionados y los servicios provistos y requeridos por ellos, mediante puertos Service y Request.

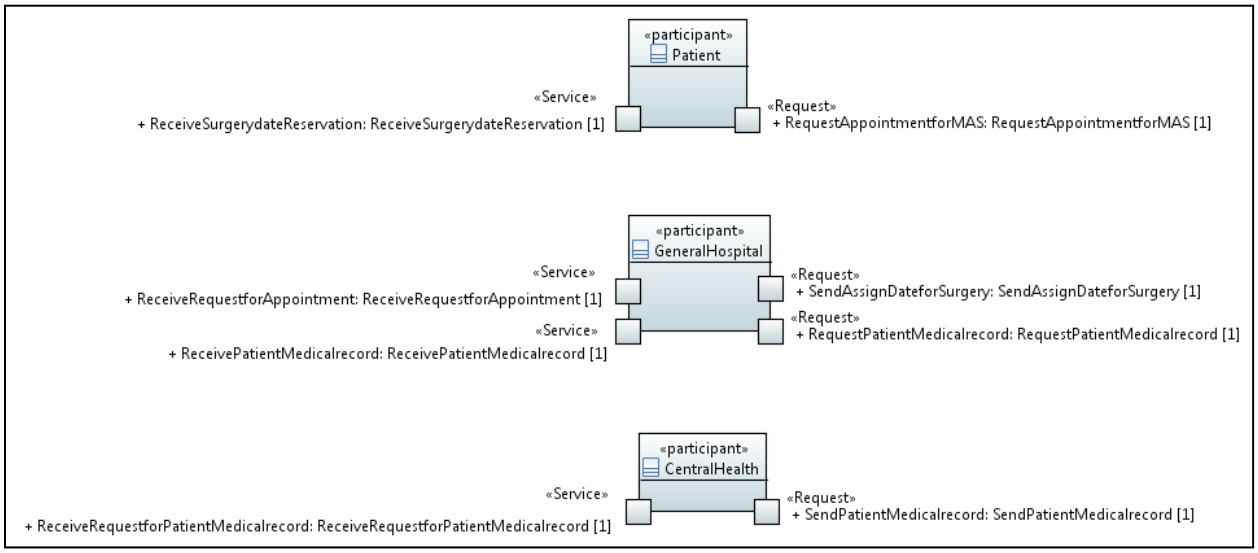

Figura 62. Diagrama de *Participants*

El participante *Patient* consume el servicio *ReceiveRequestForAppointment*, provisto por el participante GeneralHospital, para programar la CMA. Además provee el servicio *ReceiveSurgeryDateReservation* que es consumido por el *GeneralHospital* para enviar la fecha asignada.

El participante *GeneralHospital* consume el servicio *ReceiveRequestForPatientMedicalRecord*, provisto por el participante *CentralHealth*, para solicitar el registro médico del paciente. Además provee el servicio *ReceivePatientMedicalRecord* que es consumido por el *CentralHealth* para enviar la información solicitada.

El Diagrama de Arquitectura de Servicios se puede observar en la Figura 63, donde se muestran los distintos participantes con sus roles en la interacción con los servicios.

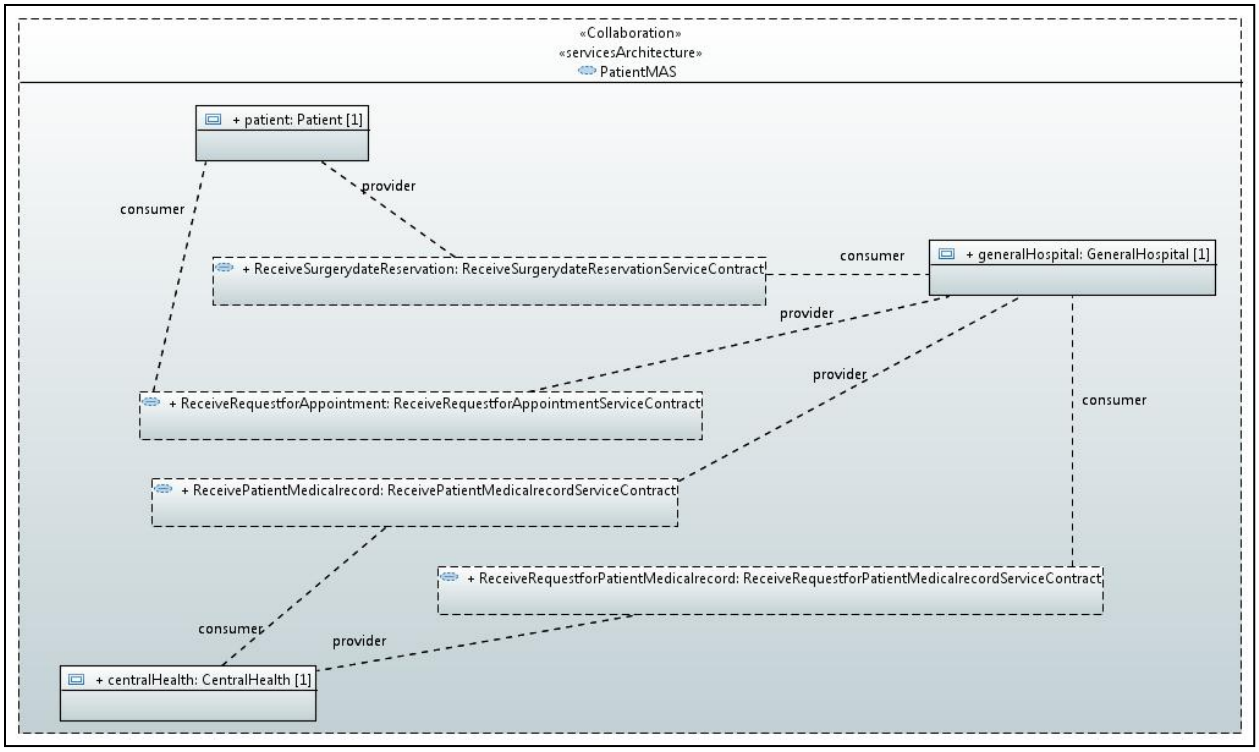

Figura 63. Diagrama de *ServicesArchitecture*

En la Figura 64 se puede ver el Diagrama de Mensajes, que incluye los mensajes intercambiados en los distintos servicios modelados.

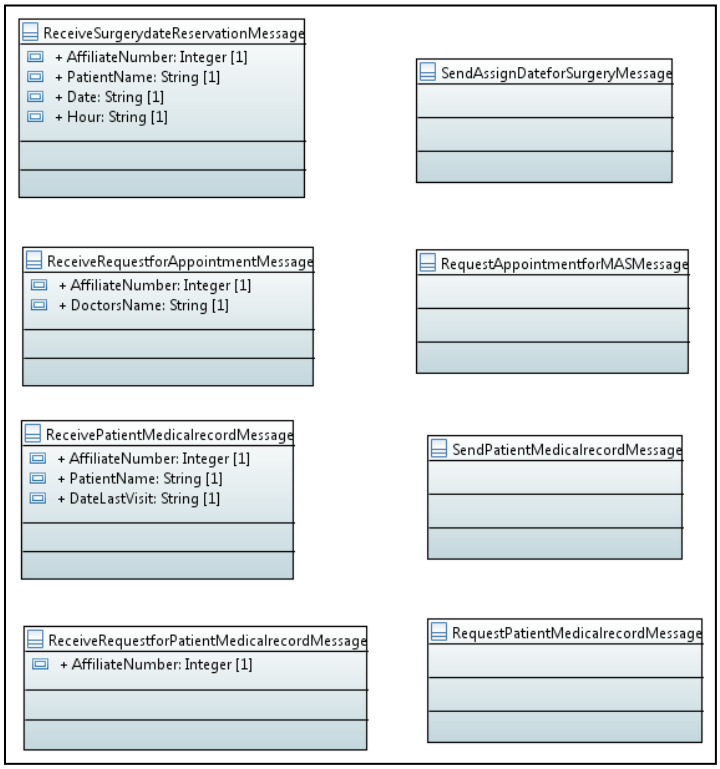

Figura 64. Diagrama de *Messages*

En la Figura 65 se observa el Diagrama de Interfaces en el cual se incluyen las interfaces que serán provistas y consumidas en los puertos de los participantes.

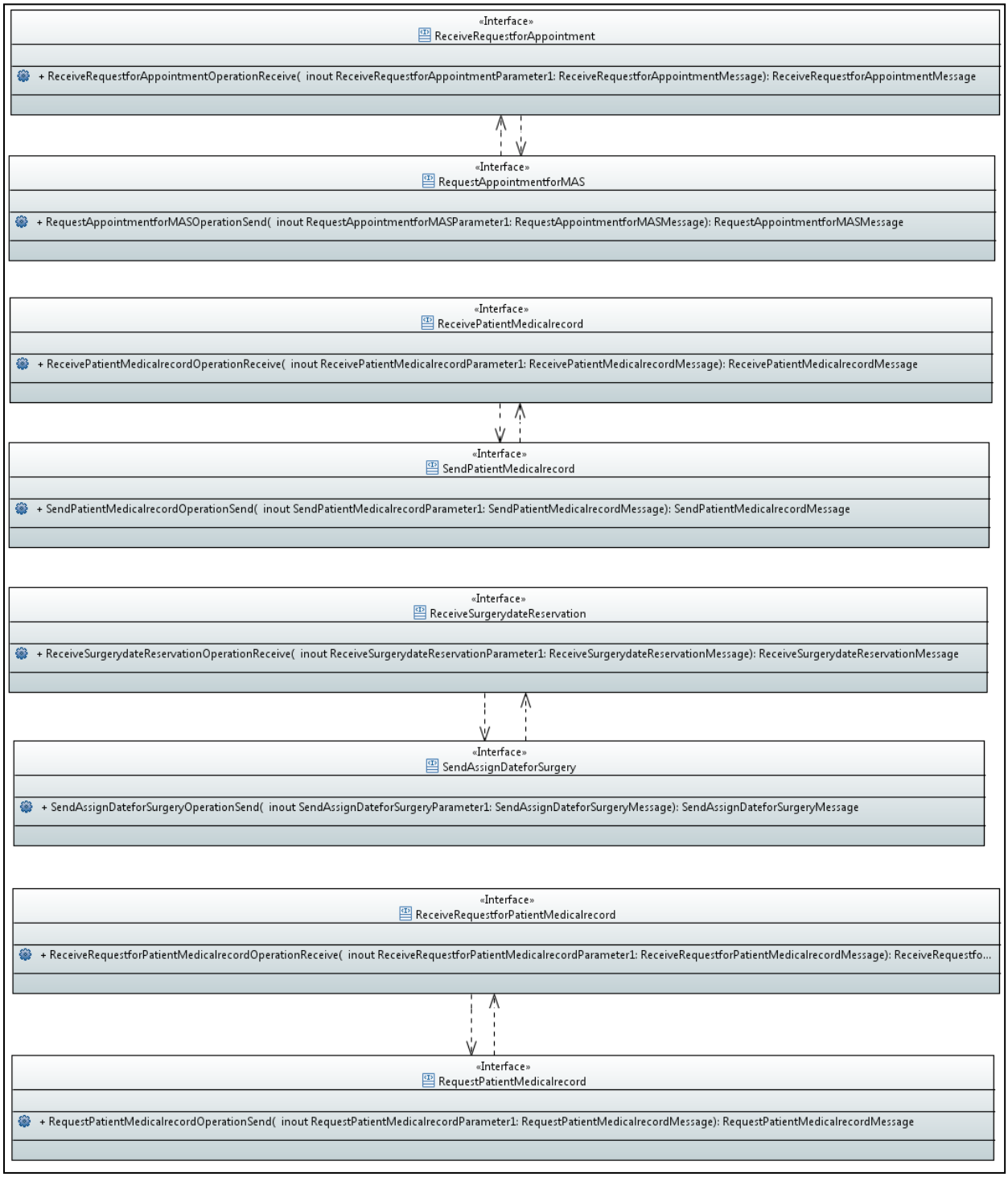

Figura 65. Diagrama de *ServicesInterface*

Por último, en la Figura 66 se puede observar el Diagrama de Contratos de Servicios en donde se especifican los contratos de servicio que definen los servicios identificados.

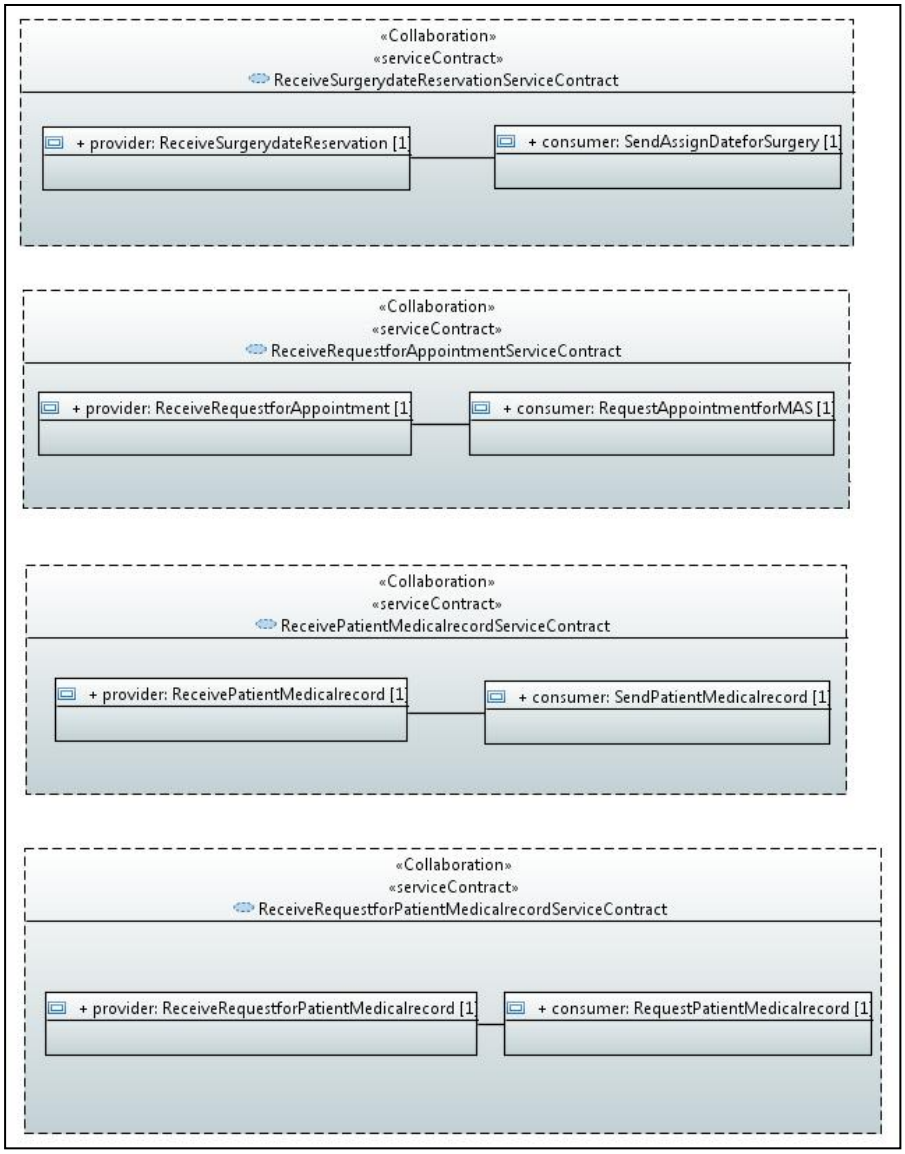

Figura 66. Diagrama de *ServiceContracts*

6.2.Realización del caso de estudio

6.2.1. Modelado

En esta sección se detalla la realización de la parte del modelado de QoS del caso de estudio, partiendo del modelo SoaML mostrado en la sección anterior y utilizando el editor construido en el proyecto de grado. En la Figura 67 se muestra una vista del plugin SoaML Toolkit incorporado como menú de Eclipse, mostrando en forma unificada los menús de los plugins anteriores.

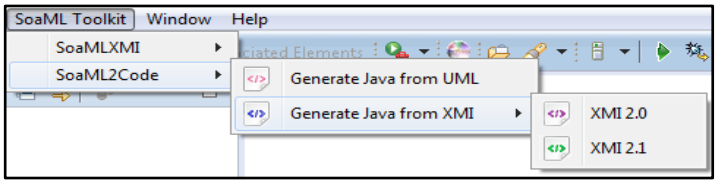

Figura 67. Vista del plugin SoaML Toolkit con el menú unificado

El modelado se podría dividir en dos partes, por un lado el modelado de una taxonomía de características de calidad en un proyecto Papyrus con lenguaje QoS, y por otro el modelado de una arquitectura SOA con atributos de calidad en un proyecto Papyrus con lenguaje SoaML+QoS.

6.2.1.1. Modelo de características de calidad QoS

Para modelar la taxonomía de características de calidad se crea un proyecto Papyrus, seleccionando como lenguaje QoS y luego el diagrama *QoS Characteristics Diagram*.

En la Figura 68 se muestra el primer paso para crear el proyecto Papyrus, en donde se debe indicar el nombre del proyecto. Luego se debe seleccionar el lenguaje del diagrama, el cual será en este caso QoS. Esto se muestra en la Figura 69.

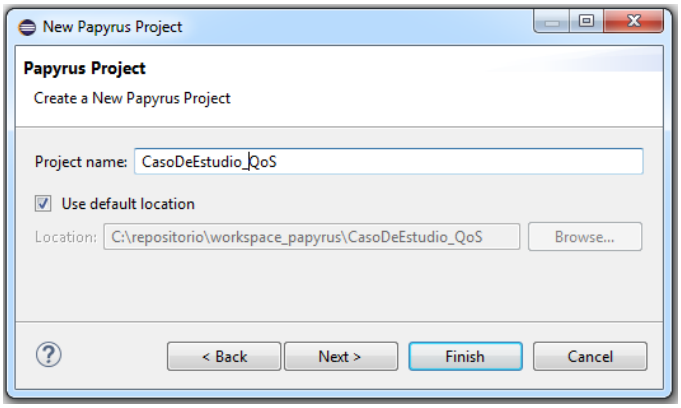

Figura 68. Primer paso para crear un proyecto Papryus con lenguaje QoS

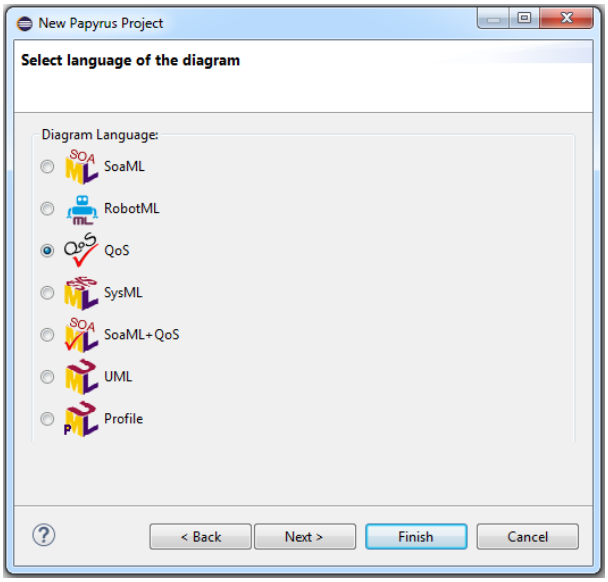

Figura 69. Segundo paso para crear un proyecto Papryus con lenguaje QoS

Por último, en el tercer paso se selecciona el tipo de diagrama que se desea generar, o sea, *QoS Characteristics Diagram*, como se muestra en la Figura 70.

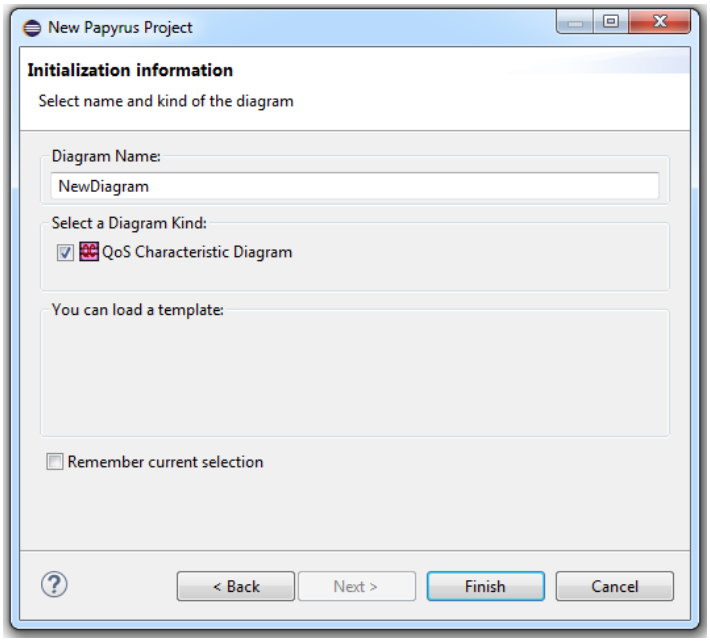

Figura 70. Tercer paso para crear un proyecto Papryus con lenguaje QoS

Luego de creado el diagrama, se crean los elementos definidos por el modelo de medidas de ejecución BPEMM que serán utilizados y otros elementos que consideramos relevantes. En este caso se crean tres elementos QoSCategory con los nombres “Performance”, “Dependability” y “Security”, como se muestra en la Figura 71.

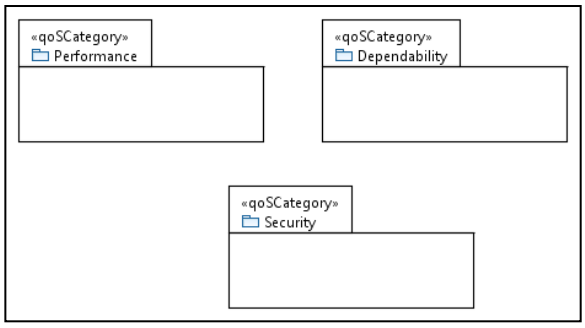

Figura 71. Caso de estudio - Modelado QoS - QoSCategory

Dentro de la categoría “Performance” se crean tres elementos QoSCharacteristic con los nombres “Service Response Time”, “Service Throughput” y “Service Capacity”, como se puede ver en la Figura 72. Estas características se corresponden con los objetivos especificados en las tablas 10, 11 y 12 respectivamente en [23].

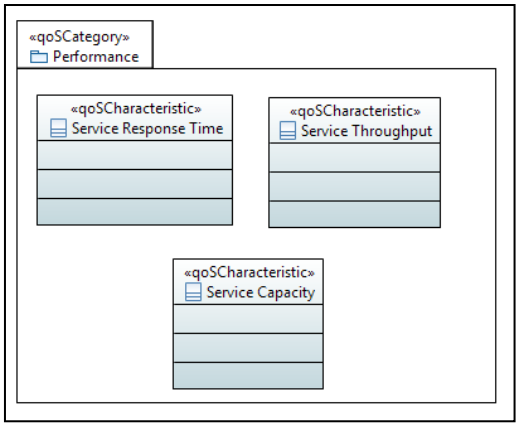

Figura 72. Caso de estudio - Modelado QoS - QoSCharacteristics

Dentro de la característica “Service Response Time” se crean tres elementos QoSDimension y se los llama “ServiceProcessingTime”, “ServiceLatencyTime” y “ServiceResponseTime”, los cuales se corresponden con las medidas M7, M8 y M9 de la Tabla 10 en [23]. Los tres son del tipo Integer y sus atributos fueron definidos de la siguiente forma: statisticalQualifier=maximum, direction=decreasing y unit=ms. Esto se puede observar en la Figura 73.

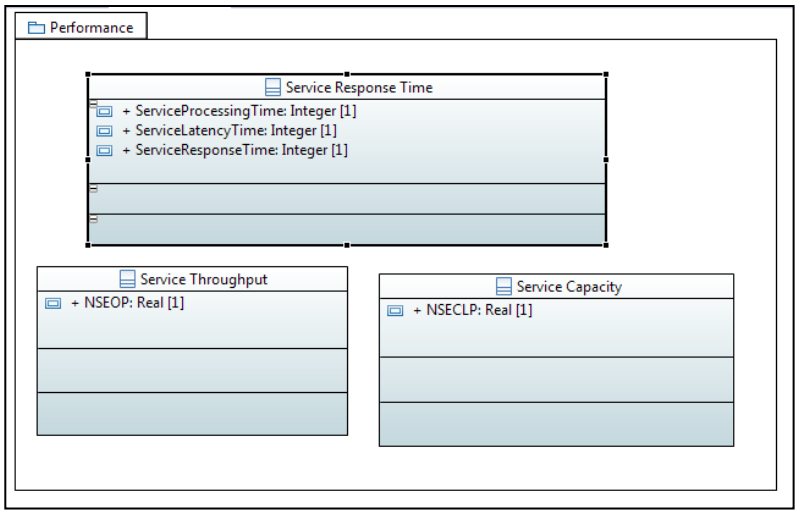

Figura 73. Caso de estudio - Modelado QoS - QoSDimensions

Para definir los atributos de una QoSDimension se selecciona la QoSDimension para ver las propiedades de ésta. Una vez ahí se va a la pestaña *Profile* y se setean los valores, como se puede ver en la Figura 74.

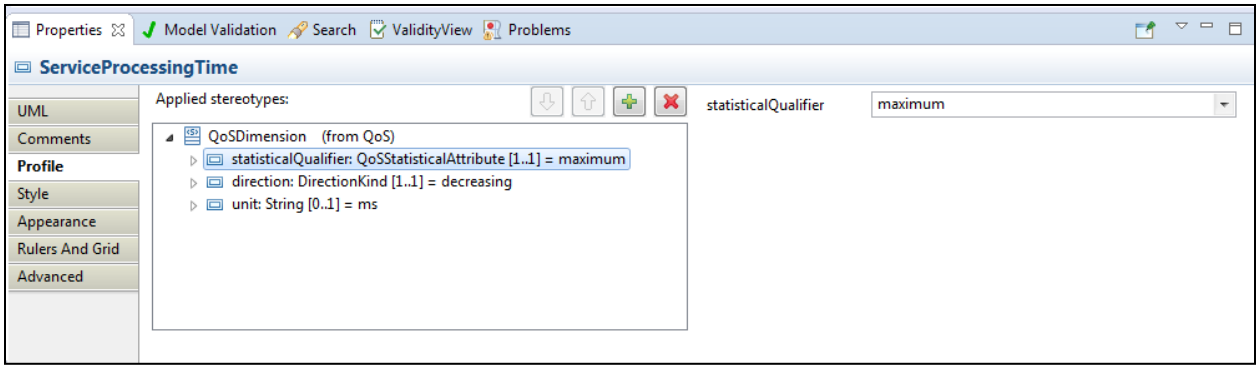

Figura 74. Caso de estudio - Modelado QoS - QoSDimension atributos

Continuando con la definición de medidas para la categoría Performance, dentro de la característica "Service Throughput" se crea un elemento QoSDimension con nombre "NSEOP" (*Number of service executions over a period of time*) que corresponde a la medida M1 de la Tabla 11 en [23]. Esta dimensión es del tipo Real y con atributos statisticalQualifier=minimum, direction=increasing y unit=executions / minute.

Dentro de la característica "Service Capacity" se crea un elemento QoSDimension con nombre "NSECLP" (*Number of completed services executions over a period of time under a given Service Response Time*) que corresponde a la medida M1 de la Tabla 12 en [23]. Esta dimensión es del tipo Real y con atributos statisticalQualifier=minimum, direction=increasing y unit=executions / minute.

A continuación se realizan los mismos pasos para el resto de las categorías definidas, comenzando por la categoría "Dependability", donde se crean dos elementos de tipo QoSCharacteristic con los nombres "Availability" y "Reliability". Estas características se corresponden con los objetivos planteados en la Tabla 13 en [23]. Dentro de la primera se crea un elemento QoSDimension con nombre "ServiceAvailability" correspondiente al indicador M3 de la misma tabla, la cual es del tipo Real y con atributos statisticalQualifier=minimum, direction=increasing y unit=%. Dentro de la segunda se crea un elemento QoSDimension con nombre "ServiceReliability" correspondiente a la medida M2, la cual es del tipo Real y con atributos statisticalQualifier=minimum, direction=increasing y unit=%.

Finalmente, dentro de la categoría "Security" se crea un elemento QoSCharacteristic con nombre "Access Control" y dentro de ésta se crea un elemento QoSDimension con nombre "Policy" del tipo String y con atributos statisticalQualifier=undefined, direction=undefined y unit="".

El diagrama final del modelo de QoS recién mencionado se muestra en la Figura 75.

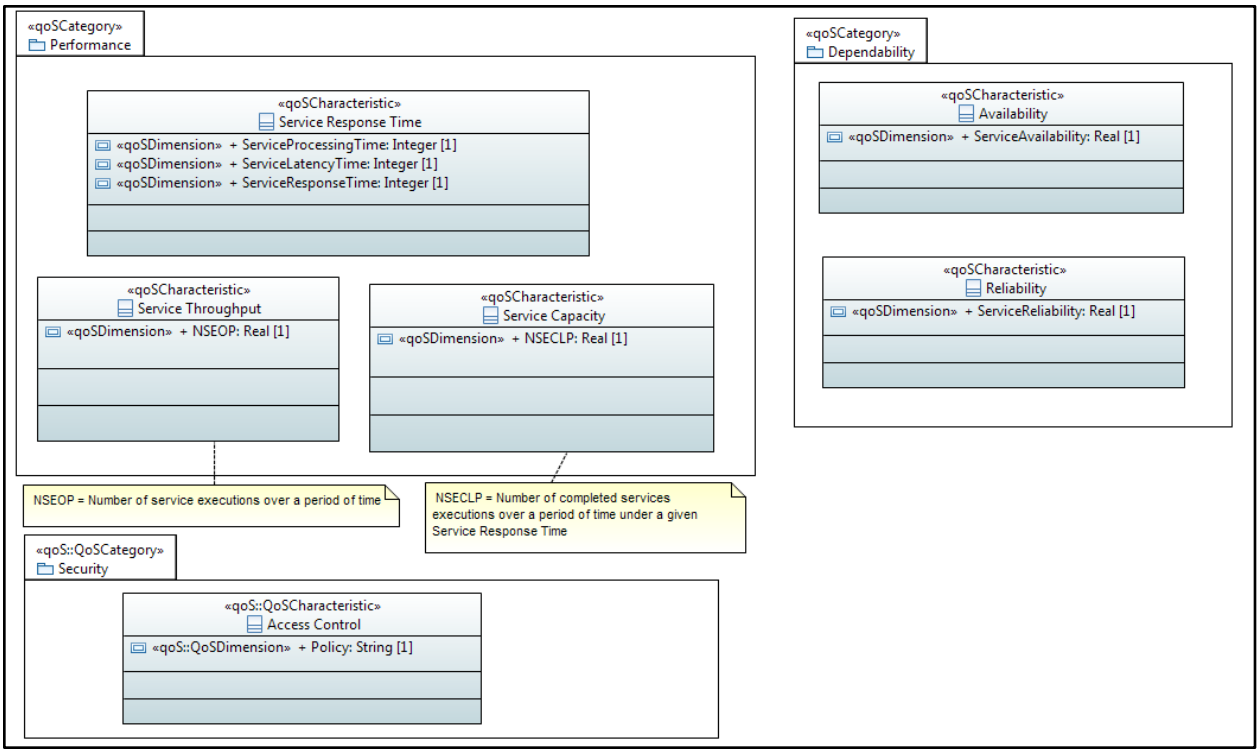

Figura 75. Diagrama QoS

6.2.1.2. *Modelo SoaML con QoS*

Para modelar la arquitectura SOA con atributos de calidad se crea un proyecto Papyrus para el lenguaje SoaML+QoS. Luego se realiza la importación del modelo SoaML provisto por el grupo que desarrolló el plugin SoaML2Code. Esta funcionalidad permite importar todos los elementos del modelo SoaML origen al modelo SoaML+QoS destino. A continuación, se construyen los diagramas que modelan la SOA arrastrando los elementos importados desde el explorador del modelo al lienzo.

Otra opción brindada es la creación de todos los diagramas SoaML desde cero como se especifica en el caso de estudio del informe final del plugin SoaML.

Luego se realiza la importación del modelo de características de calidad QoS realizado previamente. Esto permite importar todos los elementos de un modelo QoS definido con el plugin QoS a un modelo SoaML+QoS, logrando la reutilización de taxonomías de características de calidad en diversas arquitecturas SOA. Tanto la importación de modelos SoaML como de modelos QoS pueden ser realizadas utilizando las funcionalidades implementadas que aparecen en el menú contextual del modelo, como se muestra en la Figura 76.

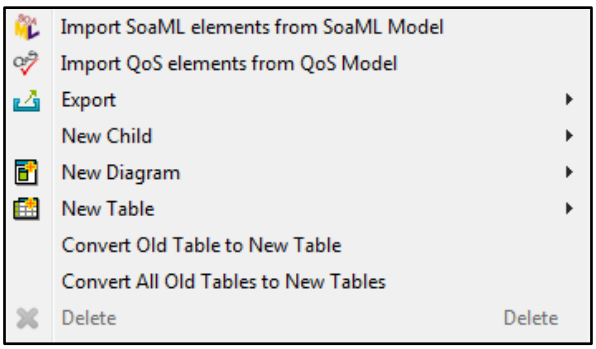

Figura 76. Importar elementos SoaML y QoS

Otra opción que se brinda es crear el diagrama de características de calidad directamente en el mismo proyecto SoaML+QoS, como se muestra en la Figura 77. Esto tiene como desventaja que la taxonomía definida en el proyecto no podrá ser reutilizada para el modelado de otros sistemas.

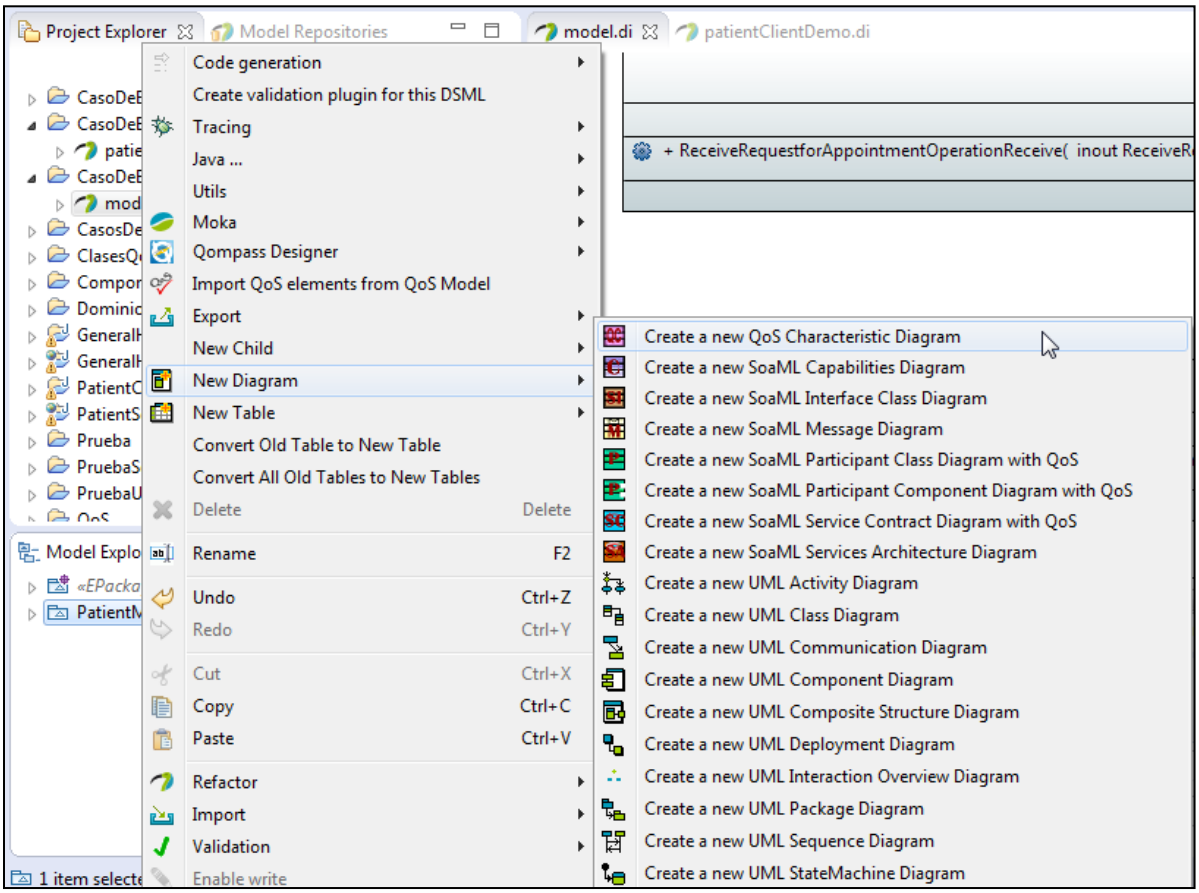

Figura 77. Creación de diagrama *QoS Characteristic*

Luego de tener modelada la SOA y definidas las características de calidad QoS, se agregan los valores de las características de calidad asociándolos a los puertos y contratos de servicio del modelo, como se detalla a continuación.

Se crea un elemento QoSValue con el nombre “Service Response Time” y se configura para que instancie la QoSCharacteristic “Service Response Time” definida en el modelo QoS. Se crean tres elementos de tipo Slot que se corresponden con los elementos QoSDimension contenidos en la QoSCharacteristic

"Service Response Time" ("ServiceProcessingTime", "ServiceLatencyTime" y "ServiceResponseTime") y se setean los valores para cada slot. Esto permite justamente instanciar cada dimensión con los valores específicos que se quieren definir para cada servicio en cada modelo SoaML. En nuestro caso, se asocia el puerto del tipo Service de nombre "ReceiveRequestforAppointment" del participante "GeneralHospital" con la QoSValue mediante un conector QoSOffered, ya que es el proveedor del servicio y es quien ofrece entonces la calidad de servicio definida.

Luego se crea un elemento QoSValue con el nombre "Access Control" y se configura para que instancie la QoSCharacteristic "Access Control" definida en el modelo QoS. Se crea otro elemento de tipo Slot que corresponde al elemento QoSDimension contenido en la QoSCharacteristic "Access Control" ("Policy") y se setea el valor para dicho slot. Luego, se asocia el puerto Service de nombre "ReceiveRequestForPatientMedicalRecord" de CentralHealth y los puertos Service de nombre "ReceiveRequestForAppointment" y "ReceivePatientMedicalRecord" de GeneralHospital con la QoSValue mediante un conector QoSRequired, indicando que se requiere autenticación para consumir los servicios.

El diagrama final es como el que se muestra en la Figura 78.

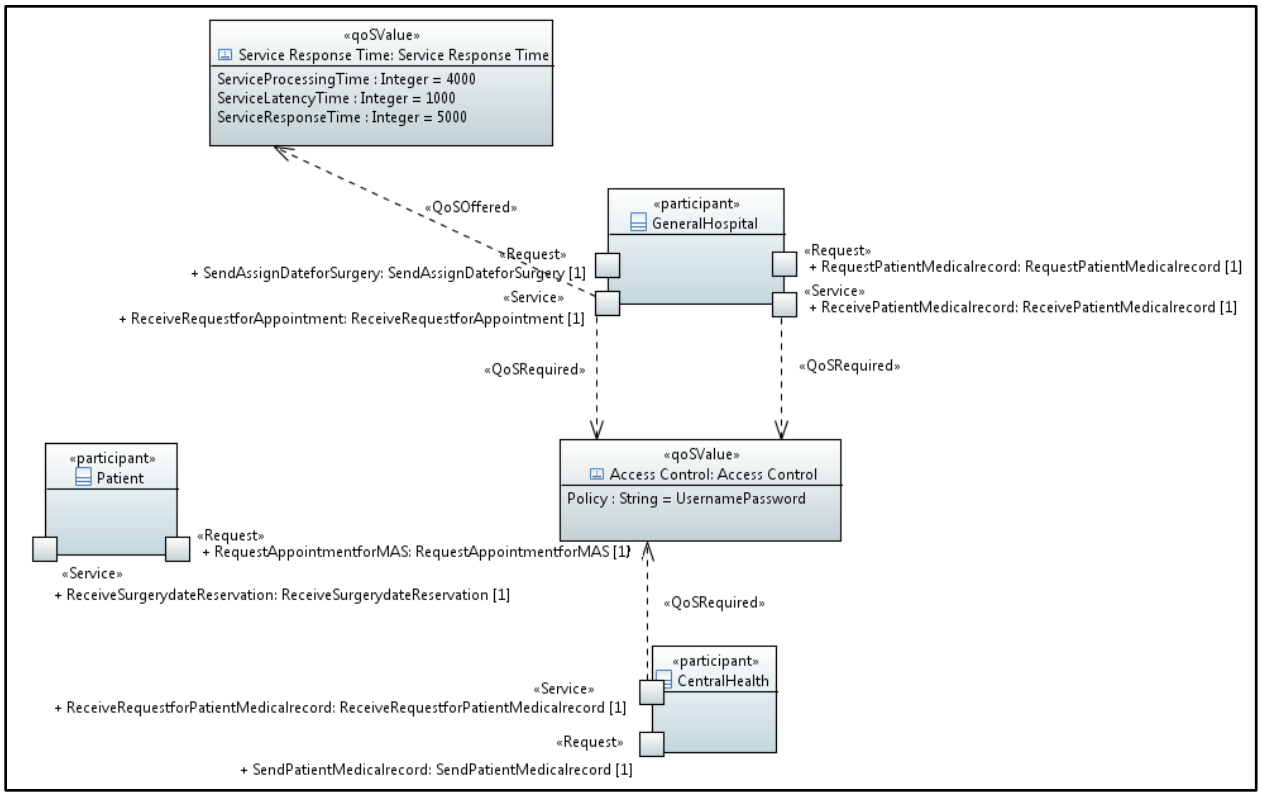

Figura 78. Diagrama Participant Class with QoS

Las características de calidad se pueden asociar también a los contratos de los servicios, como fue presentado en la sección 2.7. Para esto se crea un elemento QoSValue con el nombre "Throughput" y se configura para que instancie la QoSCharacteristic "Service Throughput" del modelo QoS definido. Se crea un nuevo elemento de tipo Slot que corresponde al elemento QoSDimension contenido en la QoSCharacteristic "Service Throughput" ("NSEOP") y se setea el valor para dicho slot. Luego, se asocia el

elemento ServiceContract de nombre “ReceiveRequestforPatientMedicalrecordServiceContract” que corresponde al contrato de servicios del servicio “ReceiveRequestForPatientMedicalRecord” con la QoSValue mediante un conector QoSContract, que indica que en ese contrato se acuerda la característica de calidad definida.

Para agregar otra, se crea un elemento QoSValue con el nombre “Availability” y se configura para que instancie la QoSCharacteristic “Availability” del modelo QoS definido. Se crea un nuevo elemento de tipo Slot que corresponde al elemento QoSDimension contenido en la QoSCharacteristic “Availability” (“ServiceAvailability”) y se setea el valor para dicho slot. Luego, se asocia el elemento ServiceContract de nombre “ReceiveRequestforPatientMedicalrecordServiceContract” que corresponde al contrato de servicios del servicio “ReceiveRequestForPatientMedicalRecord” con la QoSValue mediante un conector QoSContract, indicando así las características de calidad acordadas dentro del contrato.

Por último, se crea otro elemento QoSValue con el nombre “Availability” y se configura el QoSValue para que instancie la QoSCharacteristic “Availability” del modelo QoS definido. Se crea un nuevo elemento de tipo Slot que corresponde al elemento QoSDimension contenido en la QoSCharacteristic “Availability” (“ServiceAvailability”) y se setea el valor para dicho slot. Luego, se asocia el elemento ServiceContract de nombre “ReceiveRequestforAppointmentServiceContract” que corresponde al contrato de servicios del servicio “ReceiveRequestForAppointment” con la QoSValue mediante un conector QoSContract, para establecer las características de calidad acordadas.

En la Figura 79 se puede ver el diagrama finalizado.

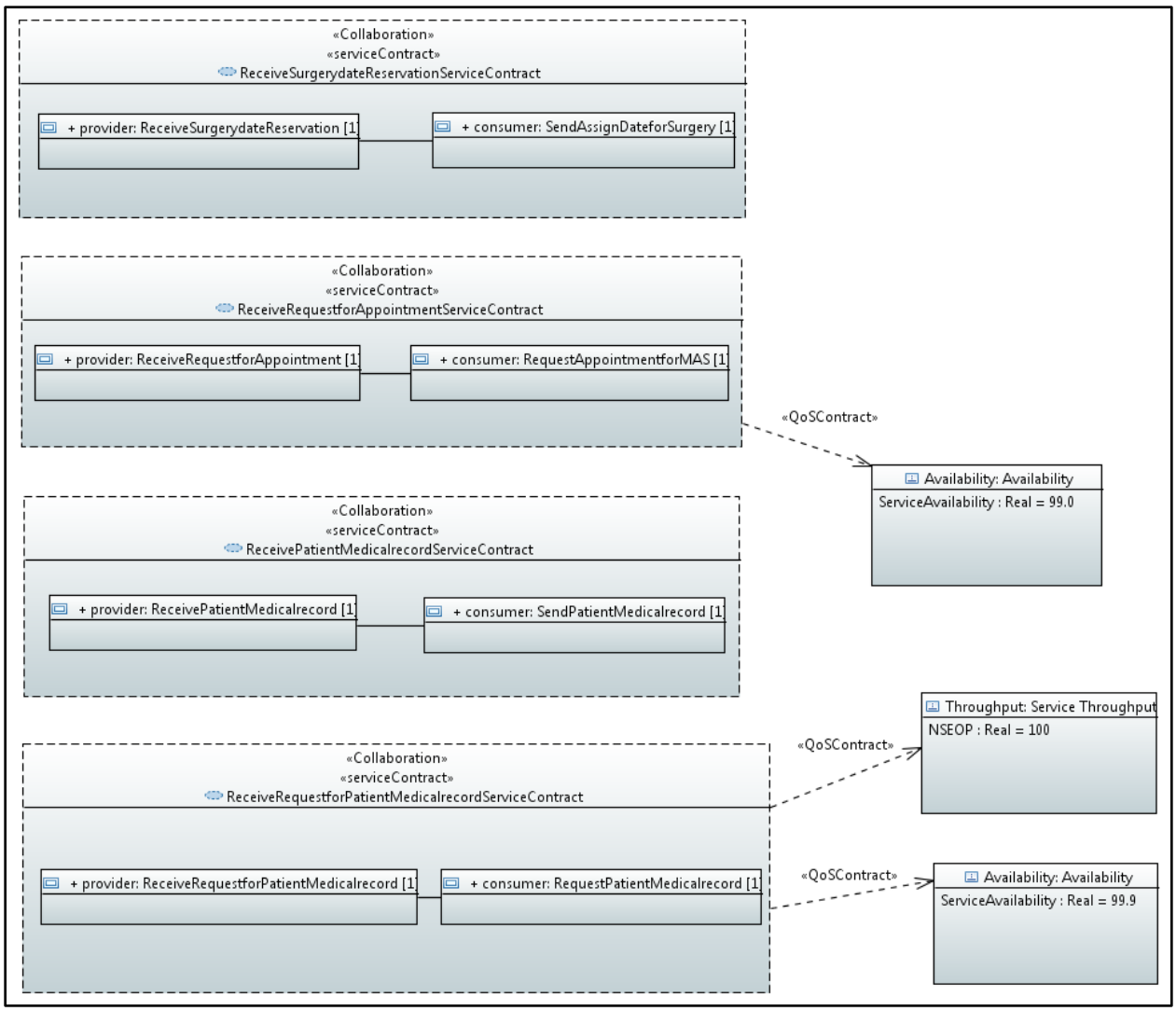

Figura 79. Diagrama Service Contract with QoS

Para finalizar, se agrupan todos los elementos QoS en un paquete nombrado “QoS”, situado luego del paquete “ServicePatientMASProcess”.

La Figura 80 muestra la estructura del árbol del modelo implementado en Papyrus.

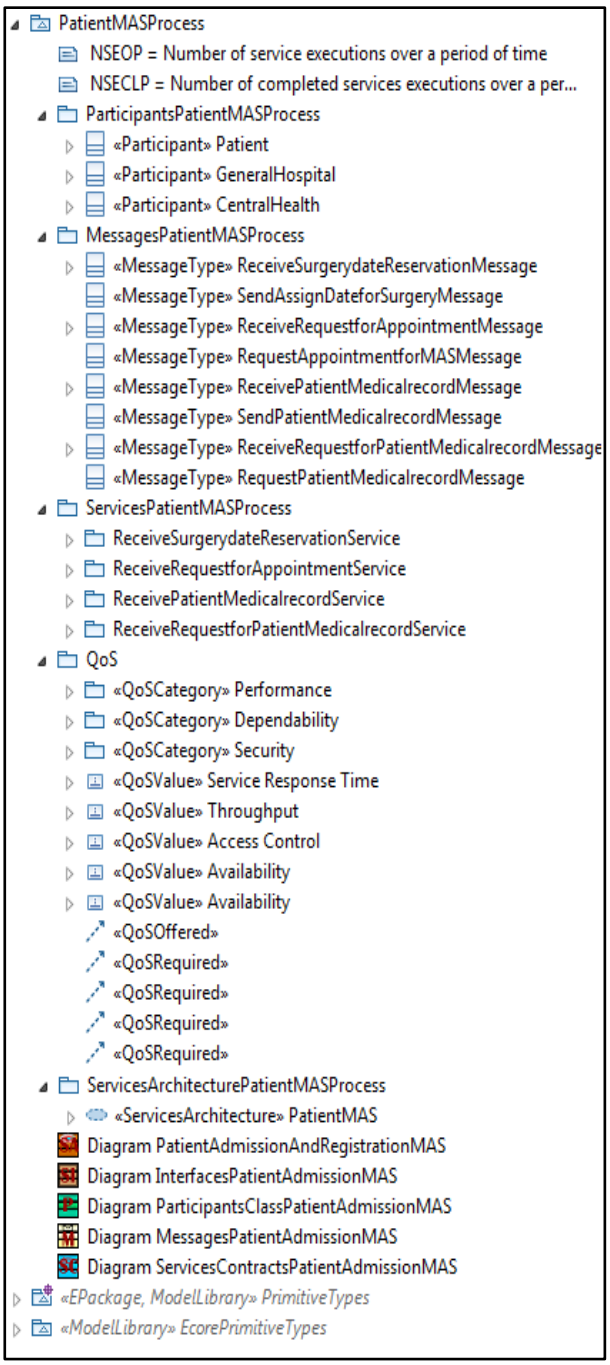

Figura 80. Model explorer del caso de estudio

6.2.2. Generación de código

A partir del modelo implementado en la parte anterior, se genera el código asociado utilizando el plugin SoaML2Code actualizado para incluir las características de QoS agregadas al modelo SoaML.

Para comenzar se selecciona la opción “Generate Java from UML” de la barra de herramientas SoaML Toolkit -> SoaML2Code y a continuación se selecciona el proyecto creado en el punto anterior correspondiente al caso de estudio.

En la primera página del *wizard*, para la primera generación se selecciona el participante "GeneralHospital", como se puede ver en la Figura 81.

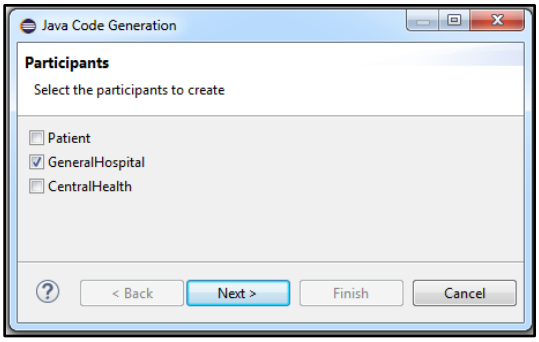

Figura 81. Wizard - Página de Participantes - GeneralHospital seleccionado

En la siguiente página se selecciona la opción "Dynamic Web Project (war)" con JAX-WS ri para JBoss 7.1.1 y también se selecciona que se genere el cliente, como se muestra en la Figura 82

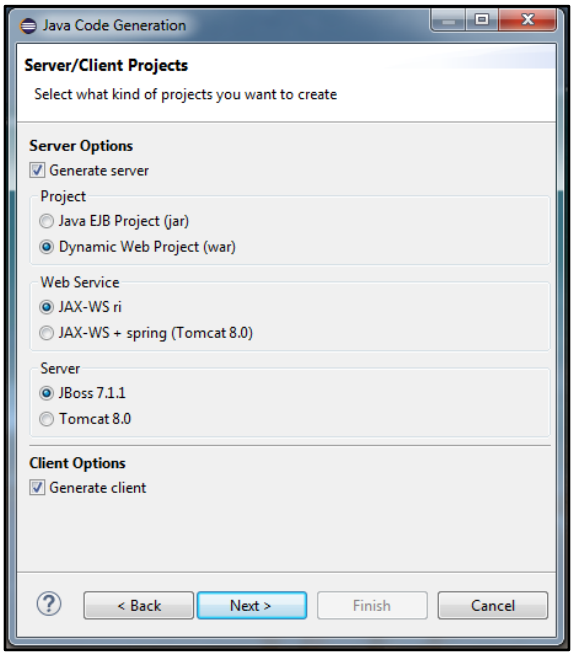

Figura 82. Wizard - Página de Servidor-Cliente - war JAX-WS ri JBoss

La página donde se debe configurar la IP y puerto donde se publicará el servicio se mantiene con sus valores por defecto, como se puede ver en la Figura 83.

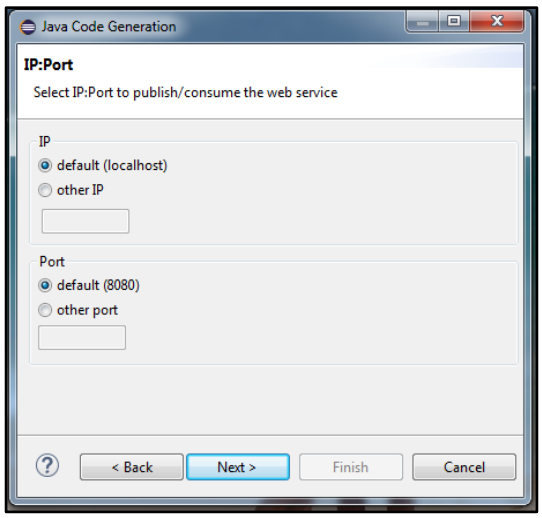

Figura 83. Wizard - Página de IP:Puerto

A continuación se debe elegir la correspondencia para la generación de características de calidad QoS, como se presentó en la sección 4.3.2.

En primer lugar presentamos la generación para el participante "GeneralHospital". Tiene una característica "Service Response Time" asociada a uno de sus puertos de servicio y otra para el control de acceso, por lo que en la última página del *wizard* se selecciona la opción "WS-Policy Generic" para "Service Response Time-ServiceLatencyTime", "WS-Policy Response Time" para "Service Response Time-ServiceResponseTime" y "WS-Security Username Token" para "Access Control-Policy", como se muestra en la Figura 84. En el caso que lo que elige el usuario no se corresponda o no tenga sentido respecto de la definición de características de QoS, el generador igualmente generará el código correspondiente a cada implementación, como se explicó en la sección 4.3.2.

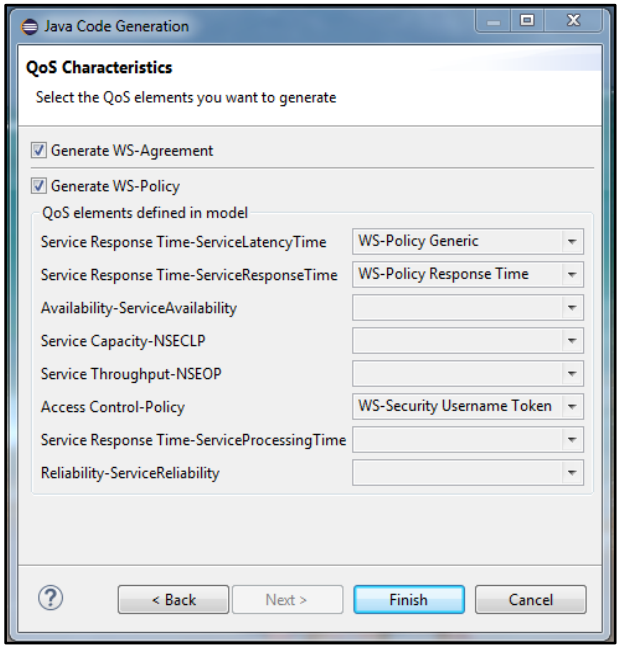

Figura 84. Wizard - Página de QoS

La estructura del proyecto GeneralHospitalServer generado se muestra en la Figura 85, donde se mantienen las correspondencias y definiciones realizadas en el generador SoaML2Code, y se agrega la generación de características de calidad QoS como se presentó en la sección 4.4.

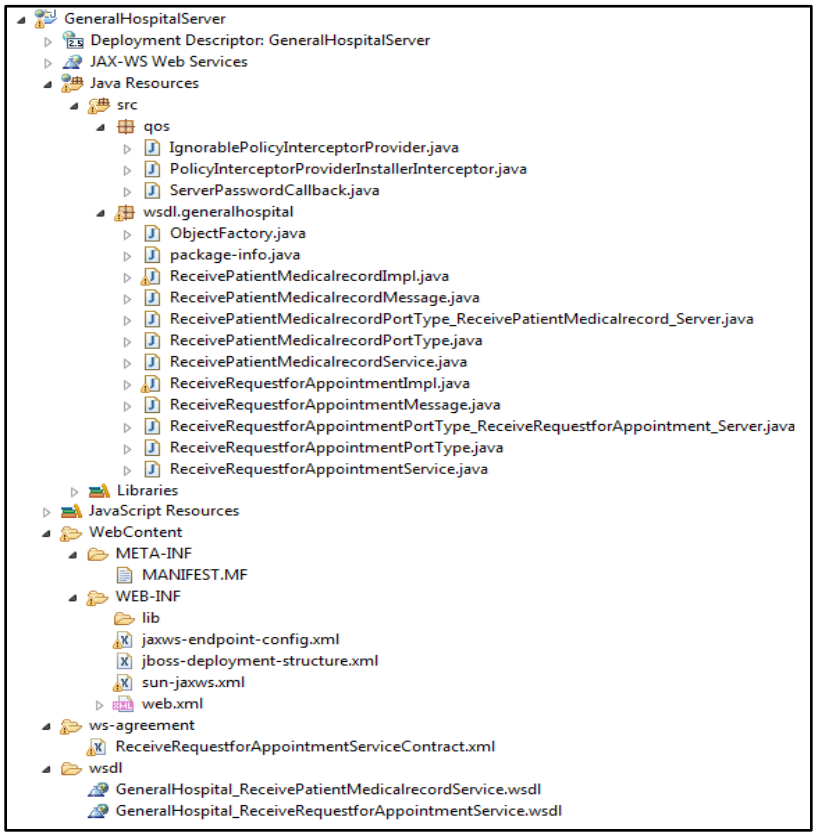

Figura 85. Estructura de GeneralHospitalServer war JAX-WS ri Jboss

Las clases generadas cuyo nombre tiene el sufijo “Impl” deben ser modificadas manualmente implementando la lógica de negocio del servicio, como se explica en el informe del plugin SoaML2Code. En la Figura 86 se puede ver una implementación de ejemplo.

```
public wsdl.generalhospital.ReceiveRequestforAppointmentMessage receiveRequestforAppointmentOperationReceive(
        ReceiveRequestforAppointmentMessage receiveRequestforAppointmentParameter1) {
    LOG.info("Executing operation receiveRequestforAppointmentOperationReceive");
    System.out.println(receiveRequestforAppointmentParameter1);
    try {
        wsdl.generalhospital.ReceiveRequestforAppointmentMessage _return = receiveRequestforAppointmentParameter1;
        System.out.println("El paciente con número de afiliado " + _return.getAffiliateNumber() +
                           " solicita programar la CMA con el médico " + _return.getDoctorsName());
        return _return;
    } catch (java.lang.Exception ex) {
        ex.printStackTrace();
        throw new RuntimeException(ex);
    }
}
```

Figura 86. Ejemplo de implementación de servicio

El WSDL generado mantiene por lo tanto la misma estructura que el generado por el plugin SoaML2Code, con la diferencia de que se agrega una policy asociada al puerto del servicio por cada QoSValue asociada a éste. Como en el modelo se asoció una QoSValue de “Service Response Time-ServiceLatencyTime” al puerto “ReceiveRequestforAppointment” y se seleccionó la implementación

"WS-Policy Generic" entonces se genera una policy con *tag* "ServiceLatencyTime" y con atributos los seteados en la definición de la característica y el valor dado a la QoSDimension en el modelo. Del mismo modo se genera la policy "ServiceResponseTime", ya que había una QoSValue de "Service Response Time-ServiceResponseTime" asociada al puerto "ReceiveRequestforAppointment" y se seleccionó la implementación "WS-Policy Response Time". Por último, como se asoció la QoSValue de "Access Control-Policy" al mismo puerto y se seleccionó la implementación "WS-Security Username Token", se genera la policy según el estándar para Username Token. En la Figura 87 se puede observar un extracto del wsdl con el agregado de QoS.

```
<wsdl:service name="ReceiveRequestforAppointmentService">
    <wsdl:port
        binding="serv:ReceiveRequestforAppointmentServiceBinding" name="ReceiveRequestforAppointment">
        <soap:address location="http://localhost:8080/GeneralHospitalServer/ReceiveRequestforAppointmentService"/>
        <wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy">
            <wsp:All>
                <ServiceLatencyTime direction="decreasing"
                    statisticalQualifier="maximum" unit="ms" value="1000"/>
                <ServiceResponseTime direction="decreasing"
                    statisticalQualifier="maximum" unit="ms" value="5000"/>
                <sp:SupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
                    <wsp:Policy>
                        <sp:UsernameToken sp:IncludeToken=
                        "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                            <wsp:Policy>
                                <sp:WssUsernameToken11/>
                            </wsp:Policy>
                        </sp:UsernameToken>
                    </wsp:Policy>
                </sp:SupportingTokens>
            </wsp:All>
        </wsp:Policy>
    </wsdl:port>
</wsdl:service>
```

Figura 87. Extracto de wsdl de GeneralHospitalServer para proyecto JAXWS-RI y JBoss

Por otro lado, dado que se seleccionó que se generara WS-Agreement, se generó el archivo ReceiveRequestforAppointmentServiceContract.xml, ya que se asoció un elemento QoSValue al contrato entre los participantes "Patient" y "GeneralHospital" y este último es el proveedor del servicio.

Para el control de acceso se generó la clase *ServerPasswordCallback* y los archivos de configuración *jaxws-endpoint-config.xml* y *jboss-deployment-structure.xml*. Además, se agregó la anotación *EndpointConfig* a la clase *ReceiveRequestforAppointmentImpl*, como se ve en la Figura 88.

```
@org.jboss.ws.api.annotation.EndpointConfig(configFile = "WEB-INF//jaxws-endpoint-config.xml",
                                            configName = "Custom WS-Security Endpoint")
public class ReceiveRequestforAppointmentImpl implements
        ReceiveRequestforAppointmentPortType {
```

Figura 88. Anotación en clase ReceiveRequestforAppointmentImpl

En el cliente se generó el mismo WSDL y el documento WS-Agreement ReceiveRequestforPatientMedicalrecordServiceContract.xml, ya que se asociaron elementos QoSValue al contrato entre los participantes "GeneralHospital" y "CentralHealth", en donde el primero, el cual es el que estamos generando, es el consumidor.

Para el envío del usuario y la contraseña requeridos para consumir el servicio se generó la clase UsernameTokenUtil dentro del package qos y se modificó la clase

ReceiveRequestforAppointmentPortType_ReceiveRequestforAppointment_Client para incluir la invocación a la clase mencionada anteriormente.

Además, dado que se seleccionó la generación de código “WS-Policy Response Time”, se agrega el registro de los tiempos de respuesta en milisegundos en la clase modificada anteriormente y se incluye log4j y su archivo de configuración.

La estructura del proyecto GeneralHospitalClient generado se muestra en la Figura 89.

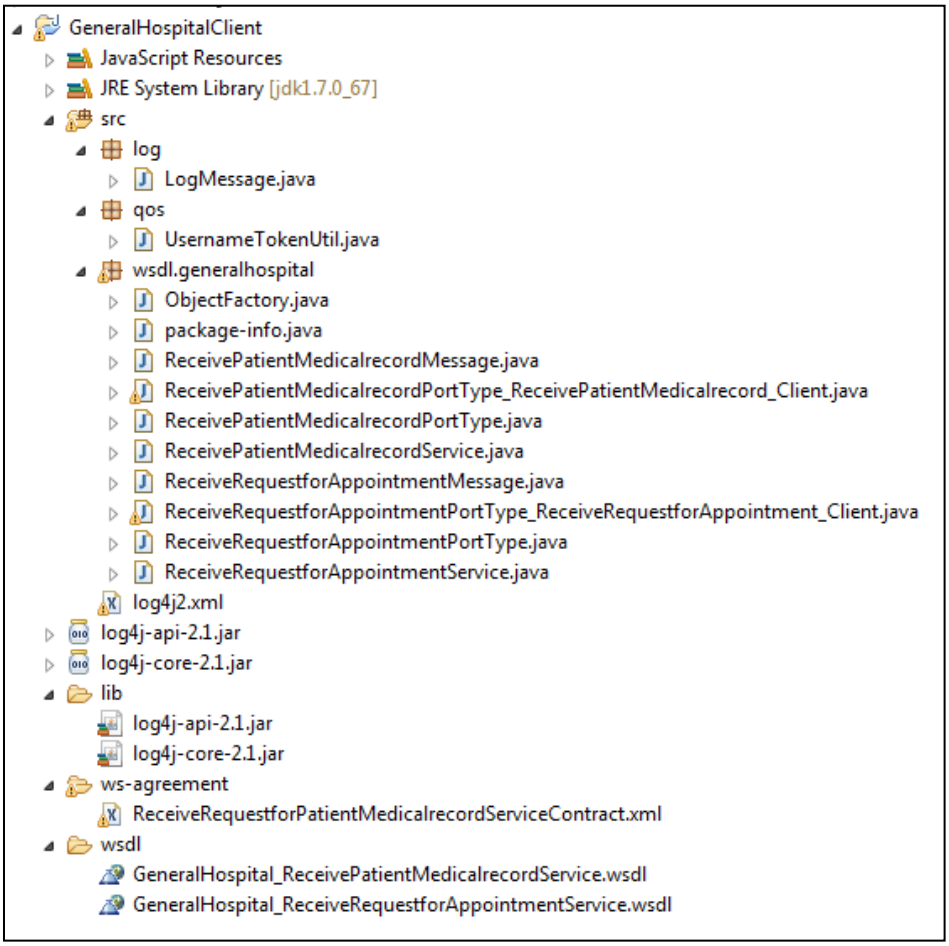

Figura 89. Estructura del proyecto GeneralHospitalClient

En este caso se deben modificar las clases cuyo sufijo es “Client” y que contienen el método main, para incluir la lógica de invocación del servicio. Un ejemplo de implementación del cliente se puede ver en la Figura 90. Allí se resaltó el código generado para imprimir los tiempos de inicio y fin de la invocación del servicio y el agregado del usuario y contraseña en el cabezal del mensaje SOAP para cumplir con la política de control de acceso. Estos datos de autenticación deben ser modificados de acuerdo a los utilizados en la organización.

```
public static void main(String args[]) throws java.lang.Exception {
    logger.info(new LogMessage(EventType.START, "ReceiveRequestforAppointmentService"));
    URL wsdlURL = ReceiveRequestforAppointmentService.WSDL_LOCATION;
    if (args.length > 0 && args[0] != null && !"".equals(args[0])) {
        File wsdlFile = new File(args[0]);
        try {
            if (wsdlFile.exists()) {
                wsdlURL = wsdlFile.toURI().toURL();
            } else {
                wsdlURL = new URL(args[0]);
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }
    ReceiveRequestforAppointmentService ss = new ReceiveRequestforAppointmentService(wsdlURL, SERVICE_NAME);
    ReceiveRequestforAppointmentPortType port = ss.getReceiveRequestforAppointment();
    UsernameTokenUtil.addUsernameTokenProfile((WSBindingProvider) port, "user", "password");
    {
        System.out.println("Invoking receiveRequestforAppointmentOperationReceive...");
        wsdl.generalhospital.ReceiveRequestforAppointmentMessage _param = new ReceiveRequestforAppointmentMessage();
        _param.setAffiliateNumber(new BigInteger("12345678"));
        _param.setDoctorsName("Christina Yang");
        wsdl.generalhospital.ReceiveRequestforAppointmentMessage _return = port.receiveRequestforAppointmentOperationReceive(_param);
        System.out.println("receiveRequestforAppointmentOperationReceive.result="+ _return);
    }
    logger.info(new LogMessage(EventType.COMPLETE,"ReceiveRequestforAppointmentService"));
    System.exit(0);
}
```

Figura 90. Ejemplo de implementación de cliente

En segundo lugar presentamos la generación para el participante “CentralHealth” para mostrar otras opciones de generación. En este caso se selecciona “Dynamic Web Project (war)” y “JAX-WS + spring (Tomcat)”.

Como el participante tiene una característica de calidad para el control de acceso asociada a su puerto de servicio “ReceiveRequestForPatientMedicalRecord”, se selecciona en la página de QoS en el *wizard* la generación de WS-Security Username Token, como se observa en la Figura 91.

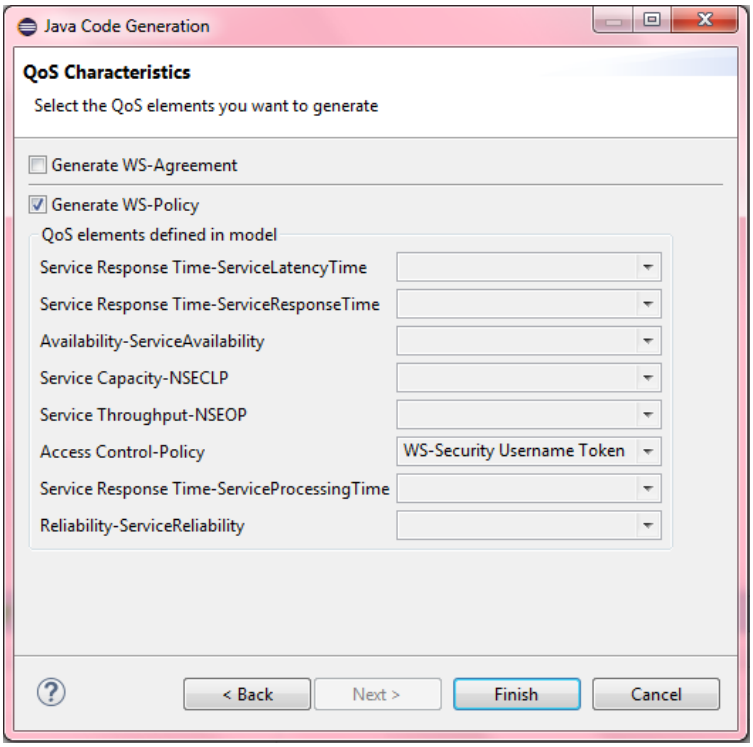

Figura 91. Wizard - Página de QoS

La estructura del proyecto CentralHealthServer generado se puede ver en la Figura 92.

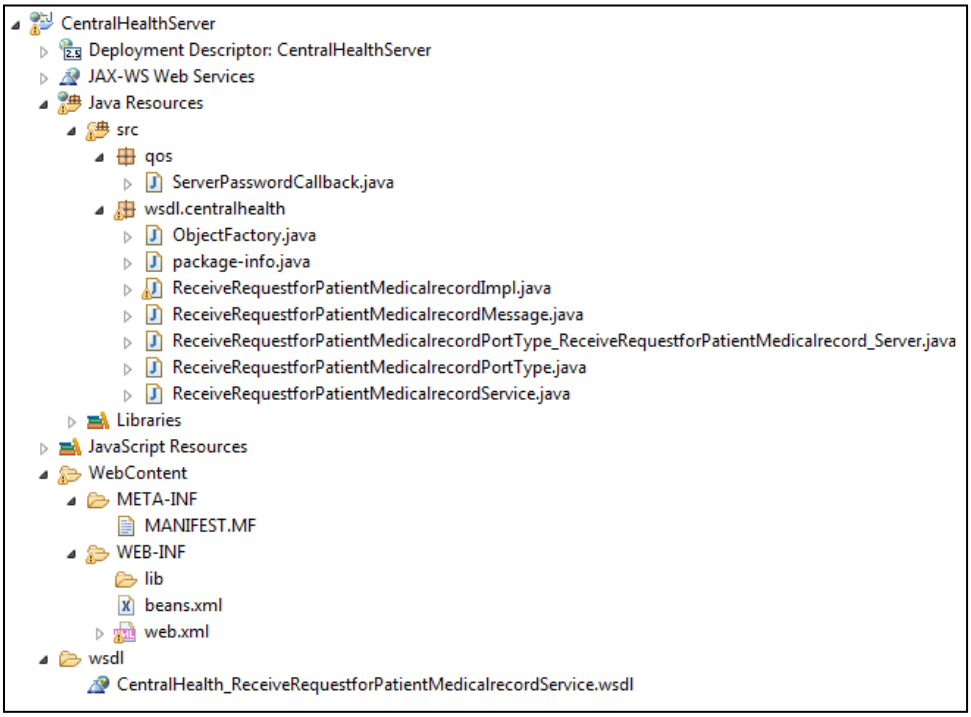

Figura 92. Estructura de CentralHealthServer war JAX-WS+Spring

El WSDL generado tiene la política para el control de acceso según el estándar para Username Token, como se puede ver en la Figura 93.

```
<wsdl:service name="ReceiveRequestforPatientMedicalrecordService">
    <wsdl:port
        binding="serv:ReceiveRequestforPatientMedicalrecordServiceBinding" name="ReceiveRequestforPatientMedicalrecord">
        <soap:address location="http://localhost:8080/CentralHealthServer/ReceiveRequestforPatientMedicalrecordService"/>
        <wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy">
            <wsp:All>
                <sp:SupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
                    <wsp:Policy>
                        <sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                            <wsp:Policy>
                                <sp:WssUsernameToken11/>
                            </wsp:Policy>
                        </sp:UsernameToken>
                    </wsp:Policy>
                </sp:SupportingTokens>
            </wsp:All>
        </wsp:Policy>
    </wsdl:port>
</wsdl:service>
```

Figura 93. Extracto de wsdl de CentralHealthServer para proyecto JAX-WS+Spring

Para el control de acceso se generó la clase *ServerPasswordCallback* y se modificó el archivo beans.xml para añadir la propiedad que mapea la clave ws-security.callback-handler con dicha clase.

Dado que la generación del cliente no depende de la tecnología utilizada en el servidor, el proyecto cliente CentralHealthClient contiene el mismo código que GeneralHospitalClient respecto al envío del usuario y contraseña para consumir el servicio.

6.3.Evaluación del caso de estudio

El caso de estudio se realizó sin problemas, tanto la parte del modelado de la arquitectura de servicios con características de calidad como la parte de generación de código asociado.

Se pudo comprobar que la posibilidad de crear modelos QoS y SoaML+QoS de forma independiente otorga flexibilidad al permitir la reutilización de un modelo de características de calidad en múltiples modelos de arquitecturas de servicios. A su vez, la posibilidad de importar modelos SoaML a un proyecto SoaML+QoS permite la incorporación de características de calidad a un modelo SoaML previamente definido, sin necesidad de crearlo desde cero.

Por el lado del generador de código, consideramos positivo el cambio de la ventana de diálogo del plugin SoaML2Code a un *wizard* con pasos que dividen claramente las distintas opciones de generación, el cual brinda mayor usabilidad.

Una desventaja del generador de código es que en la parte de selección de generación de QoS se muestran todas las características y dimensiones de calidad encontradas en el modelo, independientemente de si los participantes seleccionados tiene asociadas esas características a sus puertos o contratos.

7. Conclusiones y trabajo futuro

El capítulo se organiza en 4 secciones: en la 7.1 se exponen las evaluaciones del proceso de desarrollo, en la 7.2 se describen los aportes, en la 7.3 se comentan las posibles extensiones al producto en un futuro trabajo y por último, en la sección 7.4, se exponen las conclusiones de todo el trabajo realizado.

7.1.Evaluación

Fue posible construir un producto de acuerdo a los requerimientos definidos en el alcance del proyecto en el tiempo estipulado, cumpliendo con los cinco objetivos planteados al comienzo. Dentro del proceso de desarrollo consideramos como más relevante la migración de los plugins existentes para su correcto funcionamiento en la versión Luna de Eclipse y versión 1.0 de Papyrus, el desarrollo de los diagramas necesarios para el modelado de características de calidad y de arquitecturas SOA con aspectos no funcionales, y la inclusión de estos aspectos de calidad modelados al código generado.

Durante el desarrollo del proyecto nos encontramos con algunas dificultades. La más importante fue la falta de documentación actualizada de Papyrus, la cual llevó a evaluar la posibilidad de continuar trabajando con la versión 0.7 de Papyrus y Eclipse Helios. Finalmente decidimos invertir una cantidad de tiempo importante en investigación y en la migración, para obtener un producto de calidad que funcione en la última versión estable de Eclipse y Papyrus.

Al comienzo del proyecto se decidió continuar con la línea de utilización de estándares especificados por el OMG y realizar el modelado de características de calidad utilizando el perfil QoS definido en Quality of Service and Fault Tolerance [17]. Para maximizar la flexibilidad de modelado, se decidió construir dos plugins de modelado: uno para modelar taxonomías de características de calidad y otro para el modelado SoaML con restricciones de calidad en los servicios, los cuales funcionan de forma totalmente independiente. Este enfoque permite la reutilización de un modelo QoS en modelos diferentes de SoaML con QoS.

Finalmente, para el generador de código se evaluaron distintos estándares para la implementación de características de calidad en servicios web y se optó por implementar documentos WS-Agreement y generar políticas de control de acceso y de descripción de calidad de servicios usando los estándares WS-Security y WS-Policy, utilizando las tecnologías ya en uso en el plugin SoaML2Code.

7.2.Aportes

Consideramos que el plugin SoaML Toolkit desarrollado en este proyecto contribuye con una herramienta de modelado de características de calidad y de arquitecturas SOA con QoS cumpliendo con las especificaciones de OMG. A su vez, permite la generación de código a partir de modelos SoaML y SoaML con QoS, exponiendo Web Services utilizando distintas implementaciones, ya sea a través de anotaciones Enterprise JavaBeans (EJB) como de Web Services SOAP utilizando la API de Java JAX-WS, así como también la descripción de aspectos no funcionales de servicios mediante políticas expresadas con WS-Policy y WS-Security, y modelado de contratos de calidad con documentos WS-Agreement. Además permite también la generación de clientes Web Services para invocar los servicios expuestos y realizar pruebas sobre los mismos. De esta forma se completa el ciclo de modelado y desarrollo sin necesidad de utilizar ninguna otra herramienta externa a Eclipse.

La guía de desarrollo (Anexo Documento Técnico) contiene un manual con las instrucciones que se deben seguir para agregar nuevas implementaciones de características de calidad. Consideramos que este documento es un buen aporte que facilita la extensión del producto construido.

7.3.Posibles extensiones

Un aspecto importante a incluir en una posible extensión es el agregado de restricciones OCL en la definición de los perfiles QoS y SoaML con QoS. Esto permitiría la realización de validaciones semánticas sobre el modelo, garantizando que el modelo respeta las restricciones definidas en los perfiles.

Para la construcción del plugin de modelado QoS se debió realizar un modelo Papyrus del tipo Profile, en donde definimos todos los elementos que componen el perfil, incluso algunos elementos que no son utilizados en el diagrama de características de calidad ni en los diagramas SoaML con QoS. Una posible extensión es la implementación de nuevos tipos de diagramas que utilicen el perfil QoS de forma completa, como el modelado de niveles QoS y transiciones entre niveles.

El wizard de generación de código despliega la lista de características y dimensiones de calidad que se encuentran en el modelo, sin importar que estén asociados a puertos o contratos de los participantes seleccionados. Una posible mejora de usabilidad que se puede implementar es el filtrado de las características de calidad que no son utilizadas por los participantes seleccionados.

Por razones de tiempo, en este proyecto se implementó una de las políticas de seguridad más simples: UsernameToken con passwords de texto plano. Sería de gran interés extender esta política para la utilización de contraseñas derivadas (*digest*), y la implementación de otras políticas de seguridad como encriptación de cabezal y de cuerpo SOAP, y tokens SAML y X.509.

7.4.Conclusión

En la fase inicial del proyecto se realizó el estudio de bibliografía y artículos relacionados a la Computación Orientada a Servicios y Arquitectura Dirigida por Modelos, y las especificaciones de SoaML y QoS, lo que permitió comprender el contexto del proyecto y analizar las distintas alternativas para la implementación de modelado SoaML con características de calidad QoS. Además se estudiaron otros estándares relacionados, como WS-Policy, WS-Security y WS-Agreement, que finalmente fueron utilizados para incluir características de calidad en la generación de código.

En todas las etapas del proyecto se utilizaron herramientas que facilitaron el trabajo en equipo, permitiendo el trabajo en paralelo sobre los mismos recursos. Para la documentación, inicialmente utilizamos Google Docs, lo cual nos permitió trabajar en paralelo sobre diferentes secciones del mismo documento. La desventaja de utilizar esta herramienta es que no tiene un buen manejo del versionado, lo cual añade cierta complejidad cuando es necesario volver a versiones anteriores. Sobre el final del proyecto decidimos incorporar la documentación al servidor SVN proporcionado por la facultad, en donde estábamos versionando el código fuente de los plugins desde el inicio.

Se invirtió una cantidad de tiempo considerable a la investigación y migración de los plugins SoaML y SoaML2Code para la utilización de las últimas versiones de todas las tecnologías. Si bien la migración resultó ser una de las tareas más complejas del proyecto, principalmente por la falta de documentación actualizada, consideramos que valió la pena el esfuerzo ya que extiende la vida útil del plugin; el plugin

SoaML Toolkit funciona correctamente con la última versión de Eclipse y posiblemente en versiones posteriores si los cambios no impactan en el core utilizado.

Durante el desarrollo del plugin SoaML Toolkit, para cada problema se estudiaron diferentes alternativas de solución, investigando las ventajas y desventajas de cada una para así seleccionar la más adecuada. Se definieron distintas opciones para el modelado de características de calidad utilizando el perfil QoS. Se evaluaron las características de cada opción, y se decidió la construcción de un modelador independiente de QoS y otro de SoaML con QoS.

Se generaron casos de prueba para verificar las funcionalidades desarrolladas y comprobar la calidad del prototipo construido. Además de ejecutar casos de prueba para lo implementado en este proyecto, se volvieron a ejecutar los casos de prueba definidos para el plugin de modelado SoaML para verificar su correcto funcionamiento luego de la migración.

Se cumplió el objetivo principal del proyecto, construyendo una solución integral que permite el modelado de servicios en SoaML con la incorporación del modelado de QoS a los modelos SoaML, y la generación de código para dichas características de calidad. Para el modelado, se incorporaron los diagramas de características QoS, de contratos de servicios con QoS y de participantes como clases y participantes como componentes, ambos con QoS. Con respecto al generador, se incorporó la generación de documentos WS-Agreement y el agregado de políticas de calidad como WS-Policy genérica, WS-Policy Response Time y WS-Security Username Token, generados a partir de modelos SoaML con QoS.

Respecto a la funcionalidad de importar y exportar archivos con formatos XMI, se incorporó el parseo de modelos SoaML con QoS para así permitir el intercambio de éstos.

Como parte del caso de estudio, se logró modelar un diagrama de características de calidad basado en el modelo BPEMM [23] y utilizar estas características para modelar aspectos no funcionales en el modelo SoaML que representa el proceso de negocio CMA [39].

En el transcurso del proyecto nos encontramos con algunas dificultades que logramos superar, como ser la migración ya mencionada y lograr hacer funcionar las políticas de calidad implementadas en los distintos servidores de aplicación utilizados y con las diferentes tecnologías.

Como conclusión final, podemos decir que estamos sumamente conformes con la dedicación y el trabajo realizado. Destacamos los conocimientos adquiridos, tanto a nivel técnico como a nivel académico y humano, y valoramos el haber cumplido con todos los objetivos en el tiempo estipulado inicialmente.

8. Referencias

[1] Niels Schot. Model-Driven SOA. Abril 2012.

[2] Cedric Teyssie. Uml-based specification of qos contract negotiation and service level agreements. *Mobile Communications and Learning Technologies, Conference on Networking, Conference on Systems, International Conference on*, 0: 12, 2006.

[3] Stephen J. Mellor, Tony Clark, and Takao Futagami. Model-driven development: Guest editors' introduction. *IEEE software*, 20 (5): 14–18, 2003.

[4] Douglas C. Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39 (2): 25, 2006.

[5] Bran Selic. The pragmatics of model-driven development. *IEEE software*, 20 (5): 19–25, 2003.

[6] Colin Atkinson and Thomas Kuhne. Model-driven development: a metamodeling foundation. *Software, IEEE*, 20 (5): 36–41, 2003.

[7] Mariano Belaunde and Carol Burt. MDA guide. *Object Management Group, Tech. Rep. omg/2003-06-01*, 2003.

[8] Anneke G. Kleppe, Jos B. Warmer, and Wim Bast. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional, 2003.

[9] OMG Meta Object Facility (MOF) Core Specification version 2.4.2, 2014.

[10] OMG Unified Modeling Language version 2.5, 2013.

[11] XML Metadata Interchange Specification, 2014.

[12] Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall, 2005. ISBN 9780131858589.

[13] MP Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service oriented computing: State of the art and research challenges. *Computer*, 40 (11): 38–45, 2007.

[14] Mike P. Papazoglou and Willem-Jan Van Den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB journal*, 16 (3): 389–415, 2007.

[15] Grace A. Lewis, Edwin Morris, Soumya Simanta, and Lutz Wrage. Common misconceptions about service-oriented architecture. In *Commercial-off-the-Shelf (COTS)-Based Software Systems, 2007. ICCBSS'07. Sixth International IEEE Conference on*, pages 123–130. IEEE, 2007.

[16] Service oriented architecture Modeling Language (SoaML) Specification, v1.0.1, 2012.

[17] UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, v1.1, 2008.

[18] Niels Schot. QoS-aware model-driven SOA using SoaML. 2012.

[19] Liam O'Brien, Len Bass, and Paulo Merson. Quality attributes and service-oriented architectures. technical note. Technical report, CMU/SEI-2005-TN-014, 2005.

[20] Web Services Quality Factors Version 1.0, .

[21] OASIS, URL https://www.oasis-open.org/. [Último acceso: 07/12/2014]

[22] Quality Model for Web Services, 2005.

[23] Andrea Delgado, Barbara Weber, Francisco Ruiz, Ignacio García-Rodríguez de Guzmán, and Mario Piattini. An integrated approach based on execution measures for the continuous improvement of business processes realized by services. 2013.

[24] Andrea Delgado, Francisco Ruiz, Ignacio García-Rodríguez de Guzmán, and Mario Piattini. MINERVA: Model driven and sErvice oRiented framework for the continuous business processes improVement & relAtedtools. *5th Workshop on Engineering Service-Oriented Applications (WESOA'09) in 7th Int. Conf. on Service Oriented Computing (ICSOC'09)*, Noviembre 2009.

[25] Changzhou Wang, Guijun Wang, Haiqin Wang, Alice Chen, and Rodolfo Santiago. Quality of service (QoS) contract specification, establishment, and monitoring for service level management. In *Enterprise Distributed Object Computing Conference Workshops, 2006. EDOCW'06. 10th IEEE International*, pages 49–49. IEEE, 2006.

[26] A. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, and Ü Yalçinalp. Web Services Policy 1.5. *Framework*, 2007.

[27] Alain Andrieux and Karl Czajkowski. Web services agreement specification (WS-Agreement).

[28] *Eclipse Luna Documentation*. URL http://help.eclipse.org/luna/index.jsp. [Último acceso: 26/11/2014]

[29] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture, W3C Working Group Note, 11 February 2004. *World Wide Web Consortium*, 2004. URL http://www.w3.org/TR/ws-arch. [Último acceso: 30/11/2014]

[30] Sameer Tyagi. RESTful Web Services. 2006. URL http://www.oracle.com/technetwork/articles/-javase/index-137171.html. [Último acceso: 26/11/2014]

[31] Marc Hadley, Noah Mendelsohn, J. Moreau, H. Nielsen, and M. Gudgin. SOAP Version 1.2 Part 1: Messaging Framework. *W3C REC REC-soap12-part1-20030624, June*, pages 240–8491, 2003.

[32] LINS/FING/UDELAR. Web services. 2012. URL http://www.fing.edu.uy/inco/cursos/tsi/TSI2/-2012/teorico/tsi2-07-web-services.pdf. [Último acceso: 26/11/2014]

[33] Web Services Security 1.1, 2004.

[34] Web Services Security UsernameToken Profile 1.1, 2006.

[35] WS-SecurityPolicy 1.2, 2007.

[36] A D'Ambrogio. A Model-driven WSDL Extension for Describing the QoS of Web Services. In *Web Services, 2006. ICWS '06. International Conference on*, pages 789–796, Sept 2006.

[37] Oracle. Distributed multitiered applications. URL http://docs.oracle.com/javaee/7/tutorial/doc/-overview003.htm. [Último acceso: 29/09/2014]

[38] ProM, Process Mining Group. *Eindhoven University of Technology*.

[39] Andrea Delgado, Francisco Ruiz, Ignacio García-Rodríguez de Guzmán, and Mario Piattini. Business Process Service Oriented Methodology (BPSOM) with Service generation in SoaML. *23rd International Conference on Advanced Information Systems Engineering (CAiSE'11)*, Junio 2011.

[40] OMG. Business Process Model and Notation (BPMN) Version 2.0, 2011.

9. Anexos

- Arquitectura
- Casos de Prueba
- Casos de Uso
- Documento Técnico
- Estado del Arte
- Guía de Instalación
- Manual de Usuario

Proyecto de grado

Generación automática de Arquitecturas Orientadas a Servicios (SOAs) con SoaML y especificación de QoS

Arquitectura

Federico Bertolini, Sofía Pérez, María Noel Quiñones

Tutor
Andrea Delgado

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay
Diciembre 2014

Resumen

Este documento describe la infraestructura básica de la solución propuesta para implementar el conjunto de editores de diagramas SoaML con QoS, así como también la generación de código JEE y WS a partir de modelos del lenguaje mencionado, como plugins de Eclipse.

Contenido

1. Introducción

La solución propuesta extiende los plugins SoaML y SoaML2Code para lograr la incorporación de características de calidad al modelado SoaML y a la generación de código JEE y WS.

Para el modelado se siguió la misma idea que el proyecto SoaML y se reutilizó el código correspondiente a los plugins que implementan el diagrama de clases y el de estructura compuesta. De esta forma, se crearon nuevos plugins para cada tipo de diagrama QoS y SoaML+QoS para incorporar a la herramienta base.

Adicionalmente se realizó el upgrade del plugin SoaML a la nueva versión de Papyrus, la cual tiene gran cantidad de cambios de arquitectura. Esto permite que la solución pueda utilizarse en la última versión de Eclipse (Luna) y posiblemente posteriores si los cambios no impactan en el core utilizado.

En cuanto a la generación de código de QoS agregado a modelos SoaML, se incorporó al plugin SoaML2Code la generación de políticas WS-Policy y WS-Security en los archivos WSDL para la descripción de las características de calidad y para control de acceso a los servicios, y la generación de documentos WS-Agreement que describen las características de calidad asociadas a los contratos de servicios.

En la Figura 1 se puede ver la integración de los plugins junto con los artefactos generados por cada uno de éstos.

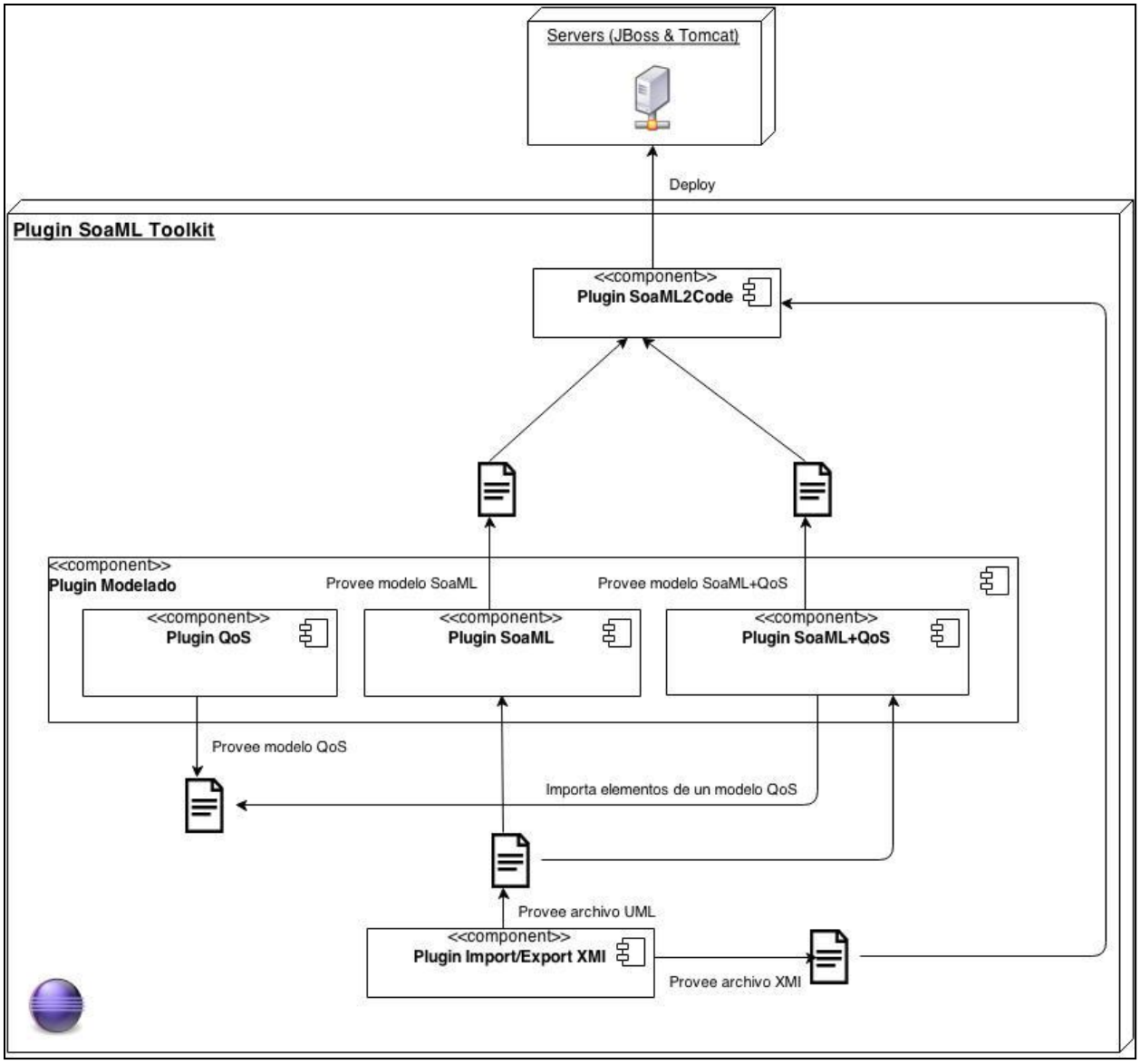

Figura 1. Integración de plugins

2. Arquitectura de la solución

La arquitectura fundamental de Eclipse está basada en plugins, lo cual permite integrar elementos (nuevas funcionalidades) al sistema, enriqueciéndolo con nueva características. Cada plugin debe extender ciertos puntos de extensión básicos del core Eclipse (relacionados al workbench, vistas, perspectivas, etc.), y a su vez puede ofrecer nuevos puntos de extensión para que otros plugins puedan reutilizar o extender ciertas funcionalidades.

Todos los tipos de diagramas que ofrece Papyrus están implementados en distintos módulos (archivos jar) que se encuentran en la carpeta plugins de Eclipse (carpeta en el directorio raíz de Eclipse que contiene los plugins para extender la funcionalidad del editor). En la arquitectura propuesta, se siguió este mismo camino, por lo que cada nuevo tipo de diagrama fue distribuido en un módulo distinto, que debe agregarse a la carpeta plugins de Eclipse. Es indispensable, para el correcto funcionamiento de los nuevos diagramas, que en el Eclipse donde sean agregados se encuentre instalado el plugin Papyrus. A su vez, el plugin para la generación de código estará implementado en otro módulo independiente.

De la misma forma en que fue realizado en el proyecto del plugin SoaML, se modificaron clases y algunos archivos de configuración de los diagramas pre-existente en Papyrus. Se decidió implementar dos plugins separados: por un lado el modelado de QoS, pudiendo así modelar taxonomías de características de calidad que podrán ser reutilizadas en distintos proyectos de modelado con SoaML u otros lenguajes como UML para sistemas tradicionales, y por otro, el modelado de SoaML con características de calidad QoS asociadas, manteniendo independiente el modelado de SoaML.

Se crearon por lo tanto dos plugin Papyrus que acceden al core del mismo, uno para modelado QoS y otro para modelado SoaML+QoS, los cuales facilitan las funcionalidades de aplicar los perfiles QoS y SoaML+QoS por defecto y agregar estas dos nuevas categorías de diagramas como opciones en el cuadro de diálogo al crear un nuevo modelo Papyrus.

En cuanto a la arquitectura del plugin encargado de importar y exportar archivos XMI, ésta se mantiene respecto a la versión existente del proyecto de modelado SoaML y la implementación fue modificada para que tuviera en cuenta el perfil del archivo de origen y así diferenciar la importación o exportación según si es SoaML o SoaML con QoS.

Para la arquitectura del plugin de generación de código se respetó la diseñada por el proyecto SoaML2Code, que incluye:

- Un parser que se encarga de realizar el parseo de modelos SoaML (archivo UML o XMI) para luego crear los objetos Java correspondientes.
- Un procesador de reglas que se encarga de procesar los objetos Java y aplicar reglas que definirán la invocación a los generadores de código de WS y Java EE.
- Un generador encargado de generar el código WS y Java EE que representa al modelo SoaML de partida.

En la Figura 2 se puede ver un esquema de los subsistemas involucrados y sus dependencias, donde los elementos resaltados en azul corresponden a los subsistemas desarrollados en este proyecto y los

resaltados en verde son los subsistemas que ya existían que requirieron modificaciones para la inclusión de modelos SoaML+QoS. El resto son subsistemas ya existentes que no sufrieron modificaciones.

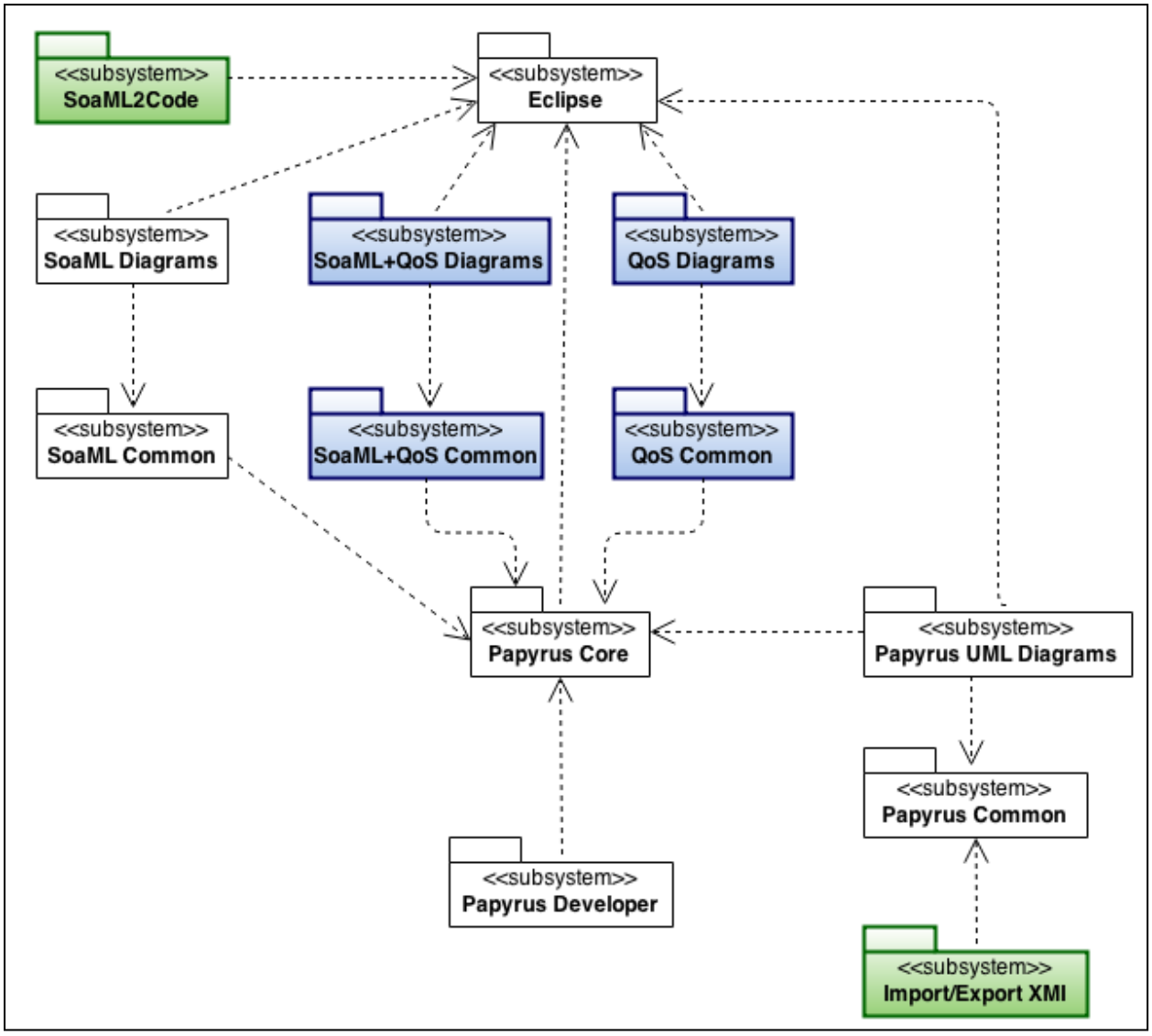

Figura 2. Subsistemas de la arquitectura Papyrus extendida

3. Implementación

En este capítulo se describe la forma en que se implementaron los distintos plugins para brindar la funcionalidad deseada.

Como se mencionó anteriormente, se decidió implementar dos plugins de forma independiente, uno para el modelado de QoS y otro para el modelado de SoaML con QoS. Los distintos componentes, así como sus relaciones y dependencias se pueden ver en la Figura 3.

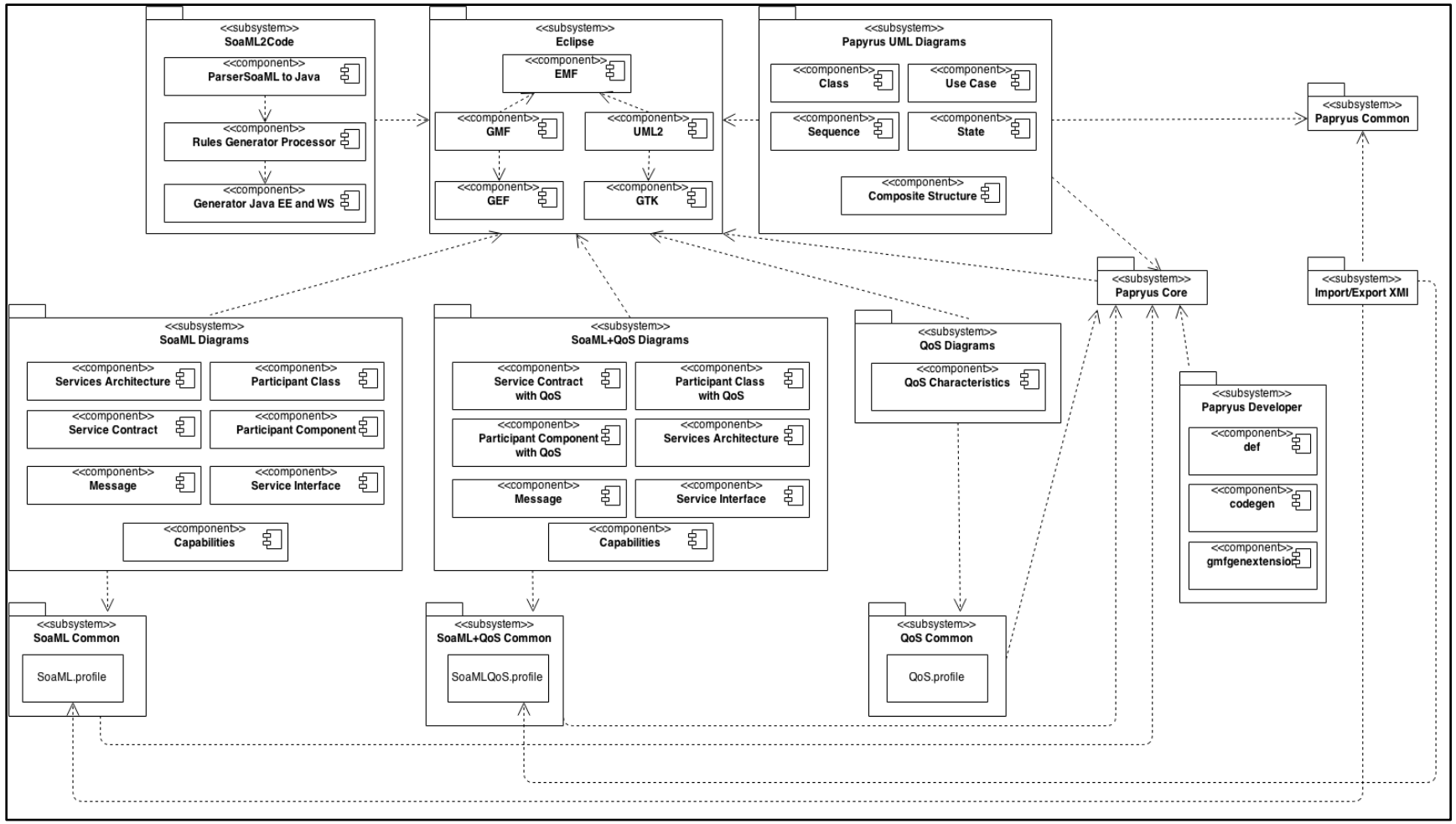

Figura 3. Diagrama de componentes de la solución

Se crearon dos nuevos plugins *"common"* distintos a los usados por los plugins pre-existentes. Uno para QoS, el cual permite aplicar el perfil QoS por defecto en el diagrama de tipo QoS y además, agregar una nueva categoría de diagramas. El otro es para SoaML+QoS, el cual, así como el anterior, permite aplicar el perfil SoaML+QoS por defecto en los nuevos diagramas de tipo SoaML+QoS y agregar una nueva categoría de diagramas. Esto se puede ver en la Figura 4.

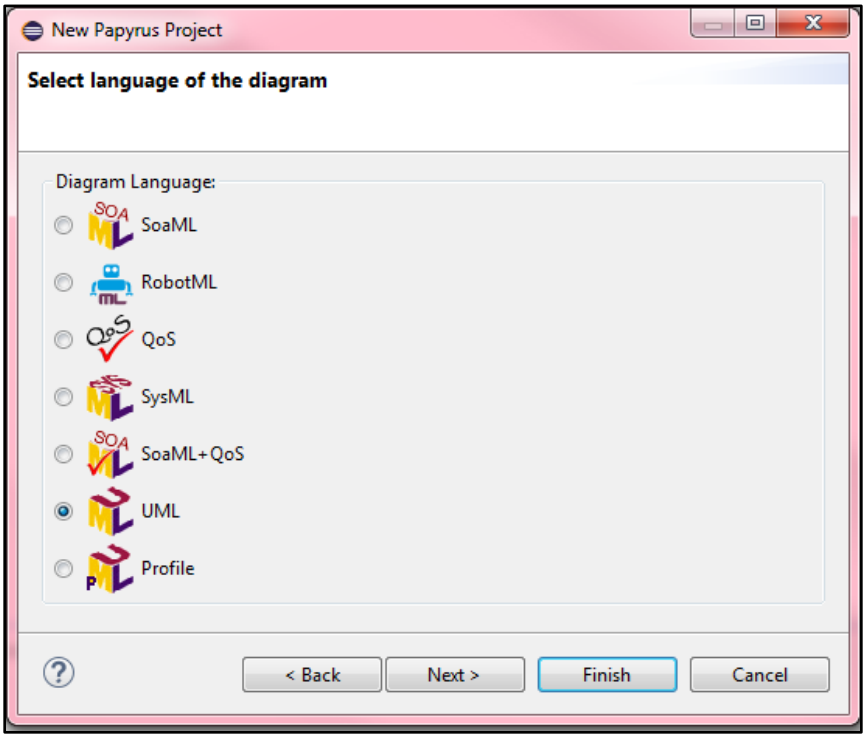

Figura 4. Opciones de categorías de diagramas Papyrus

Fue necesario generar código a partir de los perfiles, utilizando EMF, para que funcionaran correctamente los menús de Papyrus y éste mostrara las distintas opciones de diagramas que se pueden generar.

El editor de diagramas de *QoS Characteristics* fue realizado en base al editor de diagrama de clases pre-existente. Para SoaML+QoS se tomó como base lo implementado para SoaML, modificando los plugins que debían incluir características de calidad.

Como los editores de diagramas de *Service Contract*, *Participant Class* y *Participant Component* fueron creados a partir del editor de diagrama de estructura compuesta de Papyrus, ninguno contaba con el elemento *Instance Specification*, el cual es necesario para implementar el elemento *QoSValue*. Para lograr que funcione se debió agregar manualmente todo lo referente al elemento *Instance Specification*, utilizando como base el código del editor de diagrama de clases y copiándolo donde fuera necesario.

En el generador de código, dado que debíamos agregar nuevas opciones al *wizard* de generación de código, se tomó la decisión de modificarlo y hacerlo en pasos. Los primeros pasos coinciden con los pre-existentes, habiendo agregado la opción de elegir para qué servidor se desea generar en caso de que se elija generar un proyecto web dinámico. Esto se puede ver en las Figuras 5, 6 y 7.

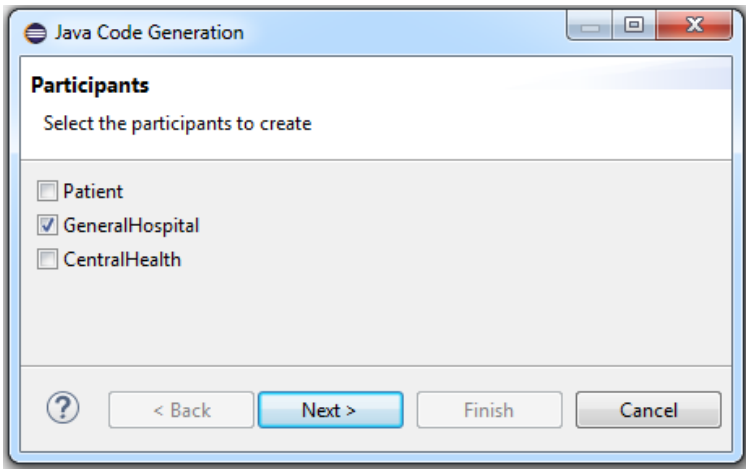

Figura 5. Primer paso del wizard - Selección de participantes

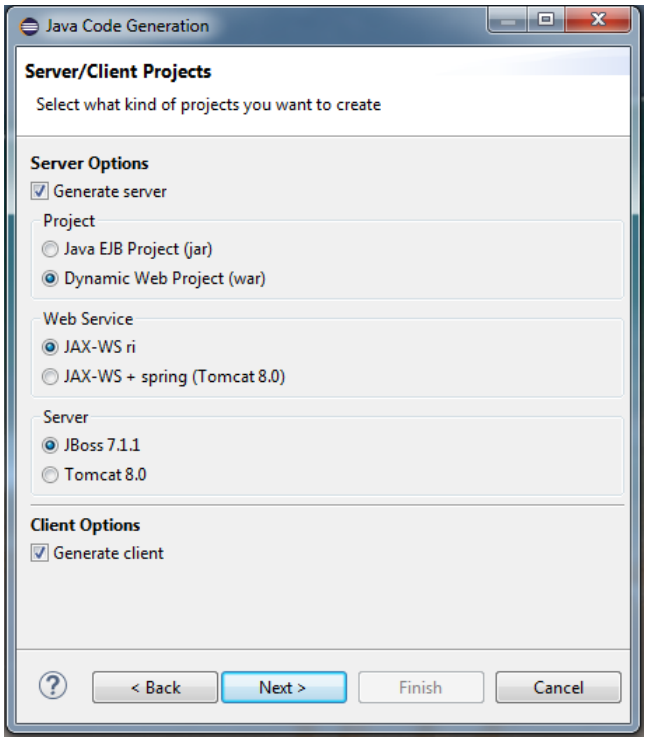

Figura 6. Segundo paso del wizard - Selección de servidor y cliente

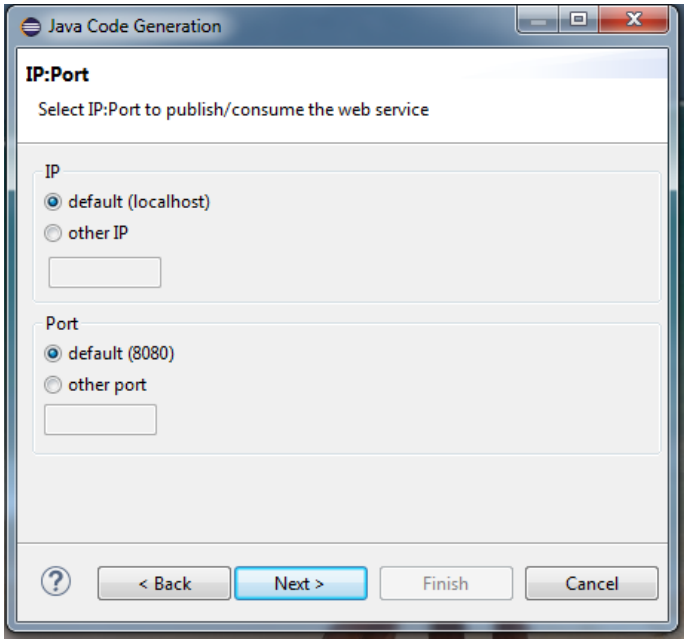

Figura 7. Tercer paso del wizard - Selección de IP y puerto

En caso de que el modelo del que se parte tenga características de calidad incluidas, al *wizard* se le agrega un cuarto paso, en el que se podrá seleccionar si se desea generar WS-Agreement y WS-Policy. En caso de generar WS-Policy se deberá seleccionar qué tipo se quiere generar para al menos un par QoSCharacteristic-QoSDimension (de las encontradas en el modelo), pudiendo elegir entre WS-Policy Generic, WS-Policy Response Time y WS-Security Username Token. En la Figura 8 se puede ver un ejemplo de lo mencionado.

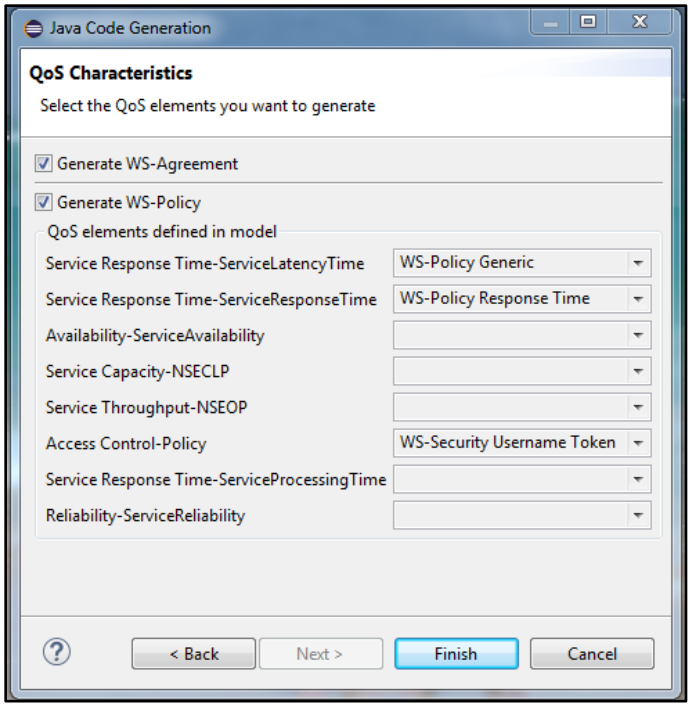

Figura 8. Cuarto paso del wizard (opcional) - Selección de WS-Agreement y WS-Policy

Se analizaron los elementos del dominio que componen el modelo SoaML con QoS y se estudió el archivo de intercambio de modelos en formato XMI. De este análisis se obtuvo el nuevo modelo de dominio que extiende el realizado en el proyecto SoaML2Code con elementos QoS. Este modelo de dominio se ilustra ver en la Figura 9, en donde los elementos resaltados en azul corresponden a los agregados en este proyecto.

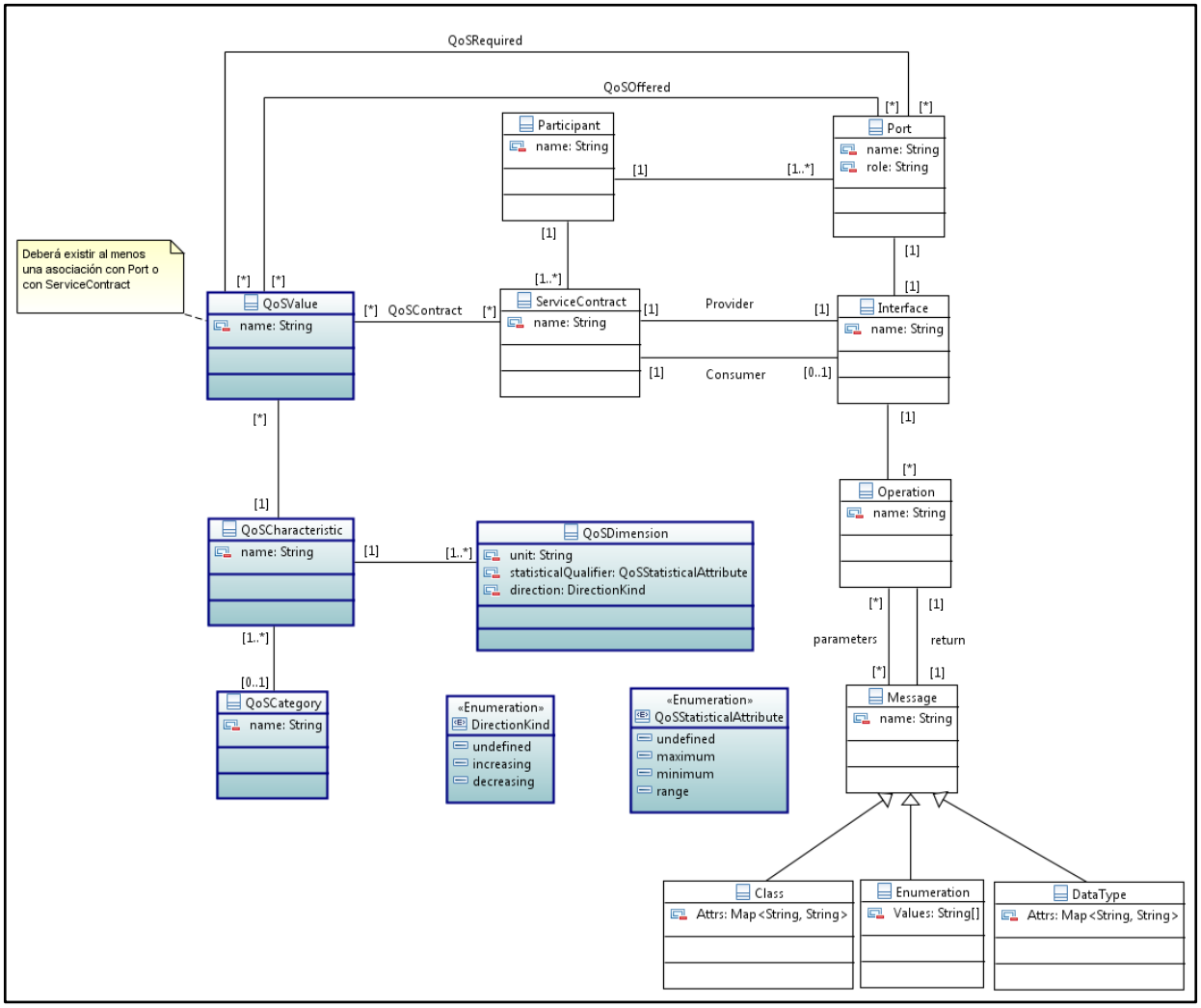

Figura 9. Modelo de dominio SoaML+QoS

El parseo existente del archivo XMI que representa el modelo SoaML se modificó para incluir los elementos de QoS que pueden aparecer o no en dicho modelo. Cabe destacar que el formato del archivo XMI a procesar debe cumplir con la estructura general presentada en la Figura 10. En particular, dentro del paquete QoS (que debe aparecer en cuarta posición respecto a los demás paquetes) deben estar todos los elementos QoS del modelo. Para generarlo de esta forma se modificó también la exportación del archivo XMI del modelo SoaML para incluir el paquete QoS y sus elementos.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI>
    <uml:Model name="PatientMASProcess" xmi:id="0">
        <packagedElement name="Participants">
            <packagedElement name="Participant1"> ... </packagedElement>
            ..
            <packagedElement name="ParticipantN"> ... </packagedElement>
        </packagedElement>
        <packagedElement name="Messages">
            <packagedElement name="Message1"/>
            ..
            <packagedElement name="MessageN"/>
        </packagedElement>
        <packagedElement name="Services">
            <packagedElement name="Service1">
                <packagedElement>
                    <ownedOperation name="Operation1">
                        <ownedParameter name="parameter"/>
                    </ownedOperation>
                </packagedElement>
                ..
                <packagedElement>
                    <ownedOperation name="OperationN">
                        <ownedParameter name="parameter"/>
                    </ownedOperation>
                </packagedElement>
            </packagedElement>
            ..
            <packagedElement name="ServiceN">
                <packagedElement>
                    <ownedOperation name="Operation1">
                        <ownedParameter name="parameter"/>
                    </ownedOperation>
                </packagedElement>
                ..
                <packagedElement>
                    <ownedOperation name="OperationN">
                        <ownedParameter name="parameter"/>
                    </ownedOperation>
                </packagedElement>
            </packagedElement>
        </packagedElement>
        <packagedElement name="QoS">
            ...
        </packagedElement>
        ..
</xmi:XMI>
```

Figura 10. Estructura general archivo XMI entrada del parser

Luego de tener el archivo parseado, resta especificar el modelo Java de correspondencia con el modelo SoaML+QoS que fue definido, es decir, la transformación de los objetos del modelo SoaML+QoS a sus correspondientes objetos Java que generan la implementación del modelo. Para simplificar el código de QoS y la recorrida de los distintos elementos al generar el código JEE y WS, se utilizó la estructura de clases que se puede ver en la Figura 11. El mapeo de los elementos del modelo SoaML+QoS a los objetos Java se puede ver en la Figura 12.

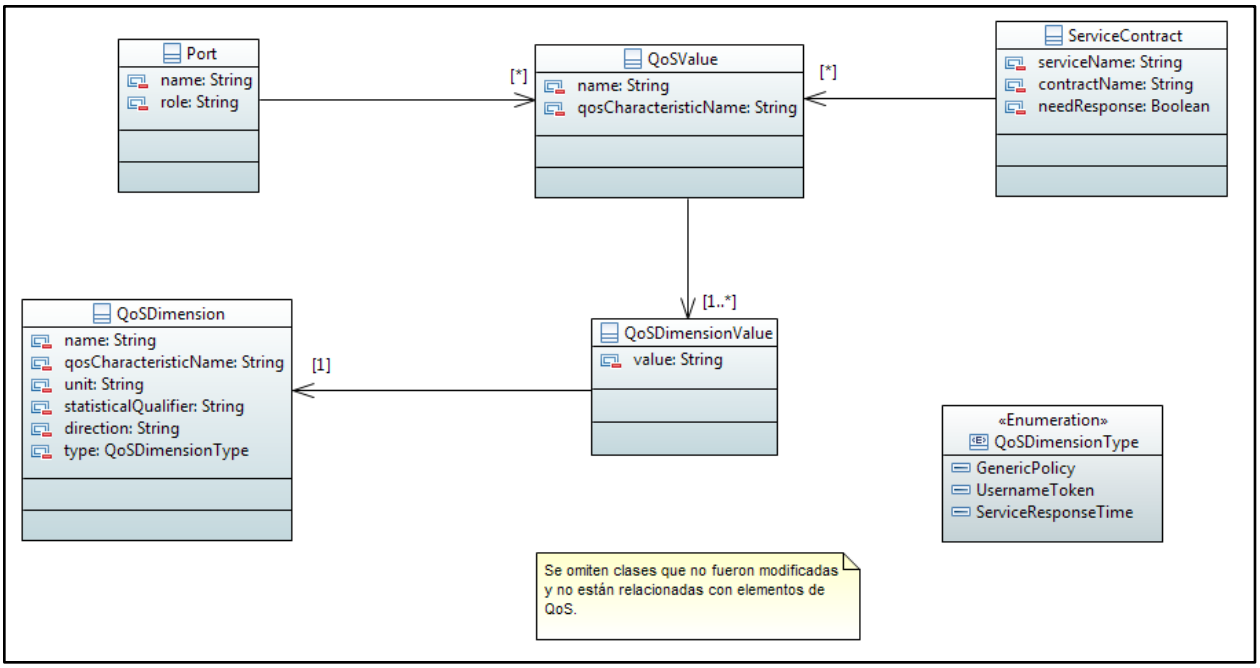

Figura 11. Diagrama de clases

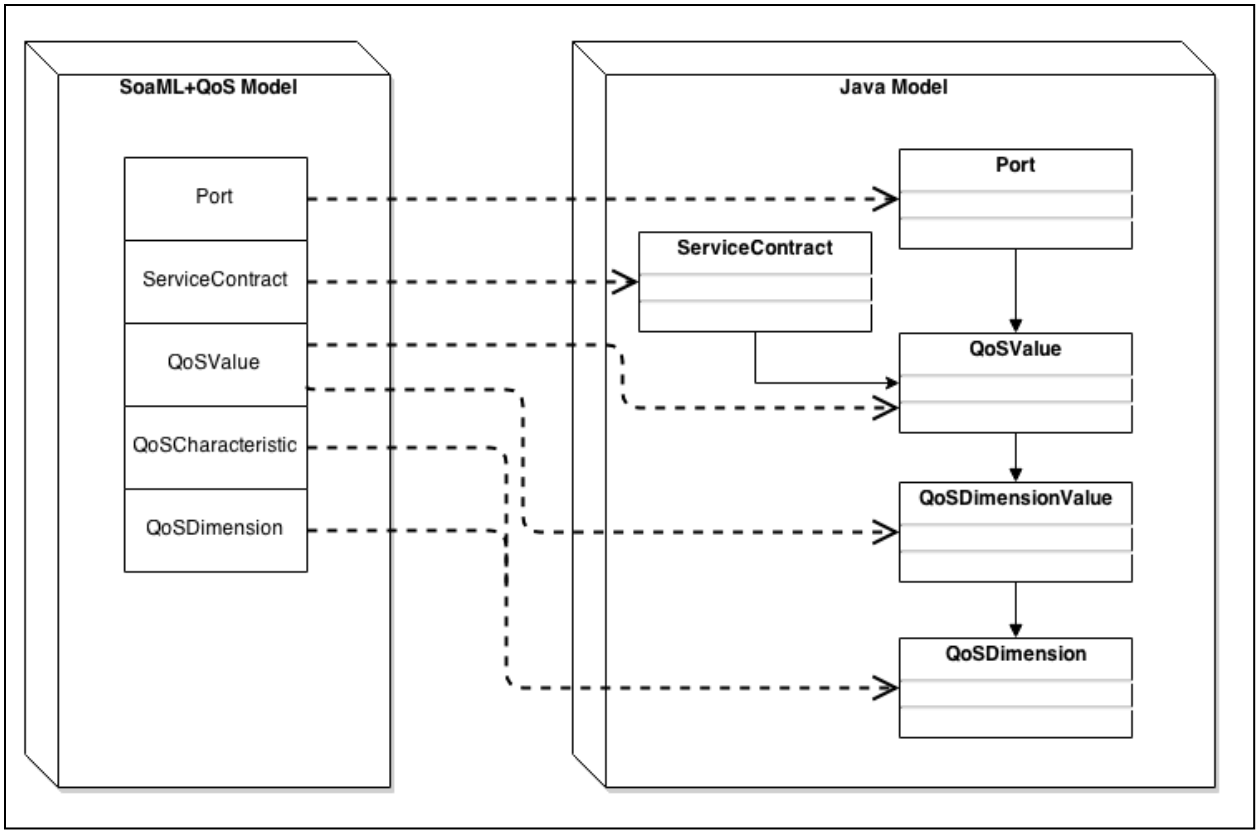

Figura 12. Mapeo de elementos de modelo SoaML+QoS a objetos Java

Para la generación de código de características de calidad se implementaron cuatro opciones:

- WS-Agreement para los contratos de servicios.

- WS-Policy Generic, la cual permite la generación de políticas genéricas a partir de las características definidas.
- WS-Policy Response Time, la cual permite el registro de los tiempos de los servicios.
- WS-Security Username Token, la cual permite el control de acceso a los servicios.

En la Tabla 1 se muestra qué tipo de política se genera para los distintos casos de generación de código. En la generación de servidores EJB, los archivos WSDL contienen las políticas de calidad y de control de acceso. Las clases Java necesarias para validar el token de seguridad o la configuración para ignorar las políticas de calidad cuando se recibe un mensaje SOAP son altamente dependientes del framework y del servidor donde serán desplegados los servicios. Por este motivo y dado que el proyecto EJB genera una biblioteca que puede ser referenciada desde proyectos web con distintas implementaciones y desplegados sobre distintos servidores, decidimos no generar las clases de validación de Username Token y configuración de WS-Policy.

Tabla 1. WS-Policies y Tecnologías

QoS	EJB	JAX-WS RI JBoss	JAX-WS RI Tomcat	JAX-WS + Spring Tomcat
WS-Policy Generic	Sólo wsdl	✓	✓	✓
WS-Policy Response Time	Sólo wsdl	✓	✓	✓
WS-Security Username Token	Sólo wsdl	✓	✓	✓
WS-Agreement	✓	✓	✓	✓

En primer lugar presentaremos por tipo de QoS la generación realizada según la correspondencia con las clases Java presentada previamente. En segundo lugar presentaremos para cada columna de la Tabla 1 correspondiente a la tecnología destino de la generación, los detalles de la misma.

3.1.Generación WS-Agreement

Para generar documentos WS-Agreement, para cada participante generado se obtienen todos los elementos del tipo Service Contract que contengan al participante (ya sea como proveedor o consumidor) y que estén asociados a al menos un elemento QoSValue. Para cada uno de los contratos, se genera un documento WS-Agreement que contiene parte de la información del contrato y del servicio implementado, y en donde se especifican las características de calidad asociadas al contrato. El contrato para el cual el participante esté como proveedor se genera en el proyecto del servidor, y en el proyecto del cliente se genera el contrato para el cual el participante esté como consumidor. La Figura 13 muestra la transformación entre los objetos Java y las distintas partes del documento WS-Agreement generado.

Figura 13. Mapeo de objetos Java a componentes del documento WS-Agreement

3.2.Generación WS-Policy

Para generar WS-Policy, el archivo WSDL es modificado para incluir las distintas políticas disponibles. Para WS-Policy Generic y WS-Policy Response Time, el resultado en el archivo WSDL es el mismo. Por cada QoSDimension dentro de un QoSValue asociado al puerto de servicio, se genera una política WS-Policy cuyo nombre es el de la dimensión de calidad. Esta transformación se ilustra en la Figura 14.

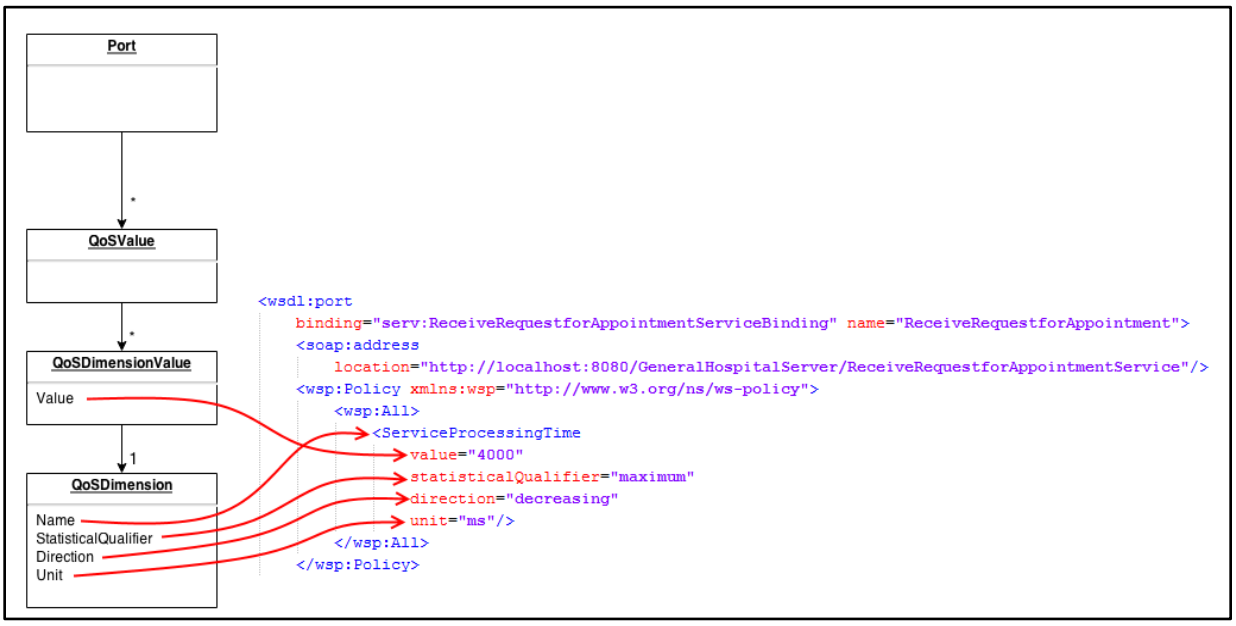

Figura 14. Mapeo de objetos Java a políticas genéricas en el WSDL

3.3.Generación WS-Security

Cuando el usuario indica que determinada dimensión de calidad se implemente como WS-Security UsernameToken, en el WSDL se agrega una política de seguridad UsernameToken, sin importar el valor definido en el elemento QoSValue ni las propiedades de la dimensión de calidad. En la Figura 15 se observa el resultado de la generación del WSDL con WS-Security UsernameToken.

```
<wsdl:service name="ReceiveSurgerydateReservationService">
    <wsdl:port
        binding="serv:ReceiveSurgerydateReservationServiceBinding" name="ReceiveSurgerydateReservation">
        <soap:address location="http://localhost:8080/PatientServer/ReceiveSurgerydateReservationService"/>
        <wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy">
            <wsp:All>
                <sp:SupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
                    <wsp:Policy>
                        <sp:UsernameToken sp:IncludeToken=
                        "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                            <wsp:Policy>
                                <sp:WssUsernameToken11/>
                            </wsp:Policy>
                        </sp:UsernameToken>
                    </wsp:Policy>
                </sp:SupportingTokens>
            </wsp:All>
        </wsp:Policy>
    </wsdl:port>
</wsdl:service>
```

Figura 15. Ejemplo de policy Username Token

3.4.Generación de políticas combinadas

Si se generan distintos tipos de políticas, éstas se anidan como se muestra en la Figura 16.

```
<wsdl:service name="ReceiveSurgerydateReservationService">
    <wsdl:port
        binding="serv:ReceiveSurgerydateReservationServiceBinding" name="ReceiveSurgerydateReservation">
        <soap:address location="http://localhost:8080/PatientServer/ReceiveSurgerydateReservationService"/>
        <wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy">
            <wsp:All>
                <ServiceLatencyTime direction="decreasing" unit="ms" value="10"/>
                <sp:SupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
                    <wsp:Policy>
                        <sp:UsernameToken sp:IncludeToken=
                        "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                            <wsp:Policy>
                                <sp:WssUsernameToken11/>
                            </wsp:Policy>
                        </sp:UsernameToken>
                    </wsp:Policy>
                </sp:SupportingTokens>
            </wsp:All>
        </wsp:Policy>
    </wsdl:port>
</wsdl:service>
```

Figura 16. Ejemplo de varias policies

La generación de código depende directamente de qué tipo de proyecto se vaya a generar y qué implementaciones de características de calidad. A continuación describimos para cada tecnología los detalles de la generación realizada.

3.5.Generación para Dynamic Web Project - JAX-WS ri

En la versión anterior de SoaML2Code se generaba el mismo código ya sea para JBoss como para Tomcat. Con la nueva versión de JBoss utilizada hay elementos que ya no son necesarios incorporar al proyecto generado, por lo que se optó por dividir las opciones para que el usuario pueda decidir qué servidor desea utilizar y así generar códigos diferentes.

3.5.1. Generación para JBoss

Con la versión 7.1.1 de JBoss ya no resulta necesario agregar las librerías de JAX-WS ya que éstas ya vienen incluidas. Tampoco es necesario agregar los servlets al archivo web.xml como se hacía anteriormente.

Las WS-Policy Generic que generamos sirven para describir las características de calidad de los servicios y no para restringir aspectos en el intercambio de mensajes. Por este motivo es que se le debe instruir al servidor que ignore esas políticas al momento de recibir una invocación. Para esto se incorpora al código del proyecto las clases *IgnorablePolicyInterceptorProvider* y *PolicyInterceptorProviderInstallerInterceptor*. La primera extiende *AbstractPolicyInterceptorProvider* y es la que se encarga de marcar la política como verificada. La segunda extiende *AbstractPhaseInterceptor<Message>* y es la que se encarga de registrar las políticas genéricas definidas junto con el interceptor que las ignora. Para que esto funcione, se debe anotar la interfaz como se muestra en la Figura 17.

```
@InInterceptors(interceptors = {"qos.PolicyInterceptorProviderInstallerInterceptor"})
```

Figura 17. Interfaz anotada con interceptor

Para que el proyecto compile se agregan al *buildpath* las librerías cxf-api-2.4.6.jar, cxf-rt-ws-policy-2.4.6.jar y cxf-common-utilities-2.4.6.jar contenidas en la instalación del JBoss. Para el runtime se crea el archivo de configuración jboss-deployment-structure.xml en donde se agregan las dependencias a librerías incluidas en JBoss. En este caso se agrega la dependencia al módulo org.apache.cxf, como se muestra en la Figura 18.

```
<dependencies>
    <module name="org.apache.cxf" />
</dependencies>
```

Figura 18. Dependencia org.apache.cxf

Si se selecciona WS-Policy Response Time lo generado es lo mismo que para WS-Policy generic, lo que difiere es lo que se genera en el cliente.

Al seleccionar WS-Security Username Token se crea la clase *ServerPasswordCallback* la cual implementa *CallbackHandler* y es quien se encarga de validar que el usuario y contraseña enviados por el cliente son correctos. Para añadir el *handler* a los servicios se crea el archivo jaxws-endpoint-config.xml que tiene el formato mostrado en la Figura 19.

```
<?xml version="1.0" encoding="UTF-8"?>
<jaxws-config xmlns="urn:jboss:jbossws-jaxws-config:4.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:javaee="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="urn:jboss:jbossws-jaxws-config:4.0 schema/jbossws-jaxws-config_4_0.xsd">
    <endpoint-config>
        <config-name>Custom WS-Security Endpoint</config-name>
        <property>
            <property-name>ws-security.callback-handler</property-name>
            <property-value>qos.ServerPasswordCallback</property-value>
        </property>
    </endpoint-config>
</jaxws-config>
```

Figura 19. Archivo jaxws-endpoint-config.xml

La clase que implementa el web service se anota para que sea invocado el *handler* como se puede ver en la Figura 20.

```
@org.jboss.ws.api.annotation.EndpointConfig(configFile = "WEB-INF//jaxws-endpoint-config.xml",
                                            configName = "Custom WS-Security Endpoint")
```

Figura 20. Anotación en clase que implementa servicio

Para que el proyecto compile se agregan al *buildpath* las librerías wss4j-1.6.5.jar y jbossws-api-1.0.0.GA.jar contenidas en la instalación del JBoss. Para el runtime se crea el archivo de configuración jboss-deployment-structure.xml en donde se agregan las dependencias a librerías incluidas en JBoss. En este caso se agrega la dependencia a los módulos org.apache.ws.security y org.jboss.ws.api, como se muestra en la Figura 21.

```
<dependencies>
    <module name="org.apache.ws.security" />
    <module name="org.jboss.ws.api" />
</dependencies>
```

Figura 21. Dependencias org.apache.ws.security y org.jboss.ws.api

Para la correcta generación de este tipo de proyecto es necesario tener instalado el server JBoss en el Eclipse y haber agregado el plugin de JAX-WS ri a la carpeta plugins de Eclipse. Para que el proyecto compile se debe crear la variable JBOSS_HOME en Eclipse apuntando a la instalación de JBoss (Ver Anexo Guía de Instalación).

3.5.2. Generación para Tomcat

Para que funcionen las políticas genéricas no fue necesario ninguna configuración especial.

Para WS-Security Username Token se crea la clase *ServerPasswordCallback*, la cual es igual a la utilizada para JBoss, a diferencia del *import* de la clase WSPasswordCallback. En JBoss dicha clase se obtiene del paquete *org.apache.ws.security* mientras que en Tomcat se obtiene de *org.apache.wss4j.common.ext*. También se crea la clase WSUsernameTokenValidator, la cual implementa *SOAPHandler<SOAPMessageContext>* y es la encargada de validar los los mensajes SOAP entrantes mediante la invocación a la clase WSPasswordCallback.

Por otro lado, se crea el archivo *handler.xml*, el cual configura que todos los mensajes SOAP sean interceptados por WSUsernameTokenValidator. Éste tiene el formato que se muestra en la Figura 22.

```
<?xml version="1.0" encoding="UTF-8"?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/javaee_web_services_metadata_handler_2_0.xsd">
    <handler-chain>
        <handler>
            <handler-name>WSUsernameTokenValidator</handler-name>
            <handler-class>qos.WSUsernameTokenValidator</handler-class>
        </handler>
    </handler-chain>
</handler-chains>
```

Figura 22. Archivo handler.xml

La clase que implementa el web service se anota para que sea invocado el *handler* como se puede ver en la Figura 23.

```
@javax.jws.HandlerChain(file = "./qos/handlers.xml")
```

Figura 23. Anotación HandlerChain en clase que implementa servicio

Por último, se agregan las bibliotecas de WSS4J y sus bibliotecas referenciadas necesarias para que compile y funcione el servicio en el servidor, como se puede ver en la Figura 24. WSS4J es una biblioteca que provee una implementación de la mayoría de estándares WS-Security, y es la utilizada en los proyectos generados con JAX-WS ri con Tomcat para procesar los Username Tokens.

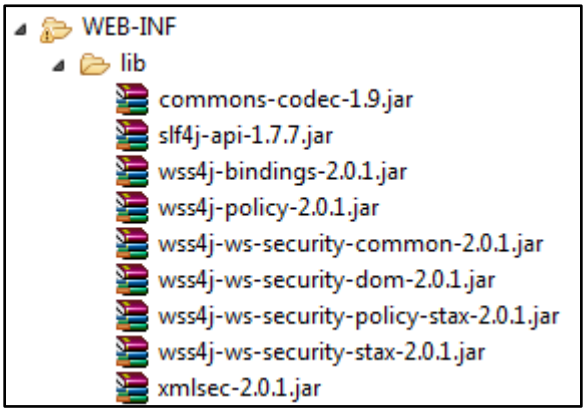

Figura 24. Bibliotecas necesarias para wss4j

3.6.Generación para Dynamic Web Project - JAX-WS + spring

Se actualizó la versión de Apache CXF a la 3.0.1 lo cual implicó algunos cambios en el archivo beans.xml.

Para que funcione WS-Policy Generic el servidor debe, al igual que para JBoss con JAX-WS ri, ignorar la política cuando el servicio es invocado. Para esto se modifica el beans.xml, agregando el interceptor provisto por CXF *org.apache.cxf.ws.policy.IgnorablePolicyInterceptorProvider*, como se puede ver en el ejemplo de la Figura 25.

```
<bean class="org.apache.cxf.ws.policy.IgnorablePolicyInterceptorProvider">
    <constructor-arg>
        <list>
            <bean class="javax.xml.namespace.QName">
                <constructor-arg value="ServiceLatencyTime"/>
            </bean>
        </list>
    </constructor-arg>
</bean>
```

Figura 25. Bean IgnorablePolicyInterceptorProvider

De la misma forma que para JBoss, si se selecciona WS-Policy Response Time lo generado es lo mismo que para WS-Policy generic. Lo que difiere es lo que se genera en el cliente, lo que se explicará más adelante.

Para WS-Security Username Token se crea la clase ServerPasswordCallback, la misma utilizada para el proyecto Dynamic Web Project - JAX-WS ri para Tomcat. Para añadir el handler a los servicios también se debe utilizar la clave ws-security.callback-handler pero en este caso se configura en el beans.xml, como se muestra en la Figura 26.

```
<jaxws:properties>
    <entry key="ws-security.callback-handler" value="qos.ServerPasswordCallback"/>
    <entry key="ws-security.enable.nonce.cache" value="false"/>
</jaxws:properties>
```

Figura 26. Extracto de archivo beans.xml

3.7.Generación del Cliente para consumo del WS

El cliente se genera de forma independiente de la tecnología seleccionada para el servidor.

Para WS-Policy Generic no se debe generar nada específico del lado del cliente ya que el servidor no espera nada e incluso ignora la política.

Para WS-Policy Response Time se registran en un archivo de log los tiempos de inicio y fin del método que invoca al servicio. Esto permite que se pueda comprobar si los tiempos de respuesta cumplen con las políticas de cada servicio.

Para esto se agregó la librería *log4j2* al cliente generado junto con el archivo de configuración log4j2.xml en el cual se puede definir el archivo en el que se quieren imprimir los logs y el formato de éste, entre otras cosas.

Para WS-Security Username Token se crea la clase UsernameTokenUtil, la cual se encarga de agregar el usuario y contraseña al cabezal SOAP. Esta es invocada desde la clase main del cliente.

Para finalizar, se muestra en la Tabla 2 un resumen de las plataformas tecnológicas de destino que el plugin provee.

Tabla 2. Tecnologías destino del generador de código

Descripción de servicios web	WSDL 1.1 (SOAP 1.1)
	XML Schema 1.1
	WS-Policy
	WS-Security y WS-SecurityPolicy
Contratos de calidad de servicios web	WS-Agreement
Java EE	JDK 7 o superior
	EJB 3.1
	JAX-WS ri 2.2.8
	JAX-WS + Spring 2.0
Entorno de desarrollo	Eclipse Web Tools

Proyecto de grado

Generación automática de Arquitecturas Orientadas a Servicios (SOAs) con SoaML y especificación de QoS

Casos de Prueba

Federico Bertolini, Sofía Pérez, María Noel Quiñones

Tutor
Andrea Delgado

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo – Uruguay
Diciembre 2014

Resumen

En este documento se exponen las distintas pruebas realizadas y los resultados obtenidos sobre la versión final de los diagramas QoS y SoaML+QoS, y el generador de código Java desarrollados durante el transcurso del proyecto. Como hemos actualizado los diagramas SoaML para ser utilizados en la versión 1.0 de Papyrus en Eclipse Luna, también hemos realizado pruebas para verificar el correcto funcionamiento de los diagramas SoaML.

Contenido

1. Iteración 1

1.1.Casos de uso de modelado

1.1.1. Crear Diagrama de Características QoS

Permite al usuario crear un diagrama de tipo *QoS Characteristic* agregando elementos al lienzo desde la paleta de herramientas. Este diagrama permite definir las distintas características de calidad, las dimensiones que cuantifican dichas características así como también la categoría a la cual pertenece y las asociaciones que pueden haber entre distintas características. Se podrá agregar elementos tales como características, dimensiones y categorías así como también asociaciones y comentarios. No se validará que las asociaciones entre elementos sean conceptualmente correctas y consistentes con los perfiles SoaML y QoS.

1.1.1.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - Crear diagrama de tipo *QoS Characteristic Diagram.*	Flujo Principal	1
E2 - Agregar elemento *QoSCategory.*	Flujo Principal	2.1
E3 - Agregar elemento *QoSCharacteristic* dentro de un elemento *QoSCategory.*	Flujo Principal	2.2
E4 - Agregar elemento *QoSCharacteristic* en el lienzo	Flujo Alternativo	2.2a.1
E5 - Agregar elemento *QoSDimension.*	Flujo Principal	2.3
E6 - Agregar una generalización entre dos elementos *QoSCharacteristic*.	Flujo Principal	2.4

1.1.1.2. Condiciones

Condición	Detalles
C1	Agregar elemento al lienzo.
C2	Agregar elemento dentro de un elemento QosCharacteristic.
C3	Definir tipo de datos.
C4	Definir valor en los atributos de QoSDimension.
C5	No definir valor en los atributos de QoSDimension.
C6	Dentro de un proyecto QoS.
C7	Dentro de un proyecto SoaML + QoS.

1.1.1.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C6	Existe un proyecto QoS	Se crea un diagrama del tipo QoS Characteristic. El diagrama despliega la paleta de herramientas conteniendo los elementos que se pueden agregar a un diagrama de este tipo.	Esperada.
E2 - C6, C1		Se agrega un QoSCategory con el estereotipo «qoSCategory».	Esperada.

E3 - C6	Existe un elemento QoSCategory	Se agrega un QoSCharacteristic con el estereotipo «qoSCharacteristic» dentro del elemento QoSCategory.	Esperada.
E4 - C6		Se agrega un QoSCharacteristic con el estereotipo «qoSCharacteristic».	Esperada.
E5 - C2, C5, C6	Existe un elemento QoSCharacteristic	Se agrega un elemento QoSDimension con el estereotipo «qoSDimension» dentro del QoSCharacteristic. El tipo de dato del elemento es Undefined. Los atributos statisticalQualifier y direction tienen valor *undefined*. El atributo unit tiene valor *null*.	Esperada.
E5 - C2, C3, C5, C6	Existe un elemento QoSCharacteristic	Se agrega un elemento QoSDimension con el estereotipo «qoSDimension» dentro del QoSCharacteristic. El tipo de dato del elemento es el que se seleccionó. Los atributos statisticalQualifier y direction tienen valor *undefined*. El atributo unit tiene valor *null*.	Esperada.
E5 - C2, C3, C4, C6	Existe un elemento QoSCharacteristic	Se agrega un elemento QoSDimension con el estereotipo «qoSDimension» dentro del QoSCharacteristic. El tipo de dato del elemento es el que se seleccionó. Los atributos statisticalQualifier, direction y unit tienen los valores seleccionados.	Esperada.
E5 - C6, C1		No agrega el elemento al lienzo	Esperada.
E6 - C6	Existen dos elementos QoSCharacteristic	Se agrega la asociación entre los dos elementos QoSCharacteristic.	Esperada.
E1 - C7	Existe un proyecto QoS	Se crea un diagrama del tipo QoS Characteristic. El diagrama despliega la paleta de herramientas conteniendo los elementos que se pueden agregar a un diagrama de este tipo.	Esperada.
E2 - C7		Se agrega un QoSCategory con el estereotipo «qoSCategory».	Esperada.
E3 - C7	Existe un elemento QoSCategory	Se agrega un QoSCharacteristic con el estereotipo «qoSCharacteristic»	Esperada.

		dentro del elemento QoSCategory.	
E4 - C7		Se agrega un QoSCharacteristic con el estereotipo «qoSCharacteristic».	Esperada.
E5 - C2, C5, C7	Existe un elemento QoSCharacteristic	Se agrega un elemento QoSDimension con el estereotipo «qoSDimension» dentro del QoSCharacteristic. El tipo de dato del elemento es Undefined. Los atributos statisticalQualifier y direction tienen valor *undefined*. El atributo unit tiene valor *null*.	Esperada.
E5 - C2, C3, C5, C7	Existe un elemento QoSCharacteristic	Se agrega un elemento QoSDimension con el estereotipo «qoSDimension» dentro del QoSCharacteristic. El tipo de dato del elemento es el que se seleccionó. Los atributos statisticalQualifier y direction tienen valor *undefined*. El atributo unit tiene valor *null*.	Esperada.
E5 - C2, C3, C4, C7	Existe un elemento QoSCharacteristic	Se agrega un elemento QoSDimension con el estereotipo «qoSDimension» dentro del QoSCharacteristic. El tipo de dato del elemento es el que se seleccionó. Los atributos statisticalQualifier, direction y unit tienen los valores seleccionados.	Esperada.
E5 - C1, C7		No agrega el elemento al lienzo	Esperada.
E6 - C7	Existen dos elementos QoSCharacteristic	Se agrega la asociación entre los dos elementos QoSCharacteristic.	Esperada.

1.1.1.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para crear diagrama de Características QoS.

1.1.2. Crear diagrama de Contrato de Servicios con QoS

Permite al usuario crear un diagrama de tipo *Service Contract with QoS*, agregando elementos al lienzo desde la paleta de herramientas del diagrama. Se pueden agregar elementos tales como contratos de servicios, roles y usos de contratos de servicios. El usuario también puede agregar valores de características de calidad de servicios, y asociaciones entre éstos y los contratos de servicios.

1.1.2.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - Agregar un elemento QoSValue	Flujo Principal	2.1
E2 - Arrastrar un elemento QoSCharacteristic desde el	Flujo Principal	2.2

árbol del modelo hasta un elemento QoSValue sin una QoSCharacteristic asociada.		
E3 - Agregar un elemento Slot dentro de un elemento QoSValue, especificando un valor y la propiedad que implementa el Slot.	Flujo Principal	2.3
E4 - Agregar una dependencia de tipo QoSContract entre un contrato de servicio y un elemento QoSValue previamente definidos.	Flujo Principal	2.4

1.1.2.2. Condiciones

Condición	Detalles
C1	Especificar qué QoSCharacteristic implementa.
C2	No especificar qué QoSCharacteristic implementa.
C3	Seleccionar una QoSDimension.
C4	No seleccionar ninguna QoSDimension.

1.1.2.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1	Existe un elemento QoSCharacteristic	Se agrega el elemento QoSValue en el lienzo, al cual se puede modificar el nombre. Se muestra correctamente el nombre del QoSCharacteristic que implementa	Esperada.
E1 - C2		Se agrega el elemento QoSValue en el lienzo, al cual se puede modificar el nombre.	Esperada.
E2 - C3	Existe un elemento QoSValue que no implemente una QosCharacteristic. Existe un elemento QoSCharacteristic con al menos una QoSDimension.	El elemento QoSValue implementa la QoSCharacteristic. En el árbol del modelo, se puede ver que el QoSValue tiene slots relacionados a los QoSDimension especificadas.	Esperada.
E2 - C4	Existe un elemento QoSValue que no implemente una QosCharacteristic. Existe un elemento QoSCharacteristic.	El elemento QoSValue implementa la QoSCharacteristic. En el árbol del modelo, se puede ver que el QoSValue no tiene ningún slot.	Esperada.
E3	Existe un elemento QoSValue. Existe un elemento QoSCharacteristic con al menos un QoSDimension.	Se agrega el slot en el QoSValue, asociado a la QoSDimension seleccionada. Se permite agregar un valor en el slot.	Esperada.

E4	Existe un elemento ServiceContract y un elemento QoSValue.	Se agrega la dependencia del tipo QoSContract entre el elemento Service contract y el QoSValue.	Esperada.

1.1.2.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para crear diagrama de Contrato de Servicios con QoS.

Se observa un comportamiento no favorable en Papyrus cuando se define un elemento QoSValue arrastrando el elemento QoSCharacteristic y seleccionando una QoSDiemension: los Slots definidos en la QoSDimension seleccionada no se ven dentro del elemento QoSValue en el diagrama pero sí se ven definidos dentro del elemento en el árbol del modelo. Para que aparezcan dentro del elemento QoSValue en el diagrama alcanza con arrastrar los Slots desde el árbol del modelo.

1.1.3. Importar modelo de Características QoS

Permite al usuario importar un modelo de características QoS a un modelo SoaML+QoS, realizando una copia en profundidad de todos los elementos del modelo origen al modelo destino.

1.1.3.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - El usuario selecciona el menú “Import QoS Elements from QoS Model” y selecciona un archivo UML.	Flujo Principal	1, 2

1.1.3.2. Condiciones

Condición	Detalles
C1	El archivo UML seleccionado corresponde a un modelo QoS con elementos.
C2	El archivo UML seleccionado corresponde a un modelo QoS vacío.
C3	El archivo UML seleccionado no corresponde a un modelo QoS.

1.1.3.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1	Existe un modelo QoS con elementos.	Se copian todos los elementos del modelo seleccionado al árbol de elementos del modelo actual.	Esperada.
E1 - C2	Existe un modelo QoS sin elementos.	El árbol de elementos del modelo actual no tiene cambios.	Esperada.
E1 - C3	Existe un modelo que no sea QoS.	Se despliega un mensaje de error indicando que el modelo seleccionado no es un modelo QoS.	No esperada. Se seleccionó un modelo SoaML+QoS y se importaron todos los elementos del

			modelo SoaML+QoS. No hay mensaje de error.

1.1.3.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para importar un modelo de Características QoS si el modelo efectivamente es de QoS.

En el caso que se importe un modelo que no es QoS, no se despliega un mensaje de error y se hace la copia de todos los elementos que estén definidos en el modelo. Este error se solucionó para la siguiente iteración.

1.1.4. Crear Diagrama de Participantes como Clases con QoS

Permite al usuario crear un diagrama de tipo *Participant Class with QoS*. Es una especificación a alto nivel en la que se pueden agregar al lienzo elementos participante y sus puertos y sus comportamientos asociados. El usuario también puede agregar características y valores de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios.

1.1.4.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - Agregar un elemento QoSValue.	Flujo Principal	2.1
E2 - Arrastrar un elemento QoSCharacteristic a un elemento QoSValue.	Flujo Principal	2.2
E3 - Agregar un elemento Slot a un elemento QoSValue.	Flujo Principal	2.3
E4 - Agregar una dependencia del tipo QoSRequired entre un Request y un QoSValue.	Flujo Principal	2.4
E5 - Agregar una dependencia del tipo QoSOffered entre un Service y un QoSValue.	Flujo Principal	2.5

1.1.4.2. Condiciones

Condición	Detalles
C1	Especificar qué QoSCharacteristic implementa.
C2	No especificar qué QoSCharacteristic implementa.
C3	Seleccionar una QoSDimension.
C4	No seleccionar ninguna QoSDimension.

1.1.4.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1	Existe un elemento QoSCharacteristic	Se agrega el elemento QoSValue en el lienzo, al cual se puede modificar el nombre. Se muestra correctamente el nombre del QoSCharacteristic que implementa.	Esperada.

E1 - C2		Se agrega el elemento QoSValue en el lienzo, al cual se puede modificar el nombre.	Esperada.
E2 - C3	Existe un elemento QoSValue que no implemente una QosCharacteristic. Existe un elemento QoSCharacteristic con al menos una QoSDimension.	El elemento QoSValue implementa la QoSCharacteristic. En el árbol del modelo, se puede ver que el QoSValue tiene slots relacionados a los QoSDimension especificadas.	Esperada.
E2 - C4	Existe un elemento QoSValue que no implemente una QosCharacteristic. Existe un elemento QoSCharacteristic.	El elemento QoSValue implementa la QoSCharacteristic. En el árbol del modelo, se puede ver que el QoSValue no tiene ningún slot.	Esperada.
E3	Existe un elemento QoSValue. Existe un elemento QoSCharacteristic con al menos un QoSDimension.	Se agrega el slot en el QoSValue, asociado a la QoSDimension seleccionada. Se permite agregar un valor en el slot.	Esperada.
E4	Existe un elemento Request y un elemento QoSValue.	Se agrega una dependencia del tipo QoSRequired entre el elemento Request y el QoSValue.	Esperada.
E5	Existe un elemento Service y un elemento QoSValue.	Se agrega una dependencia del tipo QoSOffered entre el elemento Service y el QoSValue.	Esperada.

1.1.4.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para crear diagrama de Participantes como Clases con QoS.

Se observa un comportamiento no favorable en Papyrus cuando se define un elemento QoSValue arrastrando el elemento QoSCharacteristic y seleccionando una QoSDiemension: los Slots definidos en la QoSDimension seleccionada no se ven dentro del elemento QoSValue en el diagrama pero sí se ven definidos dentro del elemento en el árbol del modelo. Para que aparezcan dentro del elemento QoSValue en el diagrama alcanza con arrastrar los Slots desde el árbol del modelo.

1.1.5. Crear Diagrama de Participantes como Componentes con QoS

Permite al usuario crear un diagrama de tipo *Participant Component with QoS* que indica la implementación de participantes como componentes. Para esto el usuario puede agregar elementos al lienzo desde la paleta de herramientas del diagrama. Se pueden agregar componentes participante y

dentro de ellos elementos tales como proveedores de servicios, puertos, asociaciones y canales de servicio entre los puertos, capacidades y sus dependencias con los participantes existentes en el diagrama, usos de arquitecturas de servicios y los correspondientes rolebindings con los proveedores de servicios asociados. El usuario también puede agregar características y valores de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios.

1.1.5.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - Agregar un elemento QoSValue.	Flujo Principal	2.1
E2 - Arrastrar un elemento QoSCharacteristic a un elemento QoSValue.	Flujo Principal	2.2
E3 - Agregar un elemento Slot a un elemento QoSValue.	Flujo Principal	2.3
E4 - Agregar una dependencia del tipo QoSRequired entre un Request y un QoSValue.	Flujo Principal	2.4
E5 - Agregar una dependencia del tipo QoSOffered entre un Service y un QoSValue.	Flujo Principal	2.5

1.1.5.2. Condiciones

Condición	Detalles
C1	Especificar qué QoSCharacteristic implementa.
C2	No especificar qué QoSCharacteristic implementa.
C3	Seleccionar una QoSDimension.
C4	No seleccionar ninguna QoSDimension.

1.1.5.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1	Existe un elemento QoSCharacteristic.	Se agrega el elemento QoSValue en el lienzo, al cual se puede modificar el nombre. Se muestra correctamente el nombre del QoSCharacteristic que implementa.	Esperada.
E1 - C2		Se agrega el elemento QoSValue en el lienzo, al cual se puede modificar el nombre.	Esperada.
E2 - C3	Existe un elemento QoSValue que no implemente una QosCharacteristic. Existe un elemento QoSCharacteristic con al menos una QoSDimension.	El elemento QoSValue implementa la QoSCharacteristic. En el árbol del modelo, se puede ver que el QoSValue tiene slots relacionados a los QoSDimension especificadas.	Esperada.

E2 - C4	Existe un elemento QoSValue que no implemente una QosCharacteristic. Existe un elemento QoSCharacteristic.	El elemento QoSValue implementa la QoSCharacteristic. En el árbol del modelo, se puede ver que el QoSValue no tiene ningún slot.	Esperada.
E3	Existe un elemento QoSValue. Existe un elemento QoSCharacteristic con al menos un QoSDimension.	Se agrega el slot en el QoSValue, asociado a la QoSDimension seleccionada. Se permite agregar un valor en el slot.	Esperada.
E4	Existe un elemento Request y un elemento QoSValue.	Se agrega una dependencia del tipo QoSRequired entre el elemento Request y el QoSValue.	Esperada.
E5	Existe un elemento Service y un elemento QoSValue.	Se agrega una dependencia del tipo QoSOffered entre el elemento Service y el QoSValue.	Esperada.

1.1.5.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para crear diagrama de Participantes como Componentes con QoS.

Se observa un comportamiento no favorable en Papyrus cuando se define un elemento QoSValue arrastrando el elemento QoSCharacteristic y seleccionando una QoSDiemension: los Slots definidos en la QoSDimension seleccionada no se ven dentro del elemento QoSValue en el diagrama pero sí se ven definidos dentro del elemento en el árbol del modelo. Para que aparezcan dentro del elemento QoSValue en el diagrama alcanza con arrastrar los Slots desde el árbol del modelo.

1.1.6. Editar Diagrama de Características QoS

Permite al usuario editar un diagrama de tipo *QoS Characteristic* existente agregando, borrando y modificando elementos del diagrama.

1.1.6.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - Agregar elemento *QoSCategory.*	Flujo Principal	2.1
E2 - Agregar elemento *QoSCharacteristic* dentro de un elemento *QoSCategory.*	Flujo Principal	2.2
E3 - Agregar elemento *QoSCharacteristic* en el lienzo	Flujo Alternativo	2.2a.1
E4 - Agregar elemento *QoSDimension.*	Flujo Principal	2.3
E5 - Agregar una generalización entre dos elementos *QoSCharacteristic*.	Flujo Principal	2.4
E6 - Modificar un elemento *QoSDimension* de un elemento *QoSCharacteristic* existente.	Flujo Principal	2.5
E7 - Modificar un elemento *QoSCategory*.	Flujo Principal	2.6

E8 - Modificar un elemento *QoSCharacteristic*.	Flujo Principal	2.7
E9 - Borrar una generalización entre dos elementos *QoSCharacteristic.*.	Flujo Principal	2.8
E10 - Borrar un elemento *QoSCategory*.	Flujo Principal	2.9
E11 - Borrar un elemento *QoSCharacteristic*.	Flujo Principal	2.10
E12 - Borrar un elemento *QoSDimension* de una característica existente.	Flujo Principal	2.11

1.1.6.2. Condiciones

Condición	Detalles
C1	Agregar elemento al lienzo.
C2	Agregar elemento dentro de un elemento QosCharacteristic.
C3	Definir tipo de datos.
C4	Definir valor en los atributos de QoSDimension.
C5	No definir valor en los atributos de QoSDimension.
C6	Modifica valor en los atributos de QoSDimension.
C7	No existen referencias al elemento eliminado.
C8	Existen referencias al elemento eliminado.

1.1.6.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1		Se agrega un QoSCategory con el estereotipo «qoSCategory».	Esperada.
E2	Existe un elemento QoSCategory.	Se agrega un QoSCharacteristic con el estereotipo «qoSCharacteristic» dentro del elemento QoSCategory.	Esperada.
E3 - C1		Se agrega un QoSCharacteristic con el estereotipo «qoSCharacteristic».	Esperada.
E4 - C2, C5	Existe un elemento QoSCharacteristic.	Se agrega un elemento QoSDimension con el estereotipo «qoSDimension» dentro del QoSCharacteristic. El tipo de dato del elemento es Undefined. Los atributos statisticalQualifier y direction tienen valor *undefined*. El atributo unit tiene valor *null*.	Esperada.
E4 - C2, C3, C5	Existe un elemento	Se agrega un elemento	Esperada.

	QoSCharacteristic.	QoSDimension con el estereotipo «qoSDimension» dentro del QoSCharacteristic. El tipo de dato del elemento es el que se seleccionó. Los atributos statisticalQualifier y direction tienen valor *undefined*. El atributo unit tiene valor *null*.	
E4 - C2, C3, C4	Existe un elemento QoSCharacteristic.	Se agrega un elemento QoSDimension con el estereotipo «qoSDimension» dentro del QoSCharacteristic. El tipo de dato del elemento es el que se seleccionó. Los atributos statisticalQualifier, direction y unit tienen los valores seleccionados.	Esperada.
E4 - C1		No agrega el elemento al lienzo	Esperada.
E5	Existen dos elementos QoSCharacteristic.	Se agrega la asociación entre los dos elementos QoSCharacteristic.	Esperada.
E6 - C6	Existe un elemento QoSCharacteristic.	Se modifica el valor de por lo menos un atributo del QoSDimension dentro de la QoSCharacteristic.	Esperada.
E7	Existe un elemento QoSCategory.	Se modifica alguna propiedad de la QoSCategory.	Esperada.
E8	Existe un elemento QoSCharacteristic.	Se modifica alguna propiedad de la QoSCharacteristic.	Esperada.
E9	Existe un elemento Generalization entre dos QoSCharacteristic.	Se elimina del modelo la generalización entre los dos elementos QoSCharacteristic.	Esperada.
E10 - C7	Existe un elemento QoSCategory.	Se elimina del modelo el elemento QoSCategory. También se eliminan todas las QoSCharacteristic junto a las QoSDimension incluidas en esa categoría así como las generalizaciones definidas.	Esperada.
E10 - C8	Existe un elemento	Se elimina del modelo el	Esperada.

	QoSCategory.	elemento QoSCategory. También se eliminan todas las QoSCharacteristic junto a las QoSDimension incluidas en esa categoría así como las generalizaciones definidas. Los elementos definidos para una QoSCharacteristic eliminada, pasa a tener tipo Undefined.	
E11 - C7	Existe un elemento QoSCharacteristic.	Se elimina del modelo el elemento QoSCharacteristic junto a las QoSDimension incluidas en él así como las generalizaciones definidas que incluyen dicha QoSCharacteristic.	Esperada.
E11 - C8	Existe un elemento QoSCharacteristic.	Se elimina del modelo el elemento QoSCharacteristic junto a las QoSDimension incluidas en él así como las generalizaciones definidas que incluyen dicha QoSCharacteristic. Los elementos definidos para esa QoSCharacteristic eliminada, pasa a tener tipo Undefined.	Esperada.
E12 - C7	Existe un elemento QoSCharacteristic.	Se elimina del modelo el elemento QoSDimension.	Esperada.
E12 - C8	Existe un elemento QoSCharacteristic.	Se elimina del modelo el elemento QoSDimension. Los Slots definidos para esa QoSDimension eliminada, pasan a tener tipo Undefined.	Esperada.

1.1.6.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para editar diagrama de Características QoS.

1.1.7. Editar diagrama de Contrato de Servicios con QoS

Permite al usuario editar un diagrama de tipo *Service Contract with QoS* existente agregando desde la paleta de herramientas del diagrama nuevos contratos de servicio al lienzo y agregar nuevos roles, usos de contratos de servicios y asociaciones dentro de un contrato de servicios existente, así como también puede agregar y modificar características y valores de calidad de servicios, y asociaciones entre valores

de calidad y contratos de servicios. El usuario puede también modificar los tipos de los elementos existentes o crear nuevos elementos para asignar como tipos de otros elementos. En todo momento el usuario puede eliminar elementos o asociaciones existentes en el diagrama.

1.1.7.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - Agregar un elemento QoSValue	Flujo Principal	2.1
E2 - Arrastrar un elemento QoSCharacteristic desde el árbol del modelo hasta un elemento QoSValue sin una QoSCharacteristic asociada.	Flujo Principal	2.2
E3 - Agregar un elemento Slot dentro de un elemento QoSValue, especificando un valor y la propiedad que implementa el Slot.	Flujo Principal	2.3
E4 - Agregar una dependencia de tipo QoSContract entre un contrato de servicio y un elemento QoSValue.	Flujo Principal	2.4
E5 - Modificar un elemento QoSValue existente en el diagrama.	Flujo Principal	2.5
E6 - Borrar una dependencia de tipo QoSContract.	Flujo Principal	2.6
E7 - Borrar un elemento QoSValue.	Flujo Principal	2.7

1.1.7.2. Condiciones

Condición	Detalles
C1	Especificar qué QoSCharacteristic implementa.
C2	No especificar qué QoSCharacteristic implementa.
C3	Seleccionar una QoSDimension.
C4	No seleccionar ninguna QoSDimension.
C5	Editar nombre.
C6	Editar valor de los slots.
C7	El elemento no tiene dependencias con otro elemento.
C8	El elemento tiene dependencias con otro elemento.

1.1.7.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1	Existe un elemento QoSCharacteristic	Se agrega el elemento QoSValue en el lienzo, al cual se puede modificar el nombre. Se muestra correctamente el nombre del QoSCharacteristic que implementa.	Esperada.
E1 - C2		Se agrega el elemento QoSValue en el lienzo, al cual se puede modificar el nombre.	Esperada.
E2 - C3	Existe un elemento	El elemento QoSValue	Esperada.

	QoSValue que no implemente una QosCharacteristic. Existe un elemento QoSCharacteristic con al menos una QoSDimension.	implementa la QoSCharacteristic. En el árbol del modelo, se puede ver que el QoSValue tiene slots relacionados a los QoSDimension especificadas.	
E2 - C4	Existe un elemento QoSValue que no implemente una QosCharacteristic. Existe un elemento QoSCharacteristic.	El elemento QoSValue implementa la QoSCharacteristic. En el árbol del modelo, se puede ver que el QoSValue no tiene ningún slot.	Esperada.
E3	Existe un elemento QoSValue. Existe un elemento QoSCharacteristic con al menos un QoSDimension.	Se agrega el slot en el QoSValue, asociado a la QoSDimension seleccionada. Se permite agregar un valor en el slot.	Esperada.
E4	Existe un elemento ServiceContract y un elemento QoSValue.	Se agrega la dependencia del tipo QoSContract entre el elemento Service contract y el QoSValue.	Esperada.
E5 - C5	Existe un elemento QoSValue.	Se modifica correctamente el nombre.	Esperada.
E5 - C6	Existe un elemento QoSValue con al menos un slot.	Se modifican correctamente los valores de los slots.	Esperada.
E6	Existe una dependencia QoSContract asociada a un QoSValue y un Service Contract	Se borra correctamente la dependencia QoSContract.	Esperada.
E7 - C7	Existe un elemento QoSValue.	Se borra correctamente el elemento QoSValue.	Esperada.
E7 - C8	Existe una dependencia QoSContract asociada a un QoSValue y un Service Contract.	Se borra correctamente el elemento QoSValue y la dependencia QoSContract.	Esperada.

1.1.7.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para editar diagrama de Contrato de Servicios con QoS.

1.1.8. Editar Diagrama de Participantes como Clases con QoS

Permite al usuario editar un diagrama de tipo *Participant Class with QoS* existente agregando nuevos elementos participante al lienzo y agregando nuevos puertos service y request a un elemento

participante existente en el lienzo. Además, el usuario podrá asignar nuevos tipos a los puertos contenidos en el lienzo, crear nuevos tipos para asignar a los puertos o eliminar asignaciones de tipos existentes. El usuario también puede agregar y modificar características y valores de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios. En todo momento el usuario podrá eliminar elementos contenidos en el diagrama.

1.1.8.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - Agregar un elemento QoSValue.	Flujo Principal	2.1
E2 - Arrastrar un elemento QoSCharacteristic a un elemento QoSValue.	Flujo Principal	2.2
E3 - Agregar un elemento Slot a un elemento QoSValue.	Flujo Principal	2.3
E4 - Agregar una dependencia del tipo QoSRequired entre un Request y un QoSValue.	Flujo Principal	2.4
E5 - Agregar una dependencia del tipo QoSOffered entre un Service y un QoSValue.	Flujo Principal	2.5
E6 - Modifica un elemento QoSValue.	Flujo Principal	2.6
E7 - Borra una dependencia QoSRequired.	Flujo Principal	2.7
E8 - Borra una dependencia QoSOffered.	Flujo Principal	2.8
E9 - Borra un elemento QoSValue.	Flujo Principal	2.9

1.1.8.2. Condiciones

Condición	Detalles
C1	Especificar qué QoSCharacteristic implementa.
C2	No especificar qué QoSCharacteristic implementa.
C3	Seleccionar una QoSDimension.
C4	No seleccionar ninguna QoSDimension.
C5	Editar nombre.
C6	Editar valor de los slots.
C7	El elemento no tiene dependencias con otro elemento.
C8	El elemento tiene dependencias QoSRequired con otro elemento.
C9	El elemento tiene dependencias QoSOffered con otro elemento.

1.1.8.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1	Existe un elemento QoSCharacteristic.	Se agrega el elemento QoSValue en el lienzo, al cual se puede modificar el nombre. Se muestra correctamente el nombre del QoSCharacteristic que implementa.	Esperada.
E1 - C2		Se agrega el elemento QoSValue en el lienzo, al	Esperada.

		cual se puede modificar el nombre.	
E2 - C3	Existe un elemento QoSValue que no implemente una QosCharacteristic. Existe un elemento QoSCharacteristic con al menos una QoSDimension.	El elemento QoSValue implementa la QoSCharacteristic. En el árbol del modelo, se puede ver que el QoSValue tiene slots relacionados a los QoSDimension especificadas.	Esperada.
E2 - C4	Existe un elemento QoSValue que no implemente una QosCharacteristic. Existe un elemento QoSCharacteristic.	El elemento QoSValue implementa la QoSCharacteristic. En el árbol del modelo, se puede ver que el QoSValue no tiene ningún slot.	Esperada.
E3	Existe un elemento QoSValue. Existe un elemento QoSCharacteristic con al menos un QoSDimension.	Se agrega el slot en el QoSValue, asociado a la QoSDimension seleccionada. Se permite agregar un valor en el slot.	Esperada.
E4	Existe un elemento Request y un elemento QoSValue.	Se agrega una dependencia del tipo QoSRequired entre el elemento Request y el QoSValue.	Esperada.
E5	Existe un elemento Service y un elemento QoSValue.	Se agrega una dependencia del tipo QoSOffered entre el elemento Service y el QoSValue.	Esperada.
E6 - C5	Existe un elemento QoSValue.	Se modifica correctamente el nombre.	Esperada.
E6 - C6	Existe un elemento QoSValue con al menos un slot.	Se modifican correctamente los valores de los slots.	Esperada.
E7	Exite una depenendcia QoSRequired asociada entre un elemento Request y un elemento QoSValue.	Se borra correctamente la dependencia QoSRequired.	Esperada.
E8	Exite una depenendcia QoSOffered asociada entre un elemento Service y un elemento QoSValue.	Se borra correctamente la dependencia QoSOffered.	Esperada.
E9 - C7	Existe un elemento QoSValue.	Se borra correctamente el elemento QoSValue.	Esperada.
E9 - C8	Existe una dependencia	Se borra correctamente el	Esperada.

	QoSRequired asociada a un QoSValue y un elemento Request.	elemento QoSValue y la dependencia QoSRequired.	
E9 - C9	Existe una dependencia QoSOffered asociada a un QoSValue y un elemento Service.	Se borra correctamente el elemento QoSValue y la dependencia QoSOffered.	Esperada.

1.1.8.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para editar diagrama de Participantes como Clases con QoS.

Se observa un comportamiento no favorable en Papyrus cuando se define un elemento QoSValue arrastrando el elemento QoSCharacteristic y seleccionando una QoSDiemension: los Slots definidos en la QoSDimension seleccionada no se ven dentro del elemento QoSValue en el diagrama pero sí se ven definidos dentro del elemento en el árbol del modelo. Para que aparezcan dentro del elemento QoSValue en el diagrama alcanza con arrastrar los Slots desde el árbol del modelo.

1.1.9. Editar Diagrama de Participantes como Componentes con QoS

Permite al usuario editar un diagrama de tipo *Participant Component with QoS*, agregando nuevos componentes participante al lienzo y agregando nuevos elementos dentro de participantes existentes tales como proveedores de servicios, puertos, dependencias, asociaciones, canales de servicio, capacidades, usos de arquitecturas de servicios y los correspondientes rolebindings con los proveedores de servicios asociados. Además el usuario puede modificar los puertos, proveedores de servicios y capacidades usos de arquitecturas de servicios existentes asignándoles nuevos tipos o borrando la asignación de tipos existente. El usuario también puede agregar y modificar características y valores de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios. En todo momento el usuario podrá eliminar elementos contenidos en el diagrama.

1.1.9.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - Agregar un elemento QoSValue.	Flujo Principal	2.1
E2 - Arrastrar un elemento QoSCharacteristic a un elemento QoSValue.	Flujo Principal	2.2
E3 - Agregar un elemento Slot a un elemento QoSValue.	Flujo Principal	2.3
E4 - Agregar una dependencia del tipo QoSRequired entre un Request y un QoSValue.	Flujo Principal	2.4
E5 - Agregar una dependencia del tipo QoSOffered entre un Service y un QoSValue.	Flujo Principal	2.5
E6 - Modifica un elemento QoSValue.	Flujo Principal	2.6
E7 - Borra una dependencia QoSRequired.	Flujo Principal	2.7
E8 - Borra una dependencia QoSOffered.	Flujo Principal	2.8
E9 - Borra un elemento QoSValue.	Flujo Principal	2.9

1.1.9.2. Condiciones

Condición	Detalles
C1	Especificar qué QoSCharacteristic implementa.
C2	No especificar qué QoSCharacteristic implementa.
C3	Seleccionar una QoSDimension.
C4	No seleccionar ninguna QoSDimension.
C5	Editar nombre.
C6	Editar valor de los slots.
C7	El elemento no tiene dependencias con otro elemento.
C8	El elemento tiene dependencias QoSRequired con otro elemento.
C9	El elemento tiene dependencias QoSOffered con otro elemento.

1.1.9.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1	Existe un elemento QoSCharacteristic.	Se agrega el elemento QoSValue en el lienzo, al cual se puede modificar el nombre. Se muestra correctamente el nombre del QoSCharacteristic que implementa.	Esperada.
E1 - C2		Se agrega el elemento QoSValue en el lienzo, al cual se puede modificar el nombre.	Esperada.
E2 - C3	Existe un elemento QoSValue que no implemente una QosCharacteristic. Existe un elemento QoSCharacteristic con al menos una QoSDimension.	El elemento QoSValue implementa la QoSCharacteristic. En el árbol del modelo, se puede ver que el QoSValue tiene slots relacionados a los QoSDimension especificadas.	Esperada.
E2 - C4	Existe un elemento QoSValue que no implemente una QosCharacteristic. Existe un elemento QoSCharacteristic.	El elemento QoSValue implementa la QoSCharacteristic. En el árbol del modelo, se puede ver que el QoSValue no tiene ningún slot.	Esperada.
E3	Existe un elemento QoSValue. Existe un elemento QoSCharacteristic con al menos un QoSDimension.	Se agrega el slot en el QoSValue, asociado a la QoSDimension seleccionada. Se permite agregar un valor en el slot.	Esperada.

E4	Existe un elemento Request y un elemento QoSValue.	Se agrega una dependencia del tipo QoSRequired entre el elemento Request y el QoSValue.	Esperada.
E5	Existe un elemento Service y un elemento QoSValue.	Se agrega una dependencia del tipo QoSOffered entre el elemento Service y el QoSValue.	Esperada.
E6 - C5	Existe un elemento QoSValue.	Se modifica correctamente el nombre.	Esperada.
E6 - C6	Existe un elemento QoSValue con al menos un slot.	Se modifican correctamente los valores de los slots.	Esperada.
E7	Exite una depenendcia QoSRequired asociada entre un elemento Request y un elemento QoSValue.	Se borra correctamente la dependencia QoSRequired.	Esperada.
E8	Exite una depenendcia QoSOffered asociada entre un elemento Service y un elemento QoSValue.	Se borra correctamente la dependencia QoSOffered.	Esperada.
E9 - C7	Existe un elemento QoSValue.	Se borra correctamente el elemento QoSValue.	Esperada.
E9 - C8	Existe una dependencia QoSRequired asociada a un QoSValue y un elemento Request.	Se borra correctamente el elemento QoSValue y la dependencia QoSRequired.	Esperada.
E9 - C9	Existe una dependencia QoSOffered asociada a un QoSValue y un elemento Service.	Se borra correctamente el elemento QoSValue y la dependencia QoSOffered.	Esperada.

1.1.9.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para editar diagrama de Participantes como Componentes con QoS.

Se observa un comportamiento no favorable en Papyrus cuando se define un elemento QoSValue arrastrando el elemento QoSCharacteristic y seleccionando una QoSDiemension: los Slots definidos en la QoSDimension seleccionada no se ven dentro del elemento QoSValue en el diagrama pero sí se ven definidos dentro del elemento en el árbol del modelo. Para que aparezcan dentro del elemento QoSValue en el diagrama alcanza con arrastrar los Slots desde el árbol del modelo.

1.1.10. Eliminar diagrama de Características QoS

Permite al usuario eliminar un diagrama preexistente de tipo *QoS Characteristic*, esto es eliminar cada uno de los elementos contenidos en el diagrama y las referencias existentes a ellos en otros diagramas.

1.1.10.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - El usuario elimina todos los elementos de un diagrama de Características QoS.	Flujo Principal	2
E2 - El usuario elimina un diagrama de Características QoS.	Flujo Principal	3

1.1.10.2. Condiciones

Condición	Detalles
C1	El diagrama tiene al menos un elemento.
C2	El diagrama no contiene elementos.
C3	Hay referencias a los elementos del diagrama en otros diagramas del modelo.
C4	No hay referencias a los elementos del diagrama en otros diagramas del modelo.

1.1.10.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1, C3	Existe un diagrama de Características QoS con al menos un elemento.	Se eliminan correctamente todos los elementos del diagrama. Se eliminan todas las referencias a los elementos del diagrama que hayan en otros diagramas del modelo.	Esperada.
E1 - C1, C4	Existe un diagrama de Características QoS con al menos un elemento.	Se eliminan correctamente todos los elementos del diagrama.	Esperada.
E2 - C1	Existe un diagrama de Características QoS con al menos un elemento.	Se elimina correctamente el diagrama, pero los elementos que estaban contenidos en el diagrama no se eliminan. En el árbol de elementos del modelo se pueden ver estos elementos.	Esperada.
E2 - C2	Existe un diagrama de Características QoS sin elementos.	Se elimina correctamente el diagrama.	Esperada.

1.1.10.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para eliminar diagrama de Características QoS.

1.1.11. Eliminar diagrama de Contrato de Servicios con QoS

Permite al usuario eliminar un diagrama preexistente de tipo *Service Contract with QoS*, esto es eliminar cada uno de los elementos contenidos en el diagrama y las referencias existentes a ellos en otros diagramas.

1.1.11.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - El usuario elimina todos los elementos de un diagrama de Contratos de Servicio con QoS.	Flujo Principal	2
E2 - El usuario elimina un diagrama de Contratos de Servicio con QoS.	Flujo Principal	3

1.1.11.2. Condiciones

Condición	Detalles
C1	El diagrama tiene al menos un elemento.
C2	El diagrama no contiene elementos.
C3	Hay referencias a los elementos del diagrama en otros diagramas del modelo.
C4	No hay referencias a los elementos del diagrama en otros diagramas del modelo.

1.1.11.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1, C3	Existe un diagrama de Contratos de Servicio con al menos un elemento.	Se eliminan correctamente todos los elementos del diagrama. Se eliminan todas las referencias a los elementos del diagrama que hayan en otros diagramas del modelo.	Esperada.
E1 - C1, C4	Existe un diagrama de Contratos de Servicio con al menos un elemento.	Se eliminan correctamente todos los elementos del diagrama.	Esperada.
E2 - C1	Existe un diagrama de Contratos de Servicio con al menos un elemento.	Se elimina correctamente el diagrama, pero los elementos que estaban contenidos en el diagrama no se eliminan. En el árbol de elementos del modelo se pueden ver estos elementos.	Esperada.
E2 - C2	Existe un diagrama de Contratos de Servicio sin elementos.	Se elimina correctamente el diagrama.	Esperada.

1.1.11.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para eliminar diagrama de Contrato de Servicios con QoS.

1.1.12. Eliminar diagrama de Participantes como Clases con QoS

Permite al usuario eliminar un diagrama preexistente de tipo *Participant Class with QoS*, esto es eliminar cada uno de los elementos contenidos en el diagrama y las referencias existentes a ellos en otros diagramas.

1.1.12.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - El usuario elimina todos los elementos de un diagrama de Participantes como Clases con QoS.	Flujo Principal	2
E2 - El usuario elimina un diagrama de Participantes como Clases con QoS.	Flujo Principal	3

1.1.12.2. Condiciones

Condición	Detalles
C1	El diagrama tiene al menos un elemento.
C2	El diagrama no contiene elementos.
C3	Hay referencias a los elementos del diagrama en otros diagramas del modelo.
C4	No hay referencias a los elementos del diagrama en otros diagramas del modelo.

1.1.12.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1, C3	Existe un diagrama de Participantes como Clases con al menos un elemento.	Se eliminan correctamente todos los elementos del diagrama. Se eliminan todas las referencias a los elementos del diagrama que hayan en otros diagramas del modelo.	Esperada.
E1 - C1, C4	Existe un diagrama de Participantes como Clases con al menos un elemento.	Se eliminan correctamente todos los elementos del diagrama.	Esperada.
E2 - C1	Existe un diagrama de Participantes como Clases con al menos un elemento.	Se elimina correctamente el diagrama, pero los elementos que estaban contenidos en el diagrama no se eliminan. En el árbol de elementos del modelo se pueden ver estos elementos.	Esperada.
E2 - C2	Existe un diagrama de Participantes como Clases sin elementos.	Se elimina correctamente el diagrama.	Esperada.

1.1.12.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para eliminar diagrama de Participantes como Clases con QoS.

1.1.13. Eliminar diagrama de Participantes como Componentes con QoS

Permite al usuario eliminar un diagrama preexistente de tipo *Participant Component with QoS*, esto es eliminar cada uno de los elementos contenidos en el diagrama y las referencias existentes a ellos en otros diagramas.

1.1.13.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - El usuario elimina todos los elementos de un diagrama de Participantes como Componentes con QoS.	Flujo Principal	2
E2 - El usuario elimina un diagrama de Participantes como Componentes con QoS.	Flujo Principal	3

1.1.13.2. Condiciones

Condición	Detalles
C1	El diagrama tiene al menos un elemento.
C2	El diagrama no contiene elementos.
C3	Hay referencias a los elementos del diagrama en otros diagramas del modelo.
C4	No hay referencias a los elementos del diagrama en otros diagramas del modelo.

1.1.13.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1, C3	Existe un diagrama de Participantes como Componentes con al menos un elemento.	Se eliminan correctamente todos los elementos del diagrama. Se eliminan todas las referencias a los elementos del diagrama que hayan en otros diagramas del modelo.	Esperada.
E1 - C1, C4	Existe un diagrama de Participantes como Componentes con al menos un elemento.	Se eliminan correctamente todos los elementos del diagrama.	Esperada.
E2 - C1	Existe un diagrama de Participantes como Componentes con al menos un elemento.	Se elimina correctamente el diagrama, pero los elementos que estaban contenidos en el diagrama no se eliminan. En el árbol de elementos del modelo se pueden ver estos elementos.	Esperada.
E2 - C2	Existe un diagrama de Participantes como	Se elimina correctamente el diagrama.	Esperada.

	Componentes sin elementos.		

1.1.13.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para eliminar diagrama de Participantes como Componentes con QoS.

1.2.Casos de uso de importar/exportar modelo

1.2.1. Importar modelo desde archivo con formato XMI

Permite al usuario importar un modelo SoaML o SoaML+QoS desde un archivo XML con formato XMI, de forma de que se generen en el árbol del modelo los elementos contenidos en el archivo importado con sus estereotipos asociados.

1.2.1.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - El usuario selecciona un archivo XMI donde está definido el modelo SoaML+QoS.	Flujo Principal	2

1.2.1.2. Condiciones

Condición	Detalles
C1	El archivo XMI tiene elementos correspondientes a un diagrama de Características QoS.
C2	El archivo XMI tiene elementos correspondientes a un diagrama de Contrato de Servicios con QoS.
C3	El archivo XMI tiene elementos correspondientes a un diagrama de Participantes como Clases con QoS.
C4	El archivo XMI tiene elementos correspondientes a un diagrama de Participantes como Componentes con QoS.
C5	El archivo XMI tiene elementos correspondientes a un diagrama de Características QoS y/o a un diagrama de Contrato de Servicios con QoS y/o a un diagrama de Participantes como Clases con QoS y/o a un diagrama de Participantes como Componentes con QoS.

1.2.1.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida

E1 - C1		Se genera un nuevo modelo Papyrus con un diagrama de Arquitectura de Servicios, el perfil SoaML+QoS aplicado y se generan en el árbol del modelo los elementos existentes en el XMI con sus respectivos estereotipos correspondientes al perfil SoaML+QoS.	Esperada.
E1 - C2		Se genera un nuevo modelo Papyrus con un diagrama de Arquitectura de Servicios, el perfil SoaML+QoS aplicado y se generan en el árbol del modelo los elementos existentes en el XMI con sus respectivos estereotipos correspondientes al perfil SoaML+QoS.	Esperada.
E1 - C3		Se genera un nuevo modelo Papyrus con un diagrama de Arquitectura de Servicios, el perfil SoaML+QoS aplicado y se generan en el árbol del modelo los elementos existentes en el XMI con sus respectivos estereotipos correspondientes al perfil SoaML+QoS.	Esperada.
E1 - C4		Se genera un nuevo modelo Papyrus con un diagrama de Arquitectura de Servicios, el perfil SoaML+QoS aplicado y se generan en el árbol del modelo los elementos existentes en el XMI con sus respectivos estereotipos correspondientes al perfil SoaML+QoS.	Esperada.
E1 - C5		Se genera un nuevo modelo Papyrus con un diagrama de Arquitectura de Servicios, el perfil SoaML+QoS aplicado y se generan en el árbol del modelo los elementos existentes en el XMI con sus respectivos estereotipos correspondientes al perfil SoaML+QoS.	Esperada.

1.2.1.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para importar modelo desde archivo con formato XMI.

1.2.2. Exportar modelo desde archivo con formato XMI

Permite al usuario exportar un modelo SoaML o SoaML+QoS con sus elementos y estereotipos asociados desde el editor a un archivo XML con formato XMI.

1.2.2.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - El usuario selecciona un archivo UML donde está definido el modelo SoaML+QoS.	Flujo Principal	2

1.2.2.2. Condiciones

Condición	Detalles
C1	El archivo UML tiene elementos correspondientes a un diagrama de Características QoS.
C2	El archivo UML tiene elementos correspondientes a un diagrama de Contrato de Servicios con QoS.
C3	El archivo UML tiene elementos correspondientes a un diagrama de Participantes como Clases con QoS.
C4	El archivo UML tiene elementos correspondientes a un diagrama de Participantes como Componentes con QoS.
C5	El archivo UML tiene elementos correspondientes a un diagrama de Características QoS y/o a un diagrama de Contrato de Servicios con QoS y/o a un diagrama de Participantes como Clases con QoS y/o a un diagrama de Participantes como Componentes con QoS.

1.2.2.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1	Existe un modelo SoaML+QoS y un diagrama de Características QoS definido en él.	Se genera un archivo XMI con el formato adecuado en donde se definen todos los elementos del modelo.	Esperada.
E1 - C2	Existe un modelo SoaML+QoS y un diagrama de Constrato de Servicios con QoS definido en él.	Se genera un archivo XMI con el formato adecuado en donde se definen todos los elementos del modelo.	Esperada.
E1 - C3	Existe un modelo SoaML+QoS y un diagrama de Participantes como	Se genera un archivo XMI con el formato adecuado en donde se definen todos los elementos del modelo.	Esperada.

	Clases con QoS definido en él.		
E1 - C4	Existe un modelo SoaML+QoS y un diagrama de Participantes como Componentes con QoS definido en él.	Se genera un archivo XMI con el formato adecuado en donde se definen todos los elementos del modelo.	Esperada.
E1 - C5	Existe un modelo SoaML+QoS y un diagrama de Características QoS y/o un diagrama de Contrato de Servicios con QoS y/o un diagrama de Participantes como Clases con QoS y/o un diagrama de Participantes como Componentes con QoS definidos en él.	Se genera un archivo XMI con el formato adecuado en donde se definen todos los elementos del modelo.	Esperada.

1.2.2.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para exportar modelo desde archivo con formato XMI.

1.3.Casos de uso de generación de código

1.3.1. Generar proyectos Java a partir de modelos SoaML+QoS y SoaML

Permite al usuario (arquitecto del sistema) generar proyectos java a partir de modelos SoaML+QoS. El modelo puede estar en formato XMI o en formato UML, y se permite generar distintos tipos de proyectos que exponen los servicios web especificados en el modelo y los clientes de estos servicios. También permite generar contratos WS-Agreement a partir de las características de calidad asociadas a los contratos especificados en el modelo, y permite generar archivos WSDL con la descripción de los servicios, y con políticas WS-Policy a partir de las características de calidad asociadas a los puertos de servicio en el modelo.

1.3.1.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - El usuario completa el wizard de generación de código Java.	Flujo Principal	-

1.3.1.2. Condiciones

Condición	Detalles
C1	El usuario genera código Java a partir de un UML.

C2	El usuario genera código Java a partir de un archivo XMI 2.1.
C3	El usuario selecciona un solo participante.
C4	El usuario selecciona más de un participante.
C5	El usuario genera servidores del tipo “Dynamic Web Project (war)”.
C6	El usuario genera servidores del tipo “Java EJB Project (jar)”.
C7	El usuario selecciona que los servicios web generados sean implementados con “JAX-WS ri”.
C8	El usuario selecciona que los servicios web generados sean implementados con “JAX-WS + spring”.
C9	El usuario selecciona que el servidor donde correrán los proyectos generados sea JBoss.
C10	El usuario selecciona que el servidor donde correrán los proyectos generados sea Tomcat.
C11	El usuario selecciona que desea generar clientes para los participantes seleccionados.
C12	El usuario selecciona la IP y puerto por defecto.
C13	El usuario especifica una IP y puerto distintos a los que están por defecto.
C14	El usuario selecciona que desea generar documentos WS-Agreement.
C15	El usuario selecciona que desea generar políticas WS-Policy. Especifica que una de las características listadas es del tipo Username Token.
C16	El usuario selecciona que desea generar políticas WS-Policy. Especifica que al menos una de las características listadas es del tipo Generic.
C17	El usuario selecciona que desea generar políticas WS-Policy. Especifica que al menos una de las características listadas es del tipo Response Time.

1.3.1.3. Casos de Prueba

Escenario - Condición	**Precondición**	**Salida Esperada**	**Salida Obtenida**
E1 - C1, C3, C5, C7, C10, C11, C12.	Existe un proyecto SoaML o SoaML+QoS con una estructura de elementos válida.	Se genera un proyecto web que utiliza JAX-WS ri y un cliente para el participante seleccionado. En los dos proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. El cliente generado invoca correctamente los servicios del servidor generado cuando se ejecuta sobre Tomcat en el IP/puerto seleccionado.	Esperada.
E1 - C2, C3, C5, C7, C10, C11, C12.	Existe un archivo XMI 2.1 correspondiente a un modelo SoaML o SoaML+QoS con una estructura de elementos válida.	Se genera un proyecto web que utiliza JAX-WS ri y un cliente para el participante seleccionado. En los dos proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. El cliente generado invoca correctamente los servicios del servidor generado	Esperada.

		cuando se ejecuta sobre Tomcat en el IP/puerto seleccionado.	
E1 - C1, C3, C5, C7, C9, C11, C12, C14, C15, C16, C17	Existe un proyecto SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante seleccionado. El participante tiene puertos del tipo Request con QoSValues asociados.	Se genera un proyecto web que utiliza JAX-WS ri y un cliente para el participante seleccionado. En los dos proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. También contienen una policy del tipo UsernameToken. Se genera un archivo Ws-Agreement por cada ServiceContract asociado a un QoSValue. El documento contiene la información de las características de calidad asociadas al contrato. El cliente generado invoca correctamente los servicios del servidor generado cuando se ejecuta sobre JBoss en el IP/puerto seleccionado. Al ejecutarse el cliente, se imprime el tiempo de respuesta del servicio en el archivo de log del cliente.	Esperada. Los proyectos se generan correctamente, incluyendo las policies en el wsdl y los documentos ws-agreement. Al ejecutar el cliente salta un error de acceso restringido en el archivo de log. Luego de modificar la configuración de en dónde se debe generar el archivo de log, el servicio se invoca correctamente.
E1 - C2, C3, C5, C7, C9, C11, C12, C14, C15, C16, C17	Existe un archivo XMI 2.1 correspondiente a un modelo SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante seleccionado. El participante tiene puertos del tipo Request con QoSValues asociados.	Se genera un proyecto web que utiliza JAX-WS ri y un cliente para el participante seleccionado. En los dos proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. También contienen una policy del tipo UsernameToken. Se genera un archivo Ws-Agreement por cada ServiceContract asociado a	Esperada. Los proyectos se generan correctamente, incluyendo las policies en el wsdl y los documentos ws-agreement. Al ejecutar el cliente salta un error de acceso restringido en el archivo de log. Luego de modificar la configuración de

		un QoSValue. El documento contiene la información de las características de calidad asociadas al contrato. El cliente generado invoca correctamente los servicios del servidor generado cuando se ejecuta sobre JBoss en el IP/puerto seleccionado. Al ejecutarse el cliente, se imprime el tiempo de respuesta del servicio en el archivo de log del cliente.	en dónde se debe generar el archivo de log, el servicio se invoca correctamente.
E1 - C2, C4, C5, C7, C9, C11, C12, C14, C15, C16, C17	Existe un archivo XMI 2.1 correspondiente a un modelo SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante seleccionado. El participante tiene puertos del tipo Request con QoSValues asociados.	Se genera un proyecto web que utiliza JAX-WS ri y un cliente para cada participante seleccionado. En todos los proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. También contienen una policy del tipo UsernameToken. Se genera un archivo Ws-Agreement por cada ServiceContract asociado a un QoSValue en el que se encuentre el participante asociado al proyecto. El documento contiene la información de las características de calidad asociadas al contrato. Los clientes generados invocan correctamente los servicios de los servidores generados cuando se ejecutan sobre JBoss en el IP/puerto seleccionado. Al ejecutarse el cliente, se imprime el tiempo de respuesta del servicio en el archivo de log del cliente.	No esperada. En los proyectos cliente y servidor de uno de los participantes se generaron documentos WS-Agreement asociados a contratos en los que no se encuentra el participante. Al ejecutar el cliente salta un error de acceso restringido en el archivo de log. Luego de modificar la configuración de en dónde se debe generar el archivo de log, el servicio se invoca correctamente.
E1 - C2, C4, C5, C8, C10, C11, C13,	Existe un archivo XMI 2.1 correspondiente a un modelo	Se genera un proyecto web que utiliza JAX-WS + spring y un cliente para cada participante seleccionado.	No esperada. En los proyectos

C14, C15, C16, C17.	SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante seleccionado. El participante tiene puertos del tipo Request con QoSValues asociados.	En todos los proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. También contienen una policy del tipo UsernameToken. Se genera un archivo Ws-Agreement por cada ServiceContract asociado a un QoSValue en el que se encuentre el participante asociado al proyecto. El documento contiene la información de las características de calidad asociadas al contrato. Los clientes generados invocan correctamente los servicios de los servidores generados cuando se ejecutan sobre Tomcat en el IP/puerto seleccionado. Al ejecutarse el cliente, se imprime el tiempo de respuesta del servicio en el archivo de log del cliente.	cliente y servidor de uno de los participantes se generaron documentos WS-Agreement asociados a contratos en los que no se encuentra el participante. Al ejecutar el cliente salta un error de acceso restringido en el archivo de log. Luego de modificar la configuración de en dónde se debe generar el archivo de log, el servicio se invoca correctamente.
E1 - C2, C3, C6, C11, C13, C14, C16.	Existe un archivo XMI 2.1 correspondiente a un modelo SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante seleccionado. El participante tiene puertos del tipo Request con QoSValues asociados.	Se genera un proyecto de tipo “Java EJB Project” y un cliente, que contienen un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. Se genera un archivo Ws-Agreement por cada ServiceContract asociado a un QoSValue en el que se encuentre el participante asociado al proyecto. El documento contiene la información de las características de calidad asociadas al contrato.	Esperada.

1.3.1.4. Evaluación de resultados

La generación de código Java y generación de WSDL con características de calidad funcionan correctamente. Se detectó un error en la generación de documentos WS-Agreement, en donde bajo ciertas condiciones se generan documentos asociados a contratos en los que el participante no posee ningún rol.

También se detectó una posible mejora en la configuración de log4j: actualmente los clientes se configuran para imprimir los logs en la ruta "C:\output.log", lo cual puede traer problemas si el cliente no tiene permisos de escritura sobre esa ruta. Se puede evitar este error si el archivo de log se genera dentro de la carpeta donde se encuentra instalado el cliente.

1.4.Casos de uso de modelado SoaML

Como parte de este proyecto consistió en migrar los plugins existentes de modelado para que funcionen correctamente en Eclipse Luna y Papyrus 1.0.0, realizamos un testing exhaustivo para verificar que el modelado de SoaML continúe funcionando satisfactoriamente.

Se ejecutaron los 655 casos de prueba diseñados por el equipo de SoaML y como se puede ver en la Tabla 1 se obtuvieron los resultados deseados, comparando con los resultados obtenidos en las iteraciones 1, 2, 3 y 4 del proyecto anterior. Cabe aclarar que los problemas detectados en la versión anterior sobre la falta de validación en las asociaciones entre elementos y la asignación de tipos se mantienen.

Caso de uso	**Resultado obtenido**
Crear diagrama de Arquitectura de Servicios	Se obtuvieron los resultados esperados.
Crear diagrama de Contrato de Servicios	Se obtuvieron los resultados esperados.
Crear diagrama de Capacidades	Se obtuvieron los resultados esperados.
Crear diagrama de Interfaces	Se obtuvieron los resultados esperados.
Crear diagrama de Participantes como Clases	Se obtuvieron los resultados esperados.
Crear diagrama de Participantes como Componentes	Se obtuvieron los resultados esperados.
Crear diagrama de Mensajes	Se obtuvieron los resultados esperados.
Editar diagrama de Arquitectura de Servicios	Se obtuvieron los resultados esperados.
Editar diagrama de Contrato de Servicios	Se obtuvieron los resultados esperados.
Editar diagrama de Capacidades	Se obtuvieron los resultados esperados.
Editar diagrama de Interfaces	Se obtuvieron los resultados esperados.
Editar diagrama de Participantes como Clases	Se obtuvieron los resultados esperados.
Editar diagrama de Participantes como Componentes	Se obtuvieron los resultados esperados.
Editar diagrama de Mensajes	Se obtuvieron los resultados esperados.
Eliminar diagrama de Arquitectura de Servicios	Se obtuvieron los resultados esperados.
Eliminar diagrama de Contrato de Servicios	Se obtuvieron los resultados esperados.
Eliminar diagrama de Capacidades	Se obtuvieron los resultados esperados.

Eliminar diagrama de Interfaces	Se obtuvieron los resultados esperados.
Eliminar diagrama de Participantes como Clases	Se obtuvieron los resultados esperados.
Eliminar diagrama de Participantes como Componentes	Se obtuvieron los resultados esperados.
Eliminar diagrama de Mensajes	Se obtuvieron los resultados esperados.
Importar modelo desde archivo con formato XMI	Se obtuvieron los resultados esperados.
Exportar modelo a archivo con formato XMI	Se obtuvieron los resultados esperados.

Tabla 1Resultados de Casos de Prueba de SoaML

Por último, es necesario destacar que con la nueva versión de Papyrus hubo un cambio importante en el uso del editor. Anteriormente, cuando uno utilizaba la tecla Suprimir para eliminar un elemento del lienzo o del modelo, ésta estaba asociada con el evento Ocultar. Es por esta razón que en los casos de prueba se escribió que al eliminar un elemento del lienzo, éste seguía existiendo en el explorador del modelo, porque en lugar de eliminar el elemento definitivamente, se estaba ocultando. Ahora dicha tecla está asociada efectivamente al evento Eliminar, por lo que los casos de prueba en donde se menciona eliminar un elemento del lienzo, éste también es eliminado del modelo. En resumen, el comportamiento es el siguiente:

- Si se elimina un diagrama que contiene elementos, éstos se mantienen en la vista del modelo aunque no estén referenciados en otro lado.
- Si se elimina un elemento de un diagrama, éste es eliminado también de la vista del modelo y de todos los diagramas en los que estuviera o fuera referenciado.

2. Iteración 2

En la primera iteración se detectaron errores en la importación de un modelo de características QoS y en la generación de código, por lo tanto se realizaron las modificaciones correspondientes para resolver dichos errores y se ejecutaron nuevamente pruebas sobre estos casos de uso en la segunda iteración. Además, en la segunda iteración se agregaron las pruebas para el caso de uso de importación de modelo SoaML.

2.1. Casos de uso de modelado

2.1.1. Importar modelo de Características QoS

Permite al usuario importar un modelo de características QoS a un modelo SoaML+QoS, realizando una copia en profundidad de todos los elementos del modelo origen al modelo destino.

2.1.1.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - El usuario selecciona el menú “Import QoS Elements from QoS Model” y selecciona un archivo UML.	Flujo Principal	1, 2

2.1.1.2. Condiciones

Condición	Detalles
C1	El archivo UML seleccionado corresponde a un modelo QoS con elementos.
C2	El archivo UML seleccionado corresponde a un modelo QoS vacío.
C3	El archivo UML seleccionado no corresponde a un modelo QoS.

2.1.1.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1	Existe un modelo QoS con elementos.	Se copian todos los elementos del modelo seleccionado al árbol de elementos del modelo actual.	Esperada.
E1 - C2	Existe un modelo QoS sin elementos.	El árbol de elementos del modelo actual no tiene cambios.	Esperada.
E1 - C3	Existe un modelo que no sea QoS.	Se despliega un mensaje de error indicando que el modelo seleccionado no es un modelo QoS.	Esperada.

2.1.1.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para importar un modelo de Características QoS. Se solucionó el error de la iteración 1 por lo que cuando se selecciona un modelo que no es QoS aparece un mensaje de error y se cancela el copiado de elementos.

2.1.2. Importar modelo SoaML

Permite al usuario importar un modelo SoaML a un modelo SoaML+QoS, realizando una copia en profundidad de todos los elementos del modelo origen al modelo destino.

2.1.2.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - El usuario selecciona el menú “Import SoaML Elements from SoaML Model” y selecciona un archivo UML.	Flujo Principal	1, 2

2.1.2.2. Condiciones

Condición	Detalles
C1	El archivo UML seleccionado corresponde a un modelo SoaML con elementos.
C2	El archivo UML seleccionado corresponde a un modelo SoaML vacío.
C3	El archivo UML seleccionado no corresponde a un modelo SoaML.

2.1.2.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1	Existe un modelo SoaML con elementos.	Se copian todos los elementos del modelo seleccionado al árbol de elementos del modelo actual.	Esperada.
E1 - C2	Existe un modelo SoaML sin elementos.	El árbol de elementos del modelo actual no tiene cambios.	Esperada.
E1 - C3	Existe un modelo que no sea SoaML.	Se despliega un mensaje de error indicando que el modelo seleccionado no es un modelo SoaML.	Esperada.

2.1.2.4. Evaluación de los resultados

Se obtuvieron resultados favorables en las pruebas para importar un modelo SoaML.

2.2. Casos de uso de generación de código

2.2.1. Generar proyectos Java a partir de modelo SoaML+QoS y SoaML

Permite al usuario (arquitecto del sistema) generar proyectos java a partir de modelos SoaML+QoS. El modelo puede estar en formato XMI o en formato UML, y se permite generar distintos tipos de proyectos que exponen los servicios web especificados en el modelo y los clientes de estos servicios. También permite generar contratos WS-Agreement a partir de las características de calidad asociadas a los contratos especificados en el modelo, y permite generar archivos WSDL con la descripción de los servicios, y con políticas WS-Policy a partir de las características de calidad asociadas a los puertos de servicio en el modelo.

2.2.1.1. Escenarios

Nombre de escenario	Flujo Inicial	Número de Flujo
E1 - El usuario completa el wizard de generación de código Java.	Flujo Principal	-

2.2.1.2. Condiciones

Condición	Detalles
C1	El usuario genera código Java a partir de un UML.
C2	El usuario genera código Java a partir de un archivo XMI 2.1.
C3	El usuario selecciona un solo participante.
C4	El usuario selecciona más de un participante.
C5	El usuario genera servidores del tipo “Dynamic Web Project (war)”.
C6	El usuario genera servidores del tipo “Java EJB Project (jar)”.
C7	El usuario selecciona que los servicios web generados sean implementados con “JAX-WS ri”.
C8	El usuario selecciona que los servicios web generados sean implementados con “JAX-WS + spring”.
C9	El usuario selecciona que el servidor donde correrán los proyectos generados sea JBoss.
C10	El usuario selecciona que el servidor donde correrán los proyectos generados sea Tomcat.
C11	El usuario selecciona que desea generar clientes para los participantes seleccionados.
C12	El usuario selecciona la IP y puerto por defecto.
C13	El usuario especifica una IP y puerto distintos a los que están por defecto.
C14	El usuario selecciona que desea generar documentos WS-Agreement.
C15	El usuario selecciona que desea generar políticas WS-Policy. Especifica que una de las características listadas es del tipo Username Token.
C16	El usuario selecciona que desea generar políticas WS-Policy. Especifica que al menos una de las características listadas es del tipo Generic.
C17	El usuario selecciona que desea generar políticas WS-Policy. Especifica que al menos una de las características listadas es del tipo Response Time.
C18	Levantar el servidor en una I.P. distinta a localhost y ejecutar el cliente desde otra máquina en la red.

2.2.1.3. Casos de Prueba

Escenario - Condición	Precondición	Salida Esperada	Salida Obtenida
E1 - C1, C3, C5, C7, C10, C11, C12.	Existe un proyecto SoaML o SoaML+QoS con una estructura de elementos válida.	Se genera un proyecto web que utiliza JAX-WS ri y un cliente para el participante seleccionado. En los dos proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. El cliente generado invoca correctamente los servicios del servidor generado cuando se ejecuta sobre	Esperada.

		Tomcat en el IP/puerto seleccionado.	
E1 - C2, C3, C5, C7, C10, C11, C12.	Existe un archivo XMI 2.1 correspondiente a un modelo SoaML o SoaML+QoS con una estructura de elementos válida.	Se genera un proyecto web que utiliza JAX-WS ri y un cliente para el participante seleccionado. En los dos proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. El cliente generado invoca correctamente los servicios del servidor generado cuando se ejecuta sobre Tomcat en el IP/puerto seleccionado.	Esperada. Los servicios se ejecutan correctamente aunque en Tomcat se imprime la siguiente advertencia: "WARNING: onComplete() failed for listener of type [org.apache.catalina.core.AsyncListenerWrapper] java.lang.IllegalStateException: It is illegal to call getRequest() after complete() or any of the dispatch() methods has been called"
E1 - C1, C3, C5, C7, C9, C11, C12, C14, C15, C16, C17	Existe un proyecto SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante seleccionado. El participante tiene puertos del tipo Request con QoSValues asociados.	Se genera un proyecto web que utiliza JAX-WS ri y un cliente para el participante seleccionado. En los dos proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. También contienen una policy del tipo UsernameToken. Se genera un archivo Ws-Agreement por cada ServiceContract asociado a un QoSValue. El documento contiene la información de las características de calidad asociadas al contrato. El cliente generado invoca correctamente los servicios del servidor generado cuando se ejecuta sobre JBoss en el IP/puerto seleccionado.	Esperada.

		Al ejecutarse el cliente, se imprime el tiempo de respuesta del servicio en el archivo de log del cliente.	
E1 - C2, C3, C5, C7, C9, C11, C12, C14, C15, C16, C17	Existe un archivo XMI 2.1 correspondiente a un modelo SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante seleccionado. El participante tiene puertos del tipo Request con QoSValues asociados.	Se genera un proyecto web que utiliza JAX-WS ri y un cliente para el participante seleccionado. En los dos proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. También contienen una policy del tipo UsernameToken. Se genera un archivo Ws-Agreement por cada ServiceContract asociado a un QoSValue. El documento contiene la información de las características de calidad asociadas al contrato. El cliente generado invoca correctamente los servicios del servidor generado cuando se ejecuta sobre JBoss en el IP/puerto seleccionado. Al ejecutarse el cliente, se imprime el tiempo de respuesta del servicio en el archivo de log del cliente.	Esperada.
E1 - C2, C4, C5, C7, C9, C11, C12, C14, C15, C16, C17	Existe un archivo XMI 2.1 correspondiente a un modelo SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante seleccionado. El	Se genera un proyecto web que utiliza JAX-WS ri y un cliente para cada participante seleccionado. En todos los proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. También contienen una policy del tipo UsernameToken. Se genera un archivo Ws-Agreement	Esperada.

	participante tiene puertos del tipo Request con QoSValues asociados.	por cada ServiceContract asociado a un QoSValue en el que se encuentre el participante asociado al proyecto. El documento contiene la información de las características de calidad asociadas al contrato. Los clientes generados invocan correctamente los servicios de los servidores generados cuando se ejecutan sobre JBoss en el IP/puerto seleccionado. Al ejecutarse el cliente, se imprime el tiempo de respuesta del servicio en el archivo de log del cliente.	
E1 - C2, C4, C5, C8, C10, C11, C13, C14, C15, C16, C17.	Existe un archivo XMI 2.1 correspondiente a un modelo SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante seleccionado. El participante tiene puertos del tipo Request con QoSValues asociados.	Se genera un proyecto web que utiliza JAX-WS + spring y un cliente para cada participante seleccionado. En todos los proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. También contienen una policy del tipo UsernameToken. Se genera un archivo Ws-Agreement por cada ServiceContract asociado a un QoSValue en el que se encuentre el participante asociado al proyecto. El documento contiene la información de las características de calidad asociadas al contrato. Los clientes generados invocan correctamente los servicios de los servidores generados cuando se ejecutan sobre Tomcat en el IP/puerto seleccionado. Al ejecutarse el cliente, se imprime el tiempo de respuesta del servicio en el archivo de log del cliente.	Esperada.

E1 - C2, C3, C6, C11, C13, C14, C16.	Existe un archivo XMI 2.1 correspondiente a un modelo SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante seleccionado. El participante tiene puertos del tipo Request con QoSValues asociados.	Se genera un proyecto de tipo "Java EJB Project" y un cliente, que contienen un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. Se genera un archivo Ws-Agreement por cada ServiceContract asociado a un QoSValue en el que se encuentre el participante asociado al proyecto. El documento contiene la información de las características de calidad asociadas al contrato.	Esperada.
E1 - C1, C3, C5, C7, C10, C11, C12, C14, C15, C16, C17	Existe un proyecto SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante seleccionado. El participante tiene puertos del tipo Request con QoSValues asociados.	Se genera un proyecto web que utiliza JAX-WS ri y un cliente para el participante seleccionado. En los dos proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. También contienen una policy del tipo UsernameToken. Se genera un archivo Ws-Agreement por cada ServiceContract asociado a un QoSValue. El documento contiene la información de las características de calidad asociadas al contrato. El cliente generado invoca correctamente los servicios del servidor generado cuando se ejecuta sobre Tomcat en el IP/puerto seleccionado. Al ejecutarse el cliente, se imprime el tiempo de respuesta del servicio en el	Esperada. Los proyectos, wsdl y documentos ws-agreement se generan correctamente. Los clientes consumen los servicios web correctamente, aunque el servidor Tomcat escribe en el log la siguiente advertencia cada vez que se invoca un servicio web: "WARNING: onComplete() failed for listener of type [org.apache.catalina.core.AsyncListenerWrapper] java.lang.IllegalStateException: It is illegal to call getRequest() after complete() or any of the dispatch()

		archivo de log del cliente.	methods has been called".
E1 - C2, C4, C5, C7, C10, C11, C12, C14, C15, C16, C17	Existe un archivo XMI 2.1 correspondiente a un modelo SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante seleccionado. El participante tiene puertos del tipo Request con QoSValues asociados.	Se genera un proyecto web que utiliza JAX-WS ri y un cliente para cada participante seleccionado. En todos los proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. También contienen una policy del tipo UsernameToken. Se genera un archivo Ws-Agreement por cada ServiceContract asociado a un QoSValue en el que se encuentre el participante asociado al proyecto. El documento contiene la información de las características de calidad asociadas al contrato. Los clientes generados invocan correctamente los servicios de los servidores generados cuando se ejecutan sobre Tomcat en el IP/puerto seleccionado. Al ejecutarse el cliente, se imprime el tiempo de respuesta del servicio en el archivo de log del cliente.	Esperada. Los proyectos, wsdl y documentos ws-agreement se generan correctamente. Los clientes consumen los servicios web correctamente, aunque el servidor Tomcat escribe en el log la siguiente advertencia cada vez que se invoca un servicio web: "WARNING: onComplete() failed for listener of type [org.apache.catalina.core.AsyncListenerWrapper] java.lang.IllegalStateException: It is illegal to call getRequest() after complete() or any of the dispatch() methods has been called".
E1 - C2, C4, C5, C7, C10, C11, C12, C14, C15, C16, C17, C18	Existe un archivo XMI 2.1 correspondiente a un modelo SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante	Se genera un proyecto web que utiliza JAX-WS ri y un cliente para cada participante seleccionado. En todos los proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. También contienen una policy del tipo UsernameToken.	Esperada. Los proyectos, wsdl y documentos ws-agreement se generan correctamente. Los clientes consumen los servicios web correctamente, aunque el servidor Tomcat escribe en el log la siguiente advertencia cada vez

	seleccionado. El participante tiene puertos del tipo Request con QoSValues asociados.	Se genera un archivo Ws-Agreement por cada ServiceContract asociado a un QoSValue en el que se encuentre el participante asociado al proyecto. El documento contiene la información de las características de calidad asociadas al contrato. Los clientes generados invocan correctamente los servicios de los servidores generados cuando se ejecutan sobre Tomcat en el IP/puerto seleccionado. Al ejecutarse el cliente, se imprime el tiempo de respuesta del servicio en el archivo de log del cliente.	que se invoca un servicio web: "WARNING: onComplete() failed for listener of type [org.apache.catalina.core.AsyncListenerWrapper] java.lang.IllegalStateException: It is illegal to call getRequest() after complete() or any of the dispatch() methods has been called".
E1 - C2, C4, C5, C8, C10, C11, C13, C14, C15, C16, C17, C18.	Existe un archivo XMI 2.1 correspondiente a un modelo SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante seleccionado. El participante tiene puertos del tipo Request con QoSValues asociados.	Se genera un proyecto web que utiliza JAX-WS + spring y un cliente para cada participante seleccionado. En todos los proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. También contienen una policy del tipo UsernameToken. Se genera un archivo Ws-Agreement por cada ServiceContract asociado a un QoSValue en el que se encuentre el participante asociado al proyecto. El documento contiene la información de las características de calidad asociadas al contrato. Los clientes generados invocan correctamente los servicios de los servidores generados cuando se ejecutan sobre Tomcat en el IP/puerto seleccionado. Al ejecutarse el cliente, se imprime el tiempo de respuesta del servicio en el	Esperada.

		archivo de log del cliente.	
E1 - C2, C4, C5, C7, C9, C11, C12, C14, C15, C16, C17, C18.	Existe un archivo XMI 2.1 correspondiente a un modelo SoaML+QoS con una estructura de elementos válida. El modelo tiene QoSValues asociado a un ServiceContract donde se encuentra el participante seleccionado. El participante tiene puertos del tipo Request con QoSValues asociados.	Se genera un proyecto web que utiliza JAX-WS ri y un cliente para cada participante seleccionado. En todos los proyectos se genera un archivo wsdl por cada puerto de tipo Service que tenga el participante. Los wsdl contienen las WS-Policies genéricas correspondiente a los QoSValues asociados al puerto del servicio que describe el wsdl. También contienen una policy del tipo UsernameToken. Se genera un archivo Ws-Agreement por cada ServiceContract asociado a un QoSValue en el que se encuentre el participante asociado al proyecto. El documento contiene la información de las características de calidad asociadas al contrato. Los clientes generados invocan correctamente los servicios de los servidores generados cuando se ejecutan sobre JBoss en el IP/puerto seleccionado. Al ejecutarse el cliente, se imprime el tiempo de respuesta del servicio en el archivo de log del cliente.	Esperada.

2.2.1.4. Evaluación de resultados

En todos los casos de prueba ejecutados los proyectos de tipo servidor y cliente son generados correctamente, con sus respectivos archivos WSDL y WS-Agreement. La validación de los datos de usuario y contraseña provistos en el cabezal SOAP se realiza correctamente. El log en el cliente con la información del tiempo de respuesta es generado correctamente.

El único problema que se detectó es cuando el proyecto servidor es generado para JAX-WS ri y Tomcat. Cuando un servicio web es invocado el servidor ejecuta correctamente el servicio, pero imprime la siguiente advertencia:

"*WARNING: onComplete() failed for listener of type [org.apache.catalina.core.AsyncListenerWrapper]*

java.lang.IllegalStateException: It is illegal to call getRequest() after complete() or any of the dispatch() methods has been called"

Proyecto de grado

Generación automática de Arquitecturas Orientadas a Servicios (SOAs) con SoaML y especificación de QoS

Casos de Uso

Federico Bertolini, Sofía Pérez, María Noel Quiñones

Tutor
Andrea Delgado

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay
Diciembre 2014

Contenido

1. Actores

1.1.Arquitecto de Software

Este actor usará el editor SoaML+QoS para especificar la arquitectura del sistema a modelar.

1.2.Usuario Diseñador

Este actor usará el editor SoaML+QoS para modelar sus contratos y arquitecturas de servicio, especificando los atributos y requerimientos de calidad de los servicios. Si solo se quiere especificar los atributos de calidad, el actor usará el editor QoS.

2. Casos de Uso

El plugin *SoaML Toolkit* debe proveer una solución integral para Eclipse que integre los plugins realizados y características QoS a modelos SoaML, así como la generación del código asociado. Para esto se identificaron varios casos de uso.

2.1.Diagramas de Casos de Uso

En la Figura 1 se muestran los casos de uso de modelado SoaML para los que no fue necesario realizar ninguna modificación.

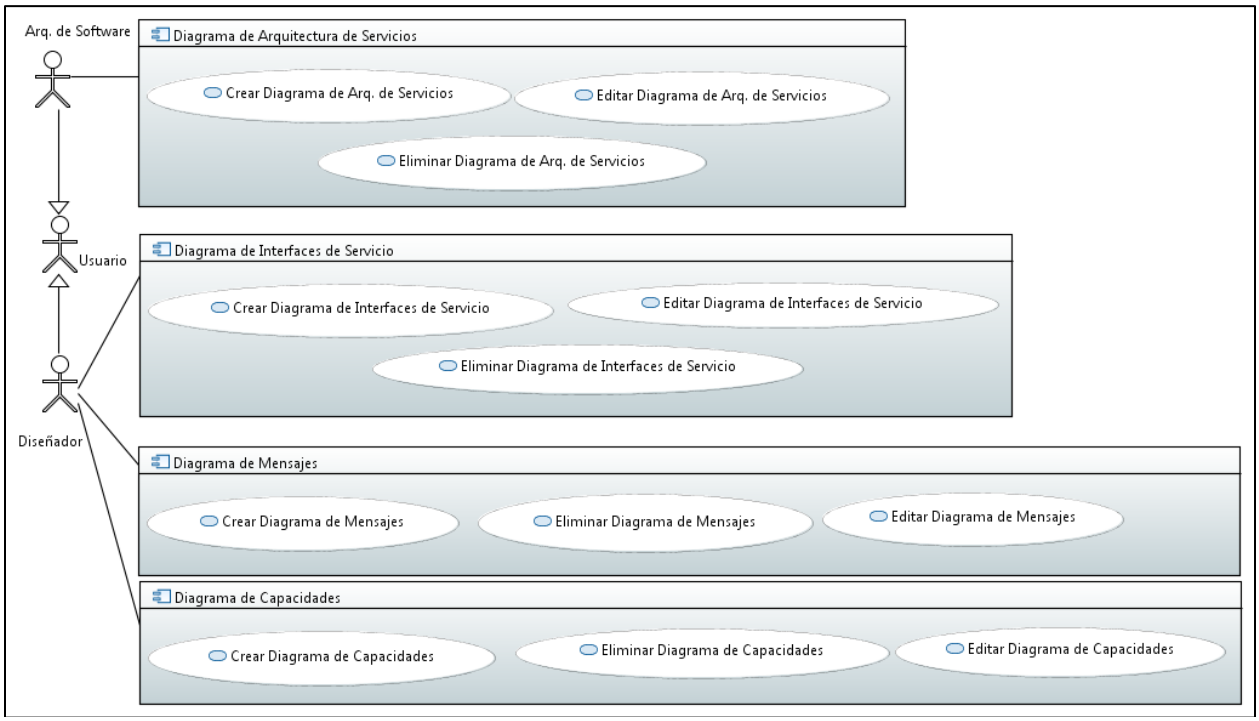

Figura 1. Casos de uso sin modificar

En la Figura 2 se ilustran los casos de uso de modelado SoaML y generación de código que requirieron modificaciones para agregar el modelado y generación de código de características de calidad.

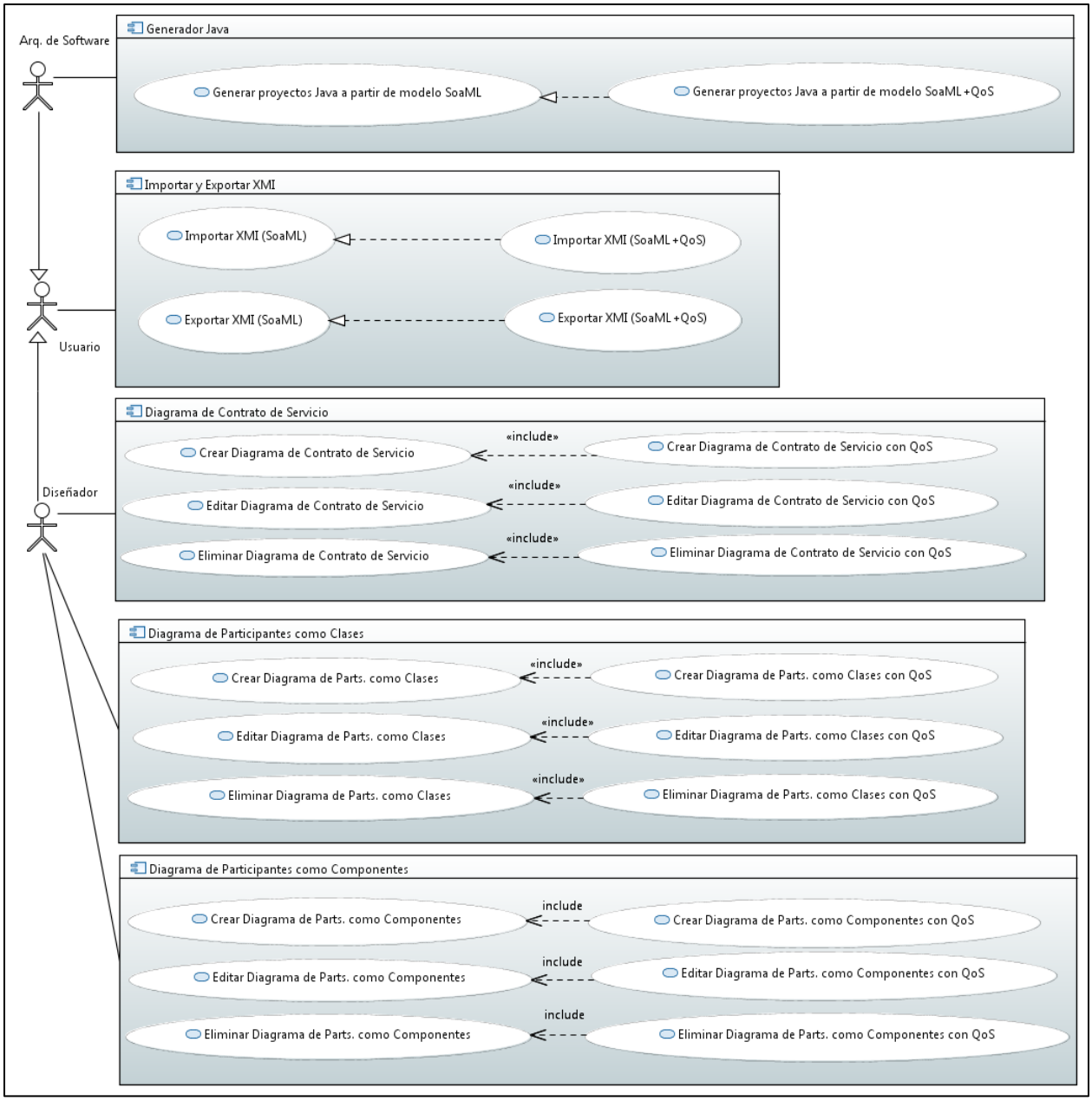

Figura 2. Casos de uso modificados

Finalmente, en la Figura 3 se presentan los casos de uso nuevos que fueron agregados para permitir la creación de modelos QoS.

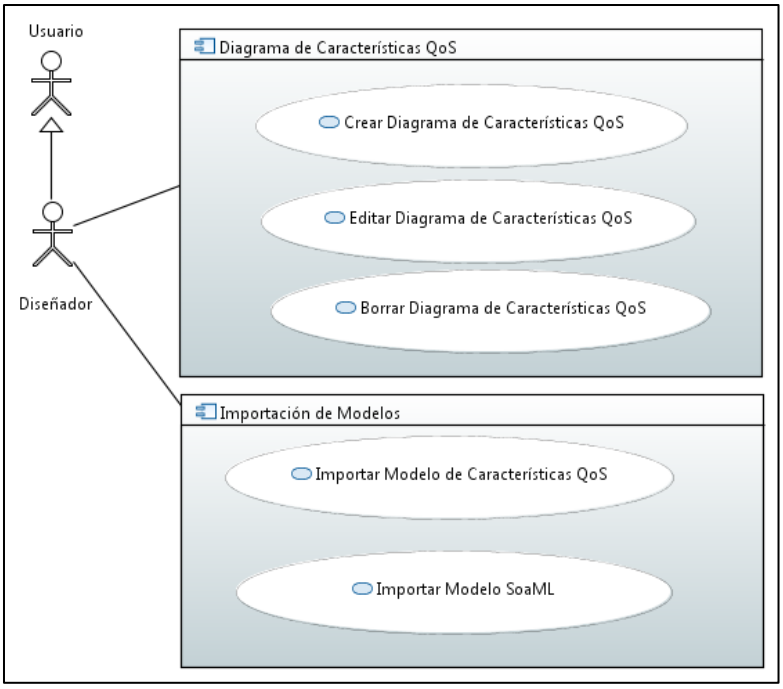

Figura 3. Casos de uso nuevos

2.2.Crear diagrama de Características QoS

2.2.1.Descripción

Permite al usuario crear un diagrama de tipo *QoS Characteristic* agregando elementos al lienzo desde la paleta de herramientas. Este diagrama permite definir las distintas características de calidad, las dimensiones que cuantifican dichas características así como también la categoría a la cual pertenece y las asociaciones que pueden haber entre distintas características. Se podrá agregar elementos tales como características, dimensiones y categorías así como también asociaciones y comentarios.

2.2.2.Pre-condiciones

Debe existir un proyecto creado para el lenguaje QoS o SoaML+QoS.

2.2.3.Flujo de eventos principales

1. El usuario crea un diagrama de tipo QoSCharacteristic.
2. El usuario agrega elementos al diagrama eligiendo entre:
    1. Agregar un elemento QoSCategory.
    2. Agregar un elemento QoSCharacteristic dentro de una QoSCategory previamente definida.
    3. Agregar un elemento QoSDimension dentro de un elemento QoSCharacteristic donde le asigna un tipo y especifica el valor del atributo *unit* y opcionalmente de los atributos *statisticalQualifier* y *direction,* dentro de un elemento QoSCharacteristic previamente definido.
    4. Agregar una generalización entre dos elementos QoSCharacteristic previamente definidos.

3. El usuario decide seguir agregando elementos al diagrama. Vuelve al paso 2.

2.2.4. Flujo de eventos alternativos

Flujo alternativo: El usuario agrega un elemento QoSCharacteristic directamente en el lienzo.

2.2a.1. El usuario agrega un elemento QoSCharacteristic en el lienzo.
2.2a.2. Continúa en el paso 3 del flujo principal.

Flujo alternativo: El usuario decide no seguir agregando elementos al diagrama.

3.a.1. El usuario decide no seguir agregando elementos al diagrama.
3.a.2. Fin del caso de uso

2.2.5. Post-condiciones

Se crea un diagrama de tipo *QoSCharacteristic* con elementos QoSCharacteristic, QoSDimension y QoSCategory y asociaciones correspondientes.

2.3. Importar modelo de Características QoS

2.3.1. Descripción

Permite al usuario importar un modelo de características QoS a un modelo SoaML+QoS, realizando una copia en profundidad de todos los elementos del modelo origen al modelo destino.

2.3.2. Pre-condiciones

Debe existir un proyecto creado para el lenguaje SoaML+QoS. Debe existir un archivo UML correspondiente a un modelo del lenguaje QoS.

2.3.3. Flujo de eventos principales

1. El usuario abre el menú contextual sobre el modelo y selecciona la opción “Import QoS Elements from QoS Model”.
2. El usuario selecciona un archivo UML correspondiente a un modelo QoS.

2.3.4. Flujo de eventos alternativos

No tiene.

2.3.5. Post-condiciones

Se agregan todos los elementos del modelo QoS seleccionado al modelo SoaML+QoS actual.

2.4. Importar modelo SoaML

2.4.1. Descripción

Permite al usuario importar un modelo SoaML a un modelo SoaML+QoS, realizando una copia en profundidad de todos los elementos del modelo origen al modelo destino.

2.4.2. Pre-condiciones

Debe existir un proyecto creado para el lenguaje SoaML+QoS. Debe existir un archivo UML correspondiente a un modelo del lenguaje SoaML.

2.4.3. Flujo de eventos principales

1. El usuario abre el menú contextual sobre el modelo y selecciona la opción “Import SoaML Elements from SoaML Model”.
2. El usuario selecciona un archivo UML correspondiente a un modelo SoaML.

2.4.4. Flujo de eventos alternativos

No tiene.

2.4.5. Post-condiciones

Se agregan todos los elementos del modelo SoaML seleccionado al modelo SoaML+QoS actual.

2.5. Crear diagrama de Contrato de Servicios con QoS

2.5.1. Descripción

Permite al usuario crear un diagrama de tipo *Service Contract with QoS*, agregando elementos al lienzo desde la paleta de herramientas del diagrama. Se pueden agregar elementos tales como contratos de servicios, roles y usos de contratos de servicios. También se pueden agregar asociaciones entre roles y especificar los roles que cumplen ciertos roles en contratos de servicios asociados. El usuario también puede agregar características y valores de calidad de servicios, y asociaciones entre valores de calidad y contratos de servicios.

2.5.2. Pre-condiciones

1. Debe existir un proyecto para el lenguaje SoaML+QoS.
2. El modelo debe contener al menos un elemento QoSCharacteristic con al menos un QoSDimension.

2.5.3. Flujo de eventos principales

1. Incluye Caso de Uso 2.3 de [1] seleccionando crear diagrama Service Contract with QoS.
2. El usuario agrega elementos a un diagrama Service Contract, eligiendo entre:
    1. Agregar un elemento QoSValue, especificando opcionalmente la QoSCharacteristic de la cual es instancia.
    2. Arrastrar un elemento QoSCharacteristic desde el árbol del modelo hasta un elemento QoSValue sin una QoSCharacteristic asociada. El usuario selecciona qué QoSDimensions de la QoSCharacteristic seleccionada implementa el elemento QoSValue.
    3. Agregar un elemento Slot dentro de un elemento QoSValue, especificando un valor y la propiedad que implementa el Slot.
    4. Agregar una dependencia de tipo QoSContract entre un contrato de servicio y un elemento QoSValue previamente definidos.
3. El usuario decide seguir agregando elementos al diagrama. Vuelve al paso 2.

2.5.4. Flujo de eventos alternativos

Flujo alternativo: El usuario decide no seguir agregando elementos al diagrama.

3.a.1. El usuario decide no seguir agregando elementos al diagrama.

3.a.2. Fin del caso de uso

2.5.5. Post-condiciones

Se crea un diagrama de tipo *Service Contract with QoS* conteniendo los elementos de tipo Role, ServiceContract Use, QoSValue, QoSContract y asociaciones definidas para dicho contrato.

2.6. Crear diagrama de Participantes como Clases con QoS

2.6.1. Descripción

Permite al usuario crear un diagrama de tipo *Participant Class with QoS*. Es una especificación a alto nivel en la que se pueden agregar al lienzo elementos participante y sus puertos y sus comportamientos asociados. El usuario también puede agregar características y valores de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios.

2.6.2. Pre-condiciones

1. Debe existir un proyecto para el lenguaje SoaML+QoS.
2. El modelo debe contener al menos un elemento QoSCharacteristic con al menos un QoSDimension.

2.6.3. Flujo de eventos principales

1. Incluye Caso de Uso 2.6 de [1] seleccionando crear Participant Class with QoS.
2. El usuario agrega elementos al diagrama eligiendo entre:
    1. Agregar un elemento QoSValue, especificando opcionalmente la QoSCharacteristic de la cual es instancia.
    2. Arrastrar un elemento QoSCharacteristic desde el árbol del modelo hasta un elemento QoSValue sin una QoSCharacteristic asociada. El usuario selecciona qué QoSDimensions de la QoSCharacteristic seleccionada implementa el elemento QoSValue.
    3. Agregar un elemento Slot dentro de un elemento QoSValue, especificando un valor y la propiedad que implementa el Slot.
    4. Agregar una dependencia QoSRequired desde un puerto de tipo Request a un QoSValue previamente definidos.
    5. Agregar una dependencia QoSOffered desde un puerto de tipo Service a un QoSValue previamente definidos.
3. El usuario decide seguir agregando elementos al diagrama. Vuelve al paso 2.

2.6.4. Flujo de eventos alternativos

Flujo alternativo: El usuario decide no seguir agregando elementos al diagrama.

3.a.1. El usuario decide no seguir agregando elementos al diagrama.

3.a.2. Fin del caso de uso.

2.6.5. Post-condiciones

Se crea un diagrama de tipo *Participant Class with QoS* con elementos de tipo Participant conteniendo los elementos de tipo Port y diagramas de secuencia adjuntos, así como también los elementos de restricciones de calidad de servicios asignados a los puertos y sus valores correspondientes.

2.7. Crear diagrama de Participantes como Componentes con QoS

2.7.1. Descripción

Permite al usuario crear un diagrama de tipo *Participant Component with QoS* que indica la implementación de participantes como componentes. Para esto el usuario puede agregar elementos al lienzo desde la paleta de herramientas del diagrama. Se pueden agregar componentes participante y dentro de ellos elementos tales como proveedores de servicios, puertos, asociaciones y canales de servicio entre los puertos, capacidades y sus dependencias con los participantes existentes en el diagrama, usos de arquitecturas de servicios y los correspondientes rolebindings con los proveedores de servicios asociados. El usuario también puede agregar características y valores de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios.

2.7.2. Pre-condiciones

1. Debe existir un proyecto para el lenguaje SoaML+QoS.
2. El modelo debe contener al menos un elemento QoSCharacteristic con al menos un QoSDimension.

2.7.3. Flujo de eventos principales

1. Incluye Caso de Uso 2.7 de [1] seleccionando crear Participant Component with QoS.
2. El usuario agrega elementos al diagrama eligiendo entre:
    1. Agregar un elemento QoSValue, especificando opcionalmente la QoSCharacteristic de la cual es instancia.
    2. Arrastrar un elemento QoSCharacteristic desde el árbol del modelo hasta un elemento QoSValue sin una QoSCharacteristic asociada. El usuario selecciona qué QoSDimensions de la QoSCharacteristic seleccionada implementa el elemento QoSValue.
    3. Agregar un elemento Slot dentro de un elemento QoSValue, especificando un valor y la propiedad que implementa el Slot.
    4. Agregar una dependencia QoSRequired desde un puerto de tipo Request a un QoSValue previamente definidos.
    5. Agregar una dependencia QoSOffered desde un puerto de tipo Service a un QoSValue previamente definidos.
3. El usuario decide seguir agregando elementos al diagrama. Vuelve al paso 2.

2.7.4. Flujo de eventos alternativos

Flujo alternativo: El usuario decide no seguir agregando elementos al diagrama.

3.a.1. El usuario decide no seguir agregando elementos al diagrama.
3.a.2. Fin del caso de uso.

2.7.5. Post-condiciones

Se crea un diagrama de tipo *Participant Component with QoS* con elementos de tipo Participante conteniendo los elementos de tipo Puerto, diagramas de secuencia adjuntos, componentes internos y sus respectivos puertos, Canales de Servicio que vinculan los puertos de los componentes, así como también los elementos de restricciones de calidad de servicios asignados a los puertos y sus valores correspondientes.

2.8. Editar diagrama de Características QoS

2.8.1. Descripción

Permite al usuario editar un diagrama de tipo *QoS Characteristic* existente agregando/borrando/modificando elementos del diagrama.

2.8.2. Pre-condiciones

Debe existir un diagrama de tipo QoS Characteristic.

2.8.3. Flujo de eventos principales

1. El usuario selecciona un diagrama de tipo QoS Characteristic existente para editar.
2. El usuario agrega/borra/modifica elementos del diagrama eligiendo entre:
    1. Agregar un elemento QoSCategory.
    2. Agregar un elemento QoSCharacteristic dentro de una QoSCategory previamente definida.
    3. Agregar un elemento QoSDimension dentro de un elemento QoSCharacteristic donde le asigna un tipo y especifica el valor del atributo *unit* y opcionalmente de los atributos *statisticalQualifier* y *direction,* dentro de un elemento QoSCharacteristic previamente definido.
    4. Agregar una generalización entre dos elementos QoSCharacteristic previamente definidos.
    5. Modificar un elemento QoSDimension de un elemento QoSCharacteristic existente.
    6. Modificar un elemento QoSCategory.
    7. Modificar un elemento QoSCharacteristic.
    8. Borrar una generalización entre dos elementos QoSCharacteristic.
    9. Borrar un elemento QoSCategory. Se eliminan todas las características incluidas en esa categoría así como las generalizaciones definidas.
    10. Borrar un elemento QoSCharacteristic.
    11. Borrar un elemento QoSDimension de una característica existente.
3. El usuario decide seguir agregando/borrando/modificando elementos al diagrama. Vuelve al paso 2.

2.8.4. Flujo de eventos alternativos

Flujo alternativo: El usuario agrega un elemento QoSCharacteristic directamente en el lienzo.

2.2a.1. El usuario agrega un elemento QoSCharacteristic en el lienzo.

2.2a.2. Sigue en el paso 3.

Flujo alternativo: El usuario decide no seguir agregando elementos al diagrama.

3.a.1. El usuario decide no seguir agregando elementos al diagrama.
3.a.2. Fin del caso de uso.

2.8.5. Post-condiciones

Se modifica un diagrama de tipo *QoS Characteristic* ya sea agregando nuevos elementos, borrando o modificando elementos existentes en el diagrama.

2.9. Editar diagrama de Contrato de Servicios con QoS

2.9.1. Descripción

Permite al usuario editar un diagrama de tipo *Service Contract with QoS* existente agregando desde la paleta de herramientas del diagrama nuevos contratos de servicio al lienzo y agregar nuevos roles, usos de contratos de servicios y asociaciones dentro de un contrato de servicios existente, así como también puede agregar y modificar características y valores de calidad de servicios, y asociaciones entre valores de calidad y contratos de servicios. El usuario puede también modificar los tipos de los elementos existentes o crear nuevos elementos para asignar como tipos de otros elementos. En todo momento el usuario puede eliminar elementos o asociaciones existentes en el diagrama.

2.9.2. Pre-condiciones

Debe existir un diagrama de tipo Service Contract with QoS.

2.9.3. Flujo de eventos principales

1. Incluye Caso de Uso 2.11 de [1] seleccionando un diagrama Service Contract with QoS.
2. El usuario agrega/borra/modifica elementos del diagrama eligiendo entre:
    1. Agregar un elemento QoSValue, especificando opcionalmente la QoSCharacteristic de la cual es instancia.
    2. Arrastrar un elemento QoSCharacteristic desde el árbol del modelo hasta un elemento QoSValue sin una QoSCharacteristic asociada. El usuario selecciona qué QoSDimensions de la QoSCharacteristic seleccionada implementa el elemento QoSValue.
    3. Agregar un elemento Slot dentro de un elemento QoSValue, especificando un valor y la propiedad que implementa el Slot.
    4. Agregar una dependencia de tipo QoSContract entre un contrato de servicio y un elemento QoSValue previamente definidos.
    5. Modificar un elemento QoSValue existente en el diagrama.
    6. Borrar una dependencia de tipo QoSContract existente en el diagrama.
    7. Borrar un elemento QoSValue. En caso de existir se eliminan las dependencias de tipo QoSContract asociadas al QoSValue eliminado.
3. El usuario decide seguir agregando/borrando/modificando elementos al diagrama. Vuelve al paso 2.

2.9.4. Flujo de eventos alternativos

Flujo alternativo: El usuario decide no seguir agregando elementos al diagrama.

3.a.1. El usuario decide no seguir agregando elementos al diagrama.
3.a.2. Fin del caso de uso.

2.9.5. Post-condiciones

Se modifica un diagrama de tipo *Service Contract with QoS* ya sea agregando nuevos elementos, borrando o modificando elementos existentes en el diagrama.

2.10. Editar diagrama de Participantes como Clases con QoS

2.10.1. Descripción

Permite al usuario editar un diagrama de tipo *Participant Class with QoS* existente agregando nuevos elementos participante al lienzo y agregando nuevos puertos service y request a un elemento participante existente en el lienzo. Además, el usuario podrá asignar nuevos tipos a los puertos contenidos en el lienzo, crear nuevos tipos para asignar a los puertos o eliminar asignaciones de tipos existentes. El usuario también puede agregar y modificar características y valores de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios. En todo momento el usuario podrá eliminar elementos contenidos en el diagrama.

2.10.2. Pre-condiciones

Debe existir un diagrama de tipo Participant Class with QoS.

2.10.3. Flujo de eventos principales

1. Incluye Caso de Uso 2.14 de [1] seleccionando un diagrama Participant Class with QoS.
2. El usuario agrega/borra/modifica elementos al diagrama eligiendo entre:
    1. Agregar un elemento QoSValue, especificando opcionalmente la QoSCharacteristic de la cual es instancia.
    2. Arrastrar un elemento QoSCharacteristic desde el árbol del modelo hasta un elemento QoSValue sin una QoSCharacteristic asociada. El usuario selecciona qué QoSDimensions de la QoSCharacteristic seleccionada implementa el elemento QoSValue.
    3. Agregar un elemento Slot dentro de un elemento QoSValue, especificando un valor y la propiedad que implementa el Slot.
    4. Agregar una dependencia QoSRequired desde un puerto de tipo Request a un QoSValue previamente definidos.
    5. Agregar una dependencia QoSOffered desde un puerto de tipo Service a un QoSValue previamente definidos.
    6. Modificar un elemento QoSValue existente en el diagrama.
    7. Borrar una dependencia QoSRequired.
    8. Borrar una dependencia QoSOffered.
    9. Borrar un elemento QoSValue. En caso de existir se eliminan las dependencia de tipo QoSRequired/QoSOffered asociadas al QoSValue eliminado.
3. El usuario decide seguir agregando elementos al diagrama. Vuelve al paso 2.

2.10.4. Flujo de eventos alternativos

Flujo alternativo: El usuario decide no seguir agregando elementos al diagrama.

3.a.1. El usuario decide no seguir agregando elementos al diagrama.
3.a.2. Fin del caso de uso.

2.10.5. Post-condiciones

Se modifica un diagrama de tipo *Participant Class with QoS* ya sea agregando nuevos elementos, borrando o modificando elementos existentes en el diagrama.

2.11. Editar diagrama de Participantes como Componentes con QoS

2.11.1. Descripción

Permite al usuario editar un diagrama de tipo *Participant Component with QoS*, agregando nuevos componentes participante al lienzo y agregando nuevos elementos dentro de participantes existentes tales como proveedores de servicios, puertos, dependencias, asociaciones, canales de servicio, capacidades, usos de arquitecturas de servicios y los correspondientes rolebindings con los proveedores de servicios asociados. Además el usuario puede modificar los puertos, proveedores de servicios y capacidades usos de arquitecturas de servicios existentes asignándoles nuevos tipos o borrando la asignación de tipos existente. El usuario también puede agregar y modificar características y valores de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios. En todo momento el usuario podrá eliminar elementos contenidos en el diagrama.

2.11.2. Pre-condiciones

Debe existir un diagrama de tipo Participant Component with QoS.

2.11.3. Flujo de eventos principales

1. Incluye Caso de Uso 2.15 de [1] seleccionando un diagrama Participant Component with QoS.
2. El usuario agrega/borra/modifica elementos al diagrama eligiendo entre:
    1. Agregar un elemento QoSValue, especificando opcionalmente la QoSCharacteristic de la cual es instancia.
    2. Arrastrar un elemento QoSCharacteristic desde el árbol del modelo hasta un elemento QoSValue sin una QoSCharacteristic asociada. El usuario selecciona qué QoSDimensions de la QoSCharacteristic seleccionada implementa el elemento QoSValue.
    3. Agregar un elemento Slot dentro de un elemento QoSValue, especificando un valor y la propiedad que implementa el Slot.
    4. Agregar una dependencia QoSRequired desde un puerto de tipo Request a un QoSValue previamente definidos.
    5. Agregar una dependencia QoSOffered desde un puerto de tipo Service a un QoSValue previamente definidos.
    6. Modificar un elemento QoSValue existente en el diagrama.
    7. Borrar una dependencia QoSRequired.
    8. Borrar una dependencia QoSOffered.

9. Borrar un elemento QoSValue. En caso de existir se eliminan las dependencia de tipo QoSRequired/QoSOffered asociadas al QoSValue eliminado.

3. El usuario decide seguir agregando elementos al diagrama. Vuelve al paso 2.

2.11.4. Flujo de eventos alternativos

Flujo alternativo: El usuario decide no seguir agregando elementos al diagrama.

3.a.1. El usuario decide no seguir agregando elementos al diagrama.

3.a.2. Fin del caso de uso.

2.11.5. Post-condiciones

Se modifica un diagrama de tipo *Participant Component with QoS* ya sea agregando nuevos elementos, borrando o modificando elementos existentes en el diagrama.

2.12. Eliminar diagrama de Características QoS

2.12.1. Descripción

Permite al usuario eliminar un diagrama preexistente de tipo *QoS Characteristic*, esto es eliminar cada uno de los elementos contenidos en el diagrama y las referencias existentes a ellos en otros diagramas.

2.12.2. Pre-condiciones

Debe existir un diagrama de tipo QoS Characteristic.

2.12.3. Flujo de eventos principales

1. El usuario selecciona un diagrama de tipo QoS Characteristic existente en el modelo.
2. El usuario elimina todos los elementos del diagrama seleccionado.
3. El usuario elimina el diagrama seleccionado.

2.12.4. Flujo de eventos alternativos

No tiene.

2.12.5. Post-condiciones

Se elimina el diagrama del modelo y todas las referencias a los elementos eliminados.

2.13. Eliminar diagrama de Contrato de Servicios con QoS

2.13.1. Descripción

Permite al usuario eliminar un diagrama preexistente de tipo *Service Contract with QoS*, esto es eliminar cada uno de los elementos contenidos en el diagrama y las referencias existentes a ellos en otros diagramas.

2.13.2. Pre-condiciones

Debe existir un diagrama de tipo Service Contract Diagram with QoS.

2.13.3. Flujo de eventos principales

1. Incluye Caso de Uso 2.18 de [1] seleccionando un diagrama Service Contract with QoS.

2.13.4. Flujo de eventos alternativos
No tiene.

2.13.5. Post-condiciones
Se elimina el diagrama del modelo y todas las referencias a los elementos eliminados.

2.14. Eliminar diagrama de Participantes como Clases con QoS

2.14.1. Descripción
Permite al usuario eliminar un diagrama preexistente de tipo Participant Class with QoS, esto es eliminar cada uno de los elementos contenidos en el diagrama y las referencias existentes a ellos en otros diagramas.

2.14.2. Pre-condiciones
Debe existir un diagrama de tipo Participant Class with QoS.

2.14.3. Flujo de eventos principales
1. Incluye Caso de Uso 2.21 de [1] seleccionando un diagrama Participant Class with QoS.

2.14.4. Flujo de eventos alternativos
No tiene.

2.14.5. Post-condiciones
Se elimina el diagrama del modelo y todas las referencias a los elementos eliminados.

2.15. Eliminar diagrama de Participantes como Componentes con QoS

2.15.1. Descripción
Permite al usuario eliminar un diagrama preexistente de tipo Participant Component with QoS, esto es eliminar cada uno de los elementos contenidos en el diagrama y las referencias existentes a ellos en otros diagramas.

2.15.2. Pre-condiciones
Debe existir un diagrama de tipo Participant Component with QoS.

2.15.3. Flujo de eventos principales
1. Incluye Caso de Uso 2.22 de [1] seleccionando un diagrama Participant Component with QoS.

2.15.4. Flujo de eventos alternativos
No tiene.

2.15.5. Post-condiciones
Se elimina el diagrama del modelo y todas las referencias a los elementos eliminados.

2.16. Importar Modelo desde archivo con formato XMI

2.16.1. Descripción

Permite al usuario importar un modelo SoaML o SoaMLQoS desde un archivo XML con formato XMI, de forma de que se generen en el árbol del modelo los elementos contenidos en el archivo importado con sus estereotipos asociados.

2.16.2. Pre-condiciones

Debe existir un modelo SoaML o SoaMLQoS.

2.16.3. Flujo de eventos principales

1. El usuario selecciona la opción “Import from XMI” dentro del menú SoaML Toolkit.
2. El usuario selecciona un archivo XMI donde está definido el modelo SoaMLQoS.
3. El editor traduce el archivo y despliega los elementos en el árbol de elementos del modelo.
4. Fin del caso de uso.

2.16.4. Flujo de eventos alternativos

Flujo alternativo: Importar archivo XMI con modelo SoaML.

2.a.1 El usuario selecciona un archivo XMI donde está definido el modelo SoaML.

2.a.2 Sigue en el paso 3 del flujo principal.

Flujo alternativo: Problemas al cargar el modelo.

3.a.1 El editor encuentra problemas para cargar los datos contenidos en el archivo XMI, y despliega un mensaje de error avisando del problema.

3.a.2 Fin del Caso de Uso.

2.16.5. Post-condiciones

Se crea un nuevo modelo conteniendo los elementos definidos en el archivo XMI.

2.17. Exportar Modelo a archivo con formato XMI

2.17.1. Descripción

Permite al usuario exportar un modelo SoaML o SoaMLQoS con sus elementos y estereotipos asociados desde el editor a un archivo XML con formato XMI.

2.17.2. Pre-condiciones

Debe existir un modelo SoaML o SoaMLQoS.

2.17.3. Flujo de eventos principales

1. El usuario selecciona la opción “Export to XMI” dentro del menú SoaML Toolkit.
2. El usuario selecciona el archivo UML donde está definido el modelo SoaMLQoS.
3. El usuario indica el nombre y la ruta donde se guardará el archivo XMI.

4. El editor traduce y persiste en un archivo XMI los elementos del modelo que están definidos en el archivo UML seleccionado.
5. Fin del caso de uso.

2.17.4. Flujo de eventos alternativos

Flujo alternativo: Seleccionar modelo SoaML.

2.a.1 El usuario selecciona el archivo UML donde está definido el modelo SoaML.

2.a.2 Sigue en el paso 3 del flujo principal.

Flujo alternativo: Problemas al persistir el modelo.

3.a.1 El editor indica al usuario que encuentra problemas para persistir los datos y despliega un mensaje de error avisando del problema.

3.a.2 Fin del Caso de Uso.

2.17.5. Post-condiciones

Se crea un nuevo archivo con esquema XMI, donde se encuentra almacenado los elementos del modelo.

2.18. Generar proyectos Java a partir de modelo SoaML+QoS

2.18.1. Descripción

Permite al usuario (arquitecto del sistema) generar proyectos java a partir de modelos SoaML+QoS. El modelo puede estar en formato XMI o en formato UML, y se permite generar distintos tipos de proyectos que exponen los servicios web especificados en el modelo y los clientes de estos servicios. También permite generar contratos WS-Agreement a partir de las características de calidad asociadas a los contratos especificados en el modelo, y permite generar archivos WSDL con la descripción de los servicios, y con políticas WS-Policy a partir de las características de calidad asociadas a los puertos de servicio en el modelo.

2.18.2. Pre-condiciones

El archivo XMI o UML seleccionado debe tener un paquete con al menos un participante, un paquete con al menos un mensaje, un paquete con al menos una interfaz de servicio y un contrato de servicio.

2.18.3. Flujo de eventos principales

1. El usuario selecciona la opción “Generate Java from UML” dentro del menú SoaML Toolkit.
2. El usuario selecciona un proyecto Papyrus.
3. El usuario selecciona los participantes para los cuales desea generar código Java.
4. El usuario selecciona que desea generar servidores para los participantes seleccionados anteriormente.
5. El usuario selecciona que desea generar proyectos del tipo “Dynamic Web Project (war)”.
6. El usuario selecciona que los servicios web generados sean implementados con “JAX-WS ri”.
7. El usuario selecciona el tipo de servidor (JBoss o Tomcat).

8. El usuario selecciona que desea generar clientes para los participantes seleccionados anteriormente.
9. El usuario selecciona la dirección I.P. y el puerto donde se publicarán los servicios web.
10. El usuario selecciona que desea generar documentos WS-Agreement.
11. El usuario selecciona que desea generar políticas WS-Policy.
12. Para cada característica listada el usuario selecciona uno de estos tipos de generación: no generar, WS-Policy Response Time, WS-Policy Generic o WS-Security Username Token.

2.18.4. Flujo de eventos alternativos

Flujo alternativo: El usuario selecciona la opción "XMI 2.0" dentro del menú "Generate Java from XMI".

1.a.1. El usuario selecciona la opción "Generate Java from XMI" y luego "XMI 2.0".
1.a.2. El usuario selecciona un archivo del tipo XMI 2.0.
1.a.3. Vuelve al paso 3 del flujo principal.

Flujo alternativo: El usuario selecciona la opción "XMI 2.1" dentro del menú "Generate Java from XMI".

1.b.1. El usuario selecciona la opción "Generate Java from XMI" y luego "XMI 2.1".
1.b.2. El usuario selecciona un archivo del tipo XMI 2.1.
1.b.3. Vuelve al paso 3 del flujo principal.

Flujo alternativo: El usuario selecciona que no desea generar servidores.

4.a.1. El usuario selecciona que no desea generar servidores para los participantes seleccionados anteriormente.
4.a.2. Vuelve al paso 8 del flujo principal.

Flujo alternativo: El usuario selecciona que desea generar proyectos del tipo "Java EJB Project (jar)".

5.a.1. El usuario selecciona que desea generar proyectos del tipo "Java EJB Project (jar)".
5.a.2. Vuelve al paso 8 del flujo principal.

Flujo alternativo: El usuario no selecciona que los servicios web generados sean implementados con "JAX-WS ri".

6.a.1. El usuario selecciona que los servicios web generados sean implementados con "JAX-WS + spring".
6.a.2. Vuelve al paso 8 del flujo principal.

Flujo alternativo: El usuario selecciona que no desea generar clientes.

8.a.1. El usuario selecciona que no desea generar clientes para los participantes seleccionados anteriormente.
8.a.2. Si el usuario seleccionó que desea generar servidores entonces vuelve al paso 9 del flujo principal. Si el usuario seleccionó que no desea generar servidores entonces vuelve al paso 4 del flujo principal.

Flujo alternativo: El usuario selecciona que no desea generar documentos WS-Agreement.

10.a.1. El usuario selecciona que no desea generar documentos WS-Agreement.

10.a.2. Vuelve al paso 11 del flujo principal.

Flujo alternativo: El modelo no contiene características QoS.

10.b.1. Fin de caso de uso.

Flujo alternativo: El usuario selecciona que no desea generar características QoS.

11.a.1. El usuario selecciona que no desea generar características QoS.

11.a.2. Fin de caso de uso.

2.18.5. Post-condiciones

Si el usuario seleccionó generar servidores, para cada participante seleccionado se crea un proyecto Java del tipo seleccionado con un conjunto de archivos WSDL que describen los servicios que expone el servidor. Si el usuario seleccionó generar documentos WS-Agreement, se genera un documento de este tipo por cada ServiceContract con elementos QoSValue asociados en los que estuviera el participante; el documento contiene la información de las características de calidad asociadas al contrato. Si el usuario seleccionó generar características QoS, el archivo WSDL tendrá una WS-Policy del tipo seleccionado. Si el usuario seleccionó generar clientes, para cada participante seleccionado se crea un proyecto Java con el cual se pueden consumir los servicios provistos por el servidor correspondiente.

3. Referencias

[1] Sofía Larroca Andrés Pastorini. Documento Modelo de Casos de Uso - Proyecto de Grado Implementación del Estándar de Modelado de Servicios SoaML como Plugin de Eclipse. *Facultad de Ingeniería, Universidad de la República*.

Proyecto de grado

Generación automática de Arquitecturas Orientadas a Servicios (SOAs) con SoaML y especificación de QoS

Documento técnico

Federico Bertolini, Sofía Pérez, María Noel Quiñones

Tutor
Andrea Delgado

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay
Diciembre 2014

Resumen

Este documento brinda la descripción detallada de cómo extender el módulo SoaML2Code del plugin SoaML Toolkit para agregar más implementaciones de características de calidad.

Contenido

1. Modificación de UI

En la página 4 del *wizard* del generador de código se muestran las distintas implementaciones de características de calidad para que el usuario pueda mapearlas a los elementos QoSCharacteristic y QoSDimension definidas en su modelo y así seleccionar las que desea generar. Esto se observa en la Figura 1.

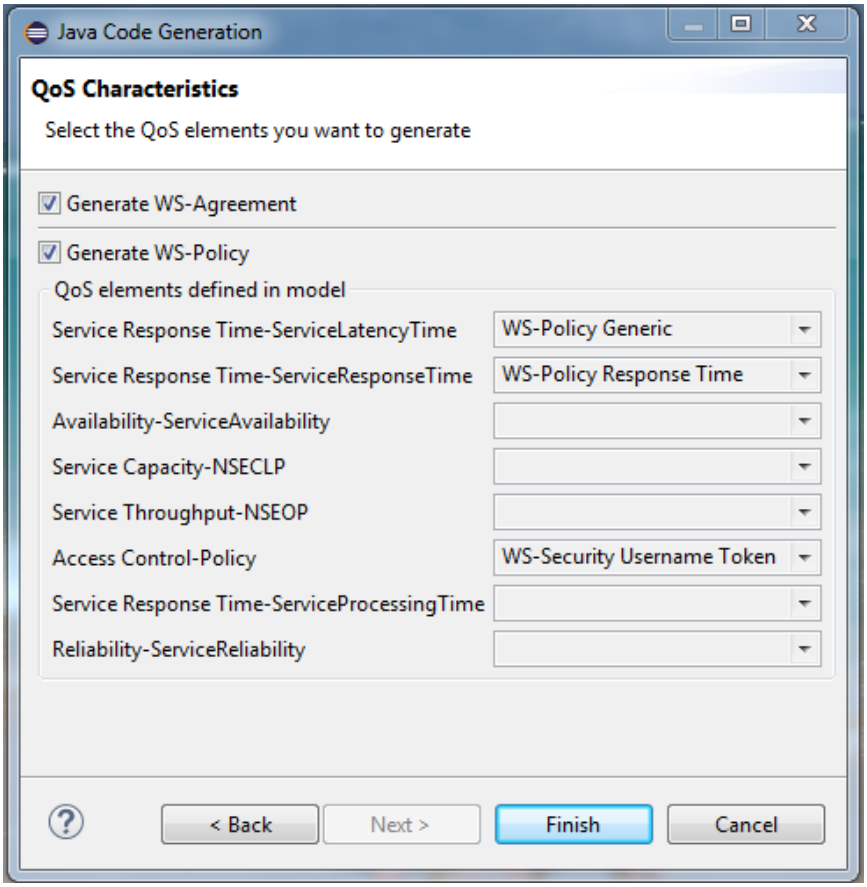

Figura 1. Página 4 del wizard del generador de código – QoS

Actualmente están disponibles la generación de WS-Agreement, WS-Policy Generic, WS-Policy Response Time y WS-Security Username Token.

Si el usuario desea agregar nuevas implementaciones de WS-Policy debe seguir los siguientes pasos:

1. Agregar los valores para las nuevas implementaciones en el enumerado *QoSDimensionType* que se encuentra en el paquete *parser.object*.
2. Modificar el constructor de la clase *QoSPage*, ubicada en el paquete *userInterface*, agregando al *hashmap implementedQoS* las nuevas implementaciones, utilizando como clave el nombre de la implementación que se desea mostrar en el *wizard* y como valor el agregado en el enumerado en el punto anterior. Por ejemplo, si se agregó el valor *NewPolicy* al enumerado en el punto anterior y se desea que en el *wizard* aparezca con el nombre *WS-Policy New*, entonces al constructor de la clase *QoSPage* se deberá agregar la siguiente línea:

```
implementedQoS.put("WS-Policy New", QoSDimensionType.NewPolicy);
```

Si lo que desea el usuario es agregar una implementación de características de calidad utilizando otro estándar, es posible hacerlo modificando el método *createControl* de la clase *QoSPage*. Allí podrá agregar, por ejemplo, otro *checkbox* que indique si se desea generar o no esta nueva implementación, al igual que aparecen las opciones *Generate WS-Agreement* y *Generate WS-Policy*.

Para agregar nuevas páginas al *wizard*, si por ejemplo se desea dividir en dos la página de QoS, se deberá implementar una clase que extienda a *WizardPage*, por cada nueva página a mostrar y ésta deberá ser agregada a la clase *CodeGenerationWizard* como se indica a continuación:

1. Agregar un atributo a la clase del tipo la página nueva definida. Si por ejemplo se le llamó *NewPage*, entonces el atributo deberá definirse de esta forma:

   ```
   protected NewPage newPage;
   ```

2. Modificar el método *addPages* en donde se deberá agregar el siguiente código:

   ```
   newPage = new NewPage();
   addPage(newPage);
   ```

2. Modificación de back-end

Actualmente existen tres clases entre las que se dividen las implementaciones de las distintas características de calidad: *WsAgreementGen*, *WsPolicy* y *WsSecurityPolicy*.

Si el usuario desea agregar nuevas implementaciones de políticas podría agregar los métodos necesarios en la clase *WsPolicy* o podría crear nuevas clases específicas. En caso de que las nuevas políticas sean de seguridad, por ejemplo *Encriptación del header o body*, el usuario podría implementarlas en la clase *WsSecurityPolicy*.

Para implementaciones de características de calidad en base a otros estándares, el usuario deberá crear nuevas clases en el paquete *generators*, que es donde están localizadas las clases mencionadas anteriormente.

Si las nuevas implementaciones implican cambios en el archivo beans.xml en la generación de código con JAX-WS+Spring, entonces se deberá modificar el método *generarBeansXml* de la clase *XmlConfigurationGen*. Si, en cambio, implican cambios en el wsdl, entonces se deberá modificar el método *generarWSDL* de la clase *xmi2wsdl*.

Una vez implementadas las nuevas características de calidad, el usuario deberá modificar la clase *JavaGen*, llamando a los nuevos métodos según si el usuario final eligió las nuevas implementaciones.

Proyecto de grado

Generación automática de Arquitecturas Orientadas a Servicios (SOAs) con SoaML y especificación de QoS

Estado del Arte

Federico Bertolini, Sofía Pérez, María Noel Quiñones

Tutor
Andrea Delgado

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay
Diciembre 2014

Contenido

1. Ingeniería y Desarrollo dirigido por modelos

Antes de introducir los conceptos de ingeniería, desarrollo y arquitectura dirigida por modelos, es conveniente definir qué es un modelo. En [1] se define a un modelo como un conjunto de elementos formales, ordenados con el fin de describir algo para su posterior análisis. Cada modelo aborda un número de objetos de estudio con un grado de abstracción - cuando el grado de abstracción es alto el modelo está más cerca del lenguaje del usuario. El código fuente de un programa puede verse como un modelo del sistema de bajo nivel de abstracción escrito en un lenguaje de programación concreto (por ejemplo, Java o C#).

La ingeniería dirigida por modelos (Model Driven Engineering, MDE) es un paradigma de desarrollo de software donde los modelos son piezas fundamentales en los diferentes procesos de ingeniería de software. Dentro de este enfoque de desarrollo podemos encontrar el desarrollo dirigido por modelos (Model Driven Development, MDD), el testing dirigido por modelos y el despliegue dirigido por modelos. Las tecnologías enfocadas a la ingeniería dirigida por modelos combinan lenguajes de modelado de dominios específicos con motores de transformaciones y generadores. Dichos lenguajes formalizan la estructura de una aplicación, comportamientos y requerimientos dentro de un dominio particular [2]. La ventaja más importante de esto es que los modelos se expresan utilizando conceptos más cercanos al dominio del problema y están mucho menos relacionados a la tecnología utilizada para implementar el sistema [3]. Los generadores y motores de transformación analizan ciertos aspectos de los modelos y generan distintos tipos de artefactos como código fuente, datos de entrada de simulación y representaciones alternativas del modelo. Esto permite mantener la consistencia entre implementaciones de programas e información de análisis asociada a los requerimientos funcionales y no funcionales de los modelos [2].

El desarrollo dirigido por modelos se visualiza en [1] como la noción de que se puede construir un modelo de un sistema que luego se podrá transformar en el sistema real. Los modelos siempre fueron utilizados en la etapa de análisis y diseño de sistemas de software, y son utilizados por los programadores como referencia de lo que se desea construir. El problema de utilizar los modelos como mera documentación es que fácilmente divergen de la realidad y se vuelven obsoletos. Para poder aprovechar realmente el potencial de los modelos, MDE se basa en aprovechar al máximo los beneficios de la automatización, incluyendo la verificación automática de modelos y generación automática de programas [3]. El modelado es apropiado para formalizar conocimiento, y por lo tanto, podemos definir la sintaxis y semántica de un lenguaje a través de la construcción de un modelo del lenguaje de modelado: un metamodelo [1].

El OMG define una arquitectura de cuatro niveles de modelado (ver Figura 1) que consiste en una jerarquía de niveles de modelos, donde cada nivel (salvo el superior) corresponde a una instancia del nivel inmediatamente superior. El nivel M0 contiene los datos del usuario – los objetos de datos que manipula el software. En el nivel M1 se encuentra un modelo de los datos de usuario (modelo de M0); en este nivel se encuentran los modelos de usuario que describen los sistemas. El nivel M2 contiene los metamodelos, que son modelos de los modelos que se encuentran en M1. El metamodelo más conocido es el lenguaje UML. Finalmente, en el nivel M3 se encuentran los meta-metamodelos como MOF, que

son modelos de metamodelos [4]. MOF se describe también a sí mismo por lo cual M3 es el nivel más general.

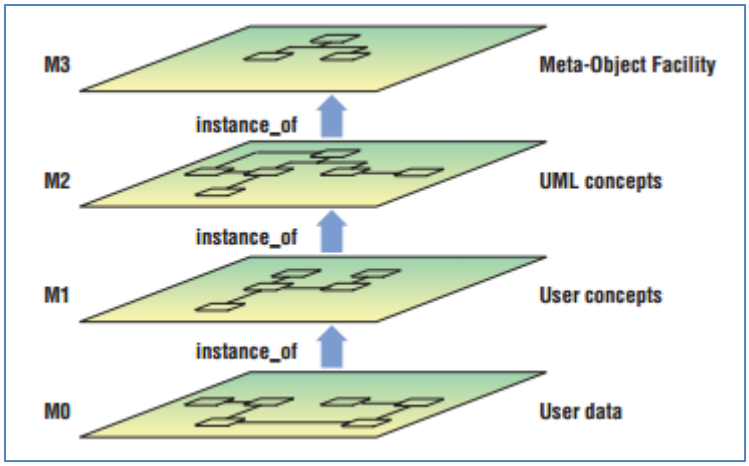

Figura 1. Arquitectura de metamodelos de [4]

2. Arquitectura dirigida por modelos

La Arquitectura Dirigida por Modelos (Model-Driven Architecture, MDA) [5] es un estándar del OMG que implementa el paradigma MDD. MDA provee un enfoque para especificar sistemas independientemente de las plataformas de implementación, especificar estas plataformas, elegir una de éstas y transformar la especificación del sistema a una especificación para la plataforma seleccionada. Los tres objetivos principales de MDA son la portabilidad, interoperabilidad y reusabilidad.

2.1.Tipos de modelos

En [5] se especifican tres tipos de modelos principales: modelo independiente de la computación (CIM), modelo independiente de la plataforma (PIM) y modelo específico a una plataforma (PSM).

El CIM es una representación del sistema desde un punto de vista enfocado en el ambiente del sistema y sus requerimientos, donde la estructura y los detalles de procesamiento del sistema están ocultos o todavía no fueron determinados. Este tipo de modelo es ocasionalmente llamado modelo de dominio y en él se utiliza un vocabulario familiar para los profesionales del dominio en cuestión. Estos modelos juegan un rol importante como nexo entre los expertos en el dominio y sus requerimientos, y los expertos en el diseño y construcción de los artefactos que cubren esos requerimientos. Ejemplos podrían ser modelos de Casos de Uso o de Procesos de Negocio.

El PIM es una representación del sistema desde un punto de vista enfocado en la operación de un sistema, ocultando todos los detalles necesarios de una plataforma en particular. Se pueden utilizar lenguajes de modelado de propósito general o lenguajes específicos al área en el que el sistema va a ser utilizado. Ejemplos serían modelos de clases de diseño UML o modelos de servicios SoaML.

El PSM es una representación del sistema atada a una plataforma en particular. En estos modelos se combinan las especificaciones del PIM con los detalles de cómo el sistema utiliza la plataforma particular.

2.2.Proceso de desarrollo en MDA

El ciclo de vida del desarrollo en MDA contiene la misma secuencia de fases que el ciclo de vida de un proceso de desarrollo tradicional: recabar requerimientos funcionales y no funcionales, análisis, diseño de bajo nivel, codificación, testing y despliegue del sistema [6]. La diferencia entre los dos ciclos de vida son los artefactos de entrada y de salida de cada fase. En la Figura 2 se puede ver una comparación de los ciclos de vida en MDA y en un proceso de desarrollo tradicional.

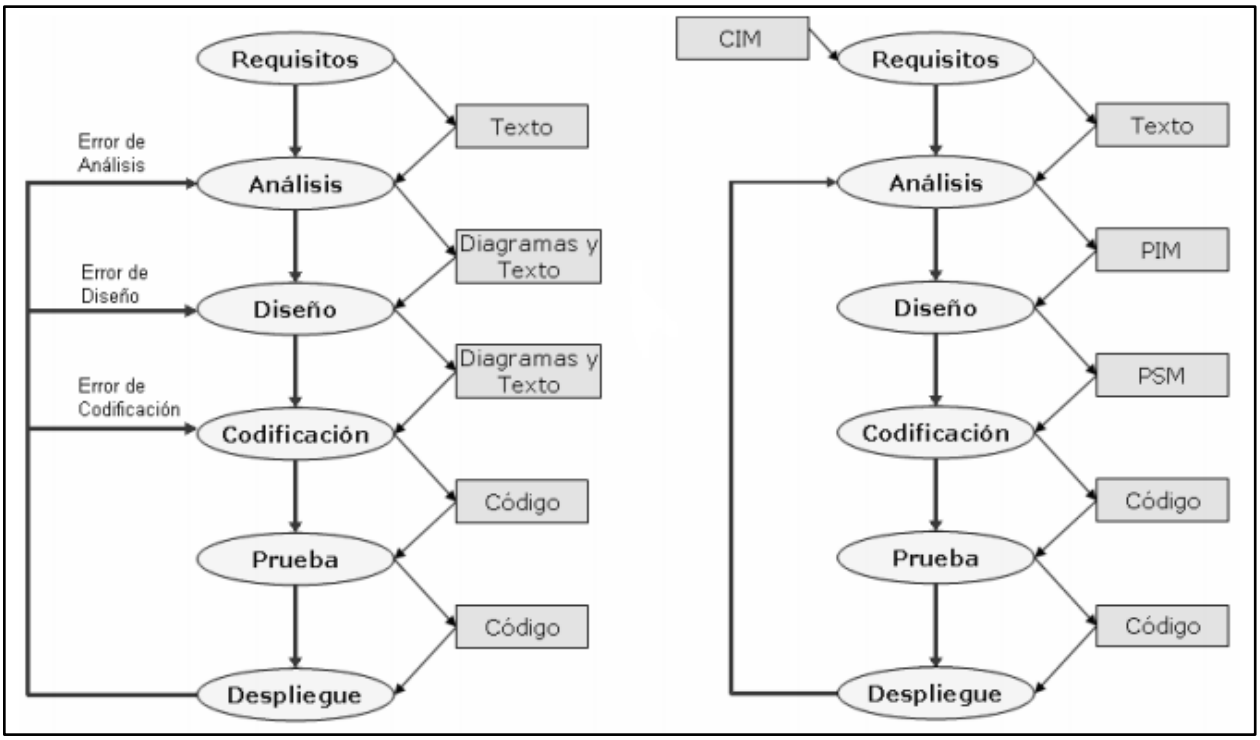

Figura 2. Ciclos de vida en proceso de desarrollo tradicional (izquierda) y en MDA (derecha) de [7]

El modelo independiente de la computación surge en procesos de modelado de negocio e idealmente se concibe antes de recabar los requisitos para un sistema en particular [7]. Los modelos independientes de la plataforma se conciben en la etapa de análisis y son transformados en modelos específicos a plataformas en la etapa de diseño. Un PIM puede ser transformado en varios PSM; uno por cada plataforma destino. La etapa siguiente corresponde a la generación de código a partir de los PSM; si la generación no es completa (se generan esqueletos del sistema), un grupo de desarrolladores necesitará completar el desarrollo del sistema.

2.3.Transformaciones

Uno de los conceptos más importantes en MDA es la transformación de modelos, lo que implica la conversión de un modelo de un sistema a otro modelo del mismo. En MDA existen transformaciones de PIM a PSM, de PSM a código fuente y de PIM a código fuente, siendo los dos primeros tipos de transformaciones los más habituales. En la Figura 3 se muestra la secuencia común de transformación de modelos en MDA.

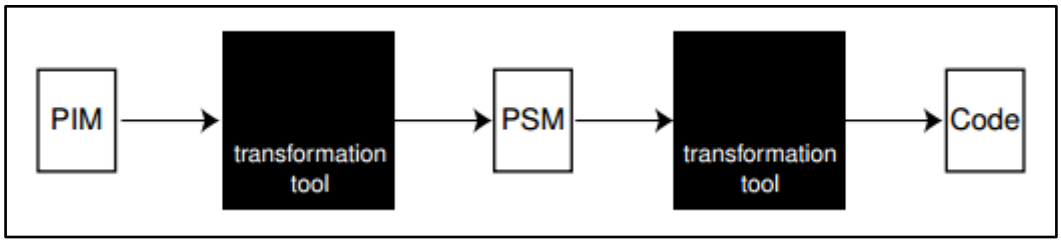

Figura 3. Ejecución de transformaciones en MDA de [6]

Los mapeos (*mapping*) en MDA proveen especificaciones para transformaciones de un PIM a un PSM de determinada plataforma [5]. Por ejemplo, se puede construir un PIM de un sistema y definir mapeos de transformación a modelos PSM con tecnología Java y .NET que finalmente serán transformados a código Java y .NET respectivamente.

Este enfoque permite centrarse en la construcción y mejora continua de modelos PIM, ya que a partir de éstos se genera el código fuente del sistema mediante una sucesión de transformaciones. Una gran ventaja de esto es que si se quiere cambiar la tecnología del sistema alcanza con aplicar una nueva transformación sobre el PIM existente y así generar el PSM para la nueva plataforma. De esta forma se logra la reutilización de modelos y la portabilidad de sistemas.

2.4.Estándares de MDA

El OMG ha adoptado varias tecnologías que juntas habilitan el enfoque dirigido por modelos. Estas tecnologías son UML, MOF, modelos específicos y perfiles UML [5].

2.4.1. Unified Modeling Language (UML)

UML es un lenguaje de modelado estándar que permite visualizar, especificar y documentar sistemas de software. Los modelos utilizados en MDA pueden ser expresados utilizando UML [5]. El objetivo de UML es proveer herramientas de análisis, diseño e implementación de sistemas de software y herramientas de modelado de procesos de negocios a los arquitectos, ingenieros y desarrolladores de software [8].

Una de las principales metas de UML es mejorar el estado de la industria al proveer interoperabilidad de herramientas de modelado de objetos. Para habilitar el intercambio de información de modelos entre distintas herramientas, son necesarias una semántica y sintaxis en común. La sintaxis abstracta de UML está especificada utilizando un modelo UML llamado metamodelo UML, que utiliza conceptos identificados en la especificación de MOF 2 [9] para la construcción de metamodelos. La especificación de UML contiene una explicación detallada de la semántica de cada concepto de modelado; estas semánticas definen de forma independiente a una tecnología en particular, cómo los conceptos UML deben ser implementados por las computadoras. UML también especifica los elementos de notación que pueden ser leídos por humanos, que representan los conceptos de modelado, y las reglas para combinar éstos en tipos de diagramas diferentes correspondientes a diferentes aspectos del sistema modelado [8].

Un modelo UML consiste de tres categorías principales de elementos de modelado, donde cada una de estas categorías puede ser utilizada para realizar declaraciones sobre diferentes tipos de cosas dentro del sistema modelado. La primer categoría son los clasificadores, y éstos describen conjuntos de objetos,

donde un objeto es un individuo con un estado y relaciones a otros objetos. La segunda categoría son los eventos, que describen un conjunto de posibles ocurrencias. Una ocurrencia es algo que sucede y que tiene algunas consecuencias en el sistema. La última categoría de elementos son los comportamientos, que describen un conjunto de posibles ejecuciones. Una ejecución es un conjunto de acciones que pueden generar y responder a eventos, incluyendo el acceso y modificación de estados de objetos.

Los conceptos de modelado UML pueden dividirse semánticamente en semántica de estructura y semántica de comportamiento, donde la primera provee los cimientos para la segunda. Esto refleja la concepción de semánticas de comportamiento en términos de cambios en el estado del sistema especificados a través de modelado estructural. La Figura 4 muestra una delineación más detallada de las áreas semánticas de UML dentro de estas categorías, y la noción de capas de estas áreas.

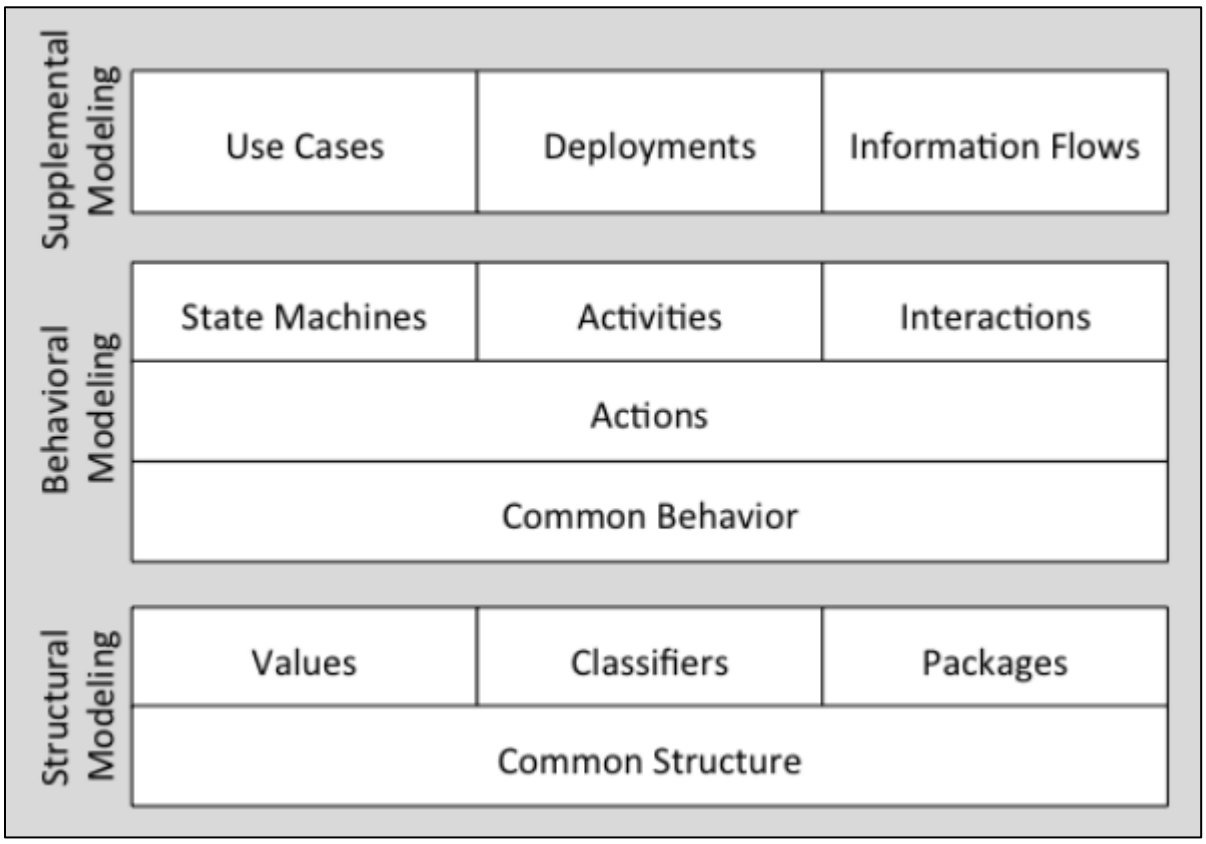

Figura 4. Áreas semánticas de UML de [8]

2.4.2. Meta Object Facility (MOF)

MOF [9] es un framework de gestión de metadatos y un conjunto asociado de servicios de metadatos, que posibilitan el desarrollo e interoperabilidad de sistemas dirigidos por modelos y metadatos. Este estándar es la base para la definición de metamodelos en la familia de lenguajes MDA del OMG.

MOF introdujo los conceptos de metamodelos formales y de modelos de metadata PIM, así como también los mapeos de PIMs a PSMs. La alineación entre MOF y UML se completó con MOF 2.4 y UML 2.4, al compartir el mismo metamodelo para la definición de UML y MOF, usando restricciones OCL para

definir el subconjunto del metamodelo relevante para MOF. Esta unificación en los conceptos de modelado de UML y MOF trae aparejados tres beneficios principales:

- Reglas más simples para modelar metadatos (solamente se necesita entender un subconjunto del modelado de clases UML sin anotaciones o conceptos de modelado adicionales).
- Varios mapeos existentes de MOF ahora aplican también a una amplia gama de modelos UML, incluyendo perfiles.
- Amplia gama de soporte de herramientas para metamodelado (cualquier herramienta de modelado UML puede usarse para modelar metadatos).

2.4.3. Perfiles UML

Los perfiles UML permiten la extensión de metaclases UML para adaptarlos a diferentes propósitos; esto incluye la habilidad de adaptar el metamodelo UML para plataformas o dominios diferentes. Los perfiles no permiten la creación de nuevos metamodelos; su intención es dar un mecanismo directo para adaptar un metamodelo existente con elementos que son específicos a un dominio, plataforma o método específico.

En [8] se listan las siguientes razones por las cuales se debería extender UML:

- Dar una terminología que se adapte a una plataforma o dominio particular.
- Dar una sintaxis para elementos que no tienen una notación.
- Dar una notación diferente a símbolos que ya existen.
- Agregar semántica adicional a UML o metaclases específicas.
- Agregar tipos que no existen en UML.
- Agregar restricciones sobre la forma en que los elementos UML son utilizados.
- Agregar información que pueda ser utilizada en las transformaciones de un modelo a otro.

El elemento principal de extensión es el *Stereotype* (estereotipo), y éste define una extensión para una o más metaclases (*Metaclass*), y habilita la utilización de terminología o notación específica adicionalmente a las utilizadas en las metaclases extendidas. Igual que una clase (*Class*), un estereotipo puede tener propiedades (*Properties*).

Los perfiles UML son paquetes (*Profile* es una especialización de *Package*) compuestos de estereotipos, y son aplicados a otros paquetes; esto significa que un perfil puede ser aplicado a otro perfil. Es posible aplicar múltiples perfiles a un paquete, aunque esto podría hacer que el paquete sea inválido si los perfiles tienen restricciones en conflicto. Cuando se aplica un perfil, se aplican recursivamente todos sus perfiles anidados e importados. Los estereotipos públicos de un perfil pueden ser aplicados a elementos de modelo en los paquetes en que el perfil fue aplicado [8].

2.4.4. Query/View/Transformation (QVT)

QVT es un conjunto de lenguajes especificados por OMG para la transformación de modelos. La especificación de QVT tiene una naturaleza declarativa/imperativa híbrida, con la parte declarativa dividida en una arquitectura de dos niveles: *Relations* (Relaciones) y *Core* (Núcleo). Esta arquitectura declarativa forma un *framework* para la semántica de ejecución de la parte imperativa [10].

Relations es una especificación declarativa de las relaciones entre modelos MOF. Este lenguaje soporta la búsqueda de patrones de objetos complejos, y crea implícitamente clases "huella" (*trace classes*) y sus instancias para registrar qué ocurre durante la ejecución de una transformación. En este lenguaje, una transformación entre modelos candidatos es especificada como un conjunto de relaciones que deben suceder para que la transformación sea satisfactoria [10].

El Core es un modelo/lenguaje pequeño que solamente soporta la búsqueda de patrones en un conjunto plano de variables por medio de la evaluación de condiciones sobre esas variables contra un conjunto de modelos. Es igual de poderoso que el lenguaje Relations, y gracias a su simpleza, su semántica puede ser definida de forma más simple, aunque las descripciones de las transformaciones descriptas por el Core son más extensas. Adicionalmente, los modelos "huella" deben ser definidos explícitamente, y no son deducidos de la descripción de la transformación, como es el caso con Relaciones [10].

El lenguaje Operational Mappings está especificado como una forma estándar de proveer implementaciones imperativas, que crea los modelos "huella" de la misma forma que el lenguaje Relations. Este lenguaje puede ser utilizado para implementar una o más relaciones cuando es difícil proveer una especificación puramente declarativa de cómo una relación debe ser poblada. Una transformación operacional representa la definición de una transformación unidireccional expresada de forma imperativa. Define una firma indicando los modelos involucrados en la transformación y define una operación para su ejecución. De la misma forma que una clase, una transformación operacional es una entidad instanciable con propiedades y operaciones [10].

La semántica de los lenguajes Core y Relations permiten:

- Transformaciones unidireccionales.
- Transformaciones bidireccionales.
- Verificaciones de que múltiples modelos están relacionados de una manera específica.
- Establecer relaciones entre modelos pre-existentes.
- Actualizaciones incrementales cuando un modelo relacionado es modificado luego de una ejecución inicial.

El enfoque de Operational Mappings no permite transformaciones bidireccionales, únicamente permite especificar transformaciones en una sola dirección. El resto de las capacidades listadas anteriormentes están disponibles en ejecuciones híbridas e imperativas [10].

2.4.5. MOF Model to Text

El estándar MOF Model to Text (mof2text) aborda cómo traducir un modelo a varios artefactos de texto como código, especificaciones de despliegue, reportes, documentos, etc. Esencialmente, el estándar aborda cómo traducir un modelo a una representación de texto linear. Una manera intuitiva de realizar este requerimiento es con un enfoque basado en plantillas donde el texto que va a ser generado a partir de modelos es especificado como un conjunto de plantillas de texto que son parametrizadas con elementos de modelo [11].

Las plantillas están compuestas por expresiones especificadas sobre entidades de metamodelo, donde el mecanismo principal para seleccionar y extraer valores del modelo son las consultas. Estos valores son convertidos en fragmentos de texto usando un lenguaje de expresión y una biblioteca de manipulación de *strings*. Se puede realizar composiciones de plantillas para crear transformaciones complejas [11].

Las especificaciones de transformaciones extensas pueden ser estructuradas en Módulos. Un Módulo consiste de un conjunto de Plantillas y Consultas, y tiene una parte pública y una privada. La parte pública expone las Plantillas y Consultas que pueden ser invocadas desde otros módulos. Una transformación (plantilla) puede ser iniciada mediante la invocación de una plantilla pública con los parámetros correctos. No hay una noción explícita de una plantilla principal [11].

Una Plantilla puede sobrescribir una o más Plantillas. Un Módulo puede extender a otro Módulo mediante herencia (solamente se puede realizar herencia simple); el Módulo que especializa hereda todas las plantillas de su superior y puede acceder o sobrescribir todas las plantillas públicas y protegidas [11].

2.4.6. XML Metadata Interchange (XMI)

El estándar XMI [12] define los siguientes aspectos involucrados en la descripción de objetos en XML:

- La representación de objetos en términos de elementos y atributos XML.
- El mecanismo estándar de vincular objetos dentro del mismo archivo o entre distintos archivos.
- La validación de documentos XMI usando XML Schemas.
- Identidad de objetos, que permite que los objetos sean referenciados desde otros objetos en términos de identificadores.

XMI describe soluciones a los puntos listados anteriormente especificando reglas de producción para crear documentos y esquemas XML que comparten objetos de forma consistente [12].

Un esquema XML provee una vía para que un procesador XML pueda validar la sintaxis y alguna de las semánticas de un documento XML. El estándar XMI provee reglas para generar esquemas de cualquier modelo basado en MOF transmisible por XMI. El uso de esquemas es opcional; un documento XML no necesita referenciar un esquema, incluso cuando existe uno. El documento resultante puede ser procesado más rápido, al costo de pérdida de confianza en la calidad del documento [12].

El proceso de producción de documentos XML está definido en XMI como un conjunto de reglas de producción. Cuando estas reglas son aplicadas a un modelo el resultado es un documento XML. El inverso de las reglas puede ser aplicado a un documento XML para reconstruir el modelo. En ambos casos, las reglas son aplicadas implícitamente en el contexto del metamodelo específico para la metadata que está siendo intercambiada [12].

3. Computación y Arquitectura Orientada a Servicios

El término “orientación a servicios” implica la división de los diferentes problemas en problemas más pequeños a resolver. La lógica requerida para resolver un problema grande puede ser construida de mejor forma y más fácilmente si es dividida en partes más pequeñas relacionadas entre sí [13]. En otras palabras, la orientación a servicios implica encapsular ciertas funcionalidades de negocio en servicios autónomos que podrán ser compartidos para que otros servicios puedan utilizarlos y combinarlos para crear otros servicios.

La Computación Orientada a Servicios es definida en [14] como un paradigma que utiliza servicios para soportar el desarrollo rápido de aplicaciones evolutivas, interoperables y distribuidas masivamente a bajo costo. Estos servicios son entidades autónomas e independientes de la plataforma, que pueden ser descriptas, publicadas y descubiertas para que otros servicios las utilicen. SOC impulsa la composición de aplicaciones distribuidas mediante el uso de servicios con un bajo acoplamiento.

La Arquitectura Orientada a Servicios, inspirada en el paradigma SOC, es definida en [13] como un modelo en el que la lógica del sistema es dividida en unidades lógicas más pequeñas (servicios) que pueden ser distribuidas de forma independiente.

En [14] se define SOA como una forma de diseñar un sistema de software que provea servicios a aplicaciones de usuarios finales u otros servicios distribuidos en una red, a través de la publicación de interfaces fáciles de encontrar.

SOA no está atada a ninguna tecnología [15], aunque actualmente la estrategia más común de implementación es el uso de *web services*, mediante la utilización de estándares como HTTP, SOAP, WSDL y UDDI [16].

Como se mencionaba anteriormente, un sistema basado en SOA está compuesto principalmente por servicios. El propósito fundamental de un servicio es el de representar una unidad de trabajo reutilizable. Cada servicio es una porción de funcionalidad expuesta con tres características esenciales: su autonomía, la independencia de la plataforma y la capacidad de ser localizado, invocado y combinado con otros servicios de forma dinámica [15]. Asimismo, el sistema está compuesto por consumidores de servicios, que son clientes de las funcionalidades provistas por los servicios (por ejemplo, aplicaciones de usuario final, sistemas u otros servicios) y por infraestructura de SOA, que conecta consumidores de servicios con los servicios [16].

4. Modelado de servicios con SoaML

SoaML [17] es un lenguaje de modelado de servicios, el cual ofrece un *framework* completo e intuitivo para representar artefactos SOA en UML, como ser participantes, puertos, servicios, contratos, etc. SoaML está implementada como un perfil UML, lo que permite su utilización en cualquier herramienta de modelado UML.

4.1. Elementos de SoaML

A continuación, se definen los principales elementos del metamodelo de SoaML según [17].

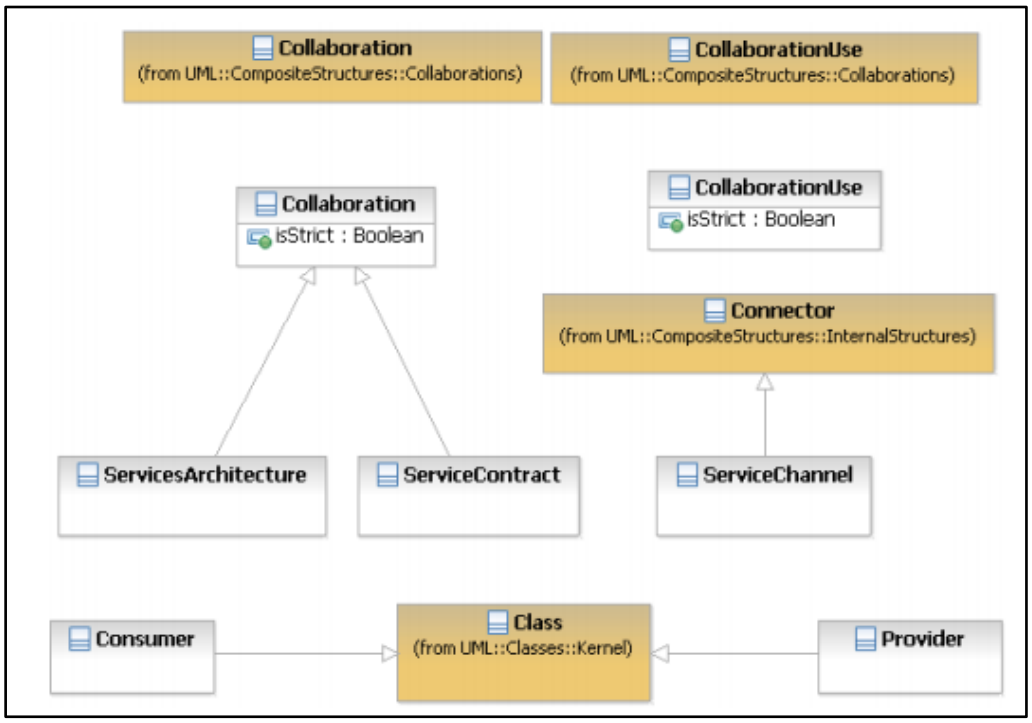

Figura 5. Metamodelo de Contratos de Servicio y Arquitecturas de Servicio de [17]

4.1.1. Collaboration

Una Collaboration representa un patrón de interacción entre roles. Esta interacción puede ser definida informalmente o puede representar acuerdos formales o requerimientos que deben ser cumplidos estrictamente. Collaboration extiende a Collaboration de UML, con el agregado de la propiedad *isStrict*, la cual indica si ésta representa un patrón estricto de interacción.

4.1.2. ServicesArchitecture

Una ServicesArchitecture es la visión de alto nivel de una SOA. Describe de qué manera un conjunto de participantes interactúan entre sí por un propósito, proporcionando y consumiendo servicios expresados como contratos de servicios.

4.1.3. ServiceContract

Un ServiceContract es la especificación del acuerdo entre proveedores y consumidores de un servicio. Ésta incluye qué información, productos, valores y obligaciones serán intercambiados entre ellos.

4.1.4. ServiceChannel

Un ServiceChannel provee una vía de comunicación entre solicitudes de consumidores y servicios de proveedores.

4.1.5. Consumer

Un Consumer modela la interfaz provista por el consumidor de un servicio. El consumidor del servicio recibe los resultados de la interacción del servicio; éste es generalmente el que inicia dicha interacción. Las interfaces de los consumidores son utilizadas como tipo de un ServiceContract y están atadas a los términos y condiciones de dicho contrato de servicio.

El Consumer fue previsto para usar como tipo de un puerto de un participante que usa un servicio.

4.1.6. Provider

Un Provider modela la interfaz provista por el proveedor de un servicio. El proveedor del servicio entrega los resultados de la interacción del servicio; éste es generalmente el que da respuestas ante dicha interacción. Las interfaces de los proveedores son utilizadas como tipo de un ServiceContract y están atadas a los términos y condiciones de dicho contrato de servicio.

El Provider fue previsto para usar como tipo de un puerto de un participante que provee un servicio.

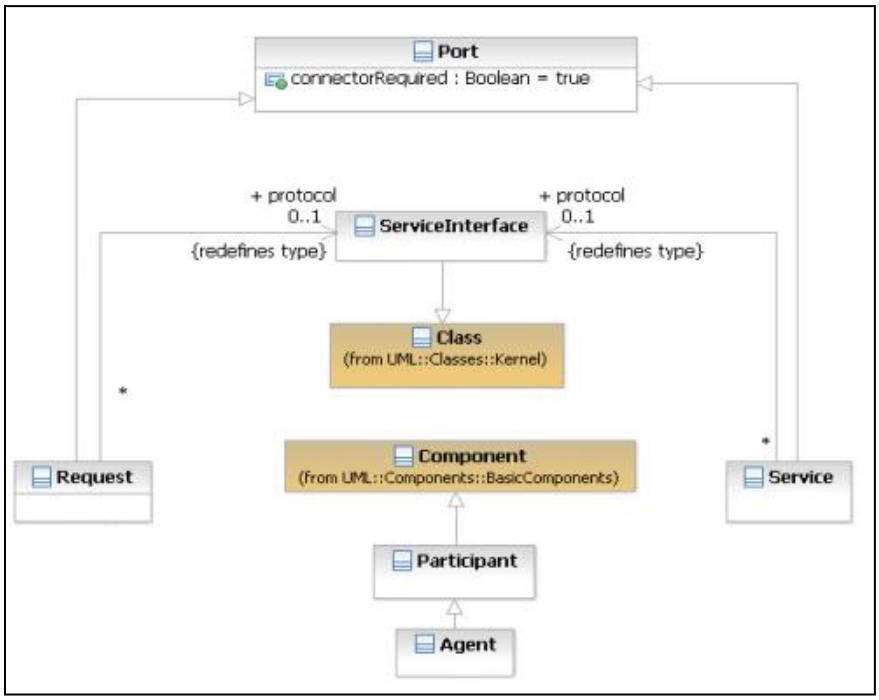

Figura 6. Metamodelo de Interfaces de Servicios y Participantes de [17]

4.1.7. Participant

Un Participant es una persona, organización o sistema que provee y/o consume servicios a través de sus puertos de servicios. Éste implementa cada una de las operaciones de servicios provistas y dicha implementación debe ser consistente con las operaciones, protocolos y restricciones especificadas por la ServiceInterface.

4.1.8. Agent

Un Agent es una especie de entidad autónoma que puede adaptarse a su entorno e interactuar con éste. Describe un conjunto de instancias de agentes que tienen características, restricciones y semántica en común. Los agentes en SoaML son también participantes, que proveen y consumen servicios.

4.1.9. Port

Los participantes proveen o consumen servicios a través de puertos. Un puerto es la parte de un participante en la que se produce la interacción para consumir o proveer un servicio. Un puerto en el que se ofrece un servicio se llama Puerto de Servicio, mientras que un puerto en el que se consume un servicio se llama Puerto de Solicitud. Este elemento extiende a Port de UML.

4.1.10. ServiceInterface

Una ServiceInterface define la interfaz y responsabilidades de un participante de proveer o consumir un servicio. Es usada como tipo de un puerto de Servicio o de Solicitud.

4.1.11. Request

Request es una extensión de Port, la cual especifica una característica de un Participant que representa un servicio que el Participant necesita y consume de otros participantes. El servicio es definido por una ServiceInterface.

4.1.12. Service

Service es una extensión de Port, la cual especifica una característica de un Participant que representa un servicio que el Participant provee y ofrece para ser consumido por otros participantes. El servicio es definido por una ServiceInterface.

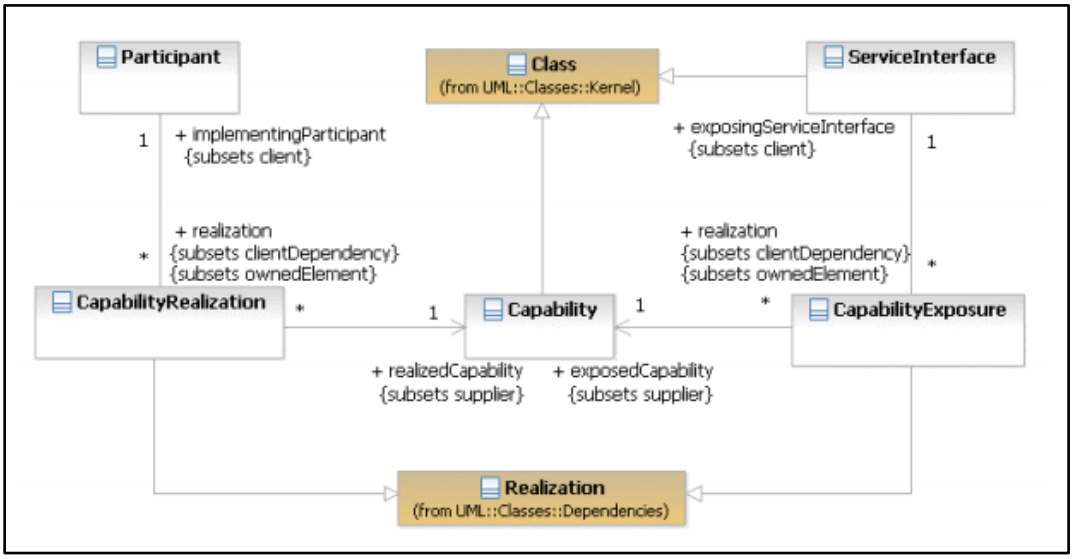

Figura 7. Metamodelo de Funcionalidades de [17]

4.1.13. Capability

Una Capability especifica un conjunto de funcionalidades que un servicio puede ofrecer. Se utiliza para identificar los servicios necesarios y para organizarlos en catálogos.

4.2.Diagramas de SoaML

A continuación se explicarán los distintos diagramas que se pueden definir en SoaML según [17] y también cómo se modelan los distintos elementos del lenguaje para definir cierta arquitectura orientada a servicios.

4.2.1. Diagrama de Arquitectura de Servicios

El diagrama de Arquitectura de Servicios (Service Architecture) es una vista de alto nivel de abstracción en donde se define un conjunto de servicios y se muestra cómo los distintos participantes cumplen diferentes roles interactuando entre sí para alcanzar ciertos objetivos.

En la Figura 8 se puede ver un ejemplo de diagrama de Services Architecture y cómo se relacionan los elementos entre sí.

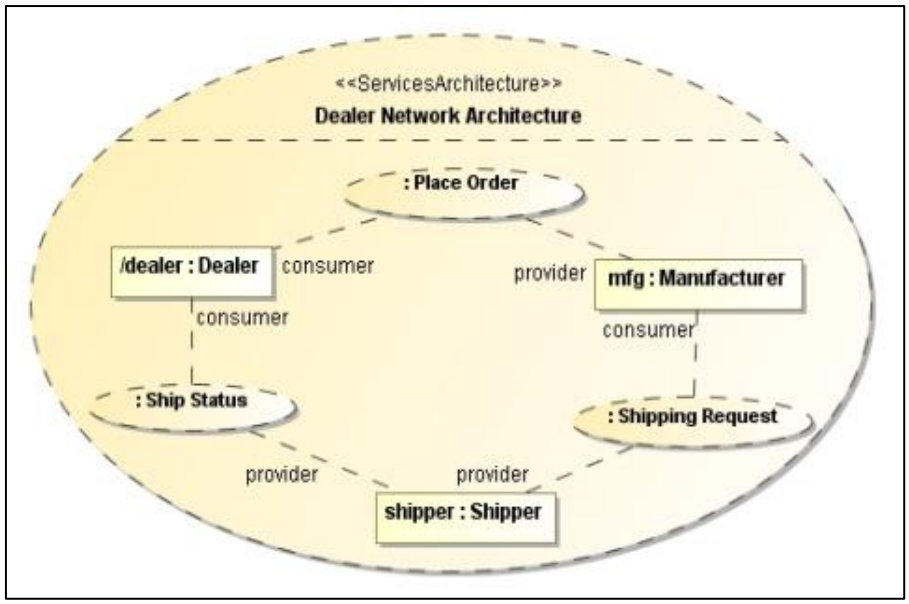

Figura 8. Ejemplo de Arquitectura de Servicios de [17]

Los participantes (*dealer*, *mfg* y *shipper*) son los clasificadores definidos tanto por los roles que desempeñan en las arquitecturas de servicios (el rol del participante) como por los requisitos de "contrato" de entidades que cumplen esos roles. Cada tipo de participante puede "jugar un rol" en cualquier cantidad de Services Architecture, así como cumplir con los requisitos de cada uno. Los requerimientos son satisfechos por los participantes que tienen puertos de servicios compatibles con los servicios que deben proveer y consumir. Los servicios definidos son *Place Order*, *Ship Status* y *Shipping Request*.

4.2.2. Diagrama de Contrato de servicios

El diagrama de Contrato de Servicios (Service Contract) define los términos, las condiciones y las interfaces con las cuales los participantes que interactúan deben estar de acuerdo para que el servicio sea ejecutado. La base del Service Contract es también una colaboración UML que se centra en las interacciones que intervienen para proporcionar un servicio.

Un Participant juega un rol como el proveedor o consumidor de los servicios especificados por el Service Contract. Cada Rol es definido como una Interface o ServiceInterface. En el Service Contract se definen las relaciones entre un conjunto de roles definidos por las Interfaces y/o ServiceInterfaces.

En la Figura 9 se puede ver un ejemplo de diagrama de Service Contract, donde se define el contrato *Purchasing Service* compuesto por otros dos contratos *Place Order* y *Returns Service*. Además se muestran los roles definidos para el contrato (*buyer* y *seller*) que a su vez son consumidores y proveedores de los servicios definidos en los contratos incluidos dentro del mismo.

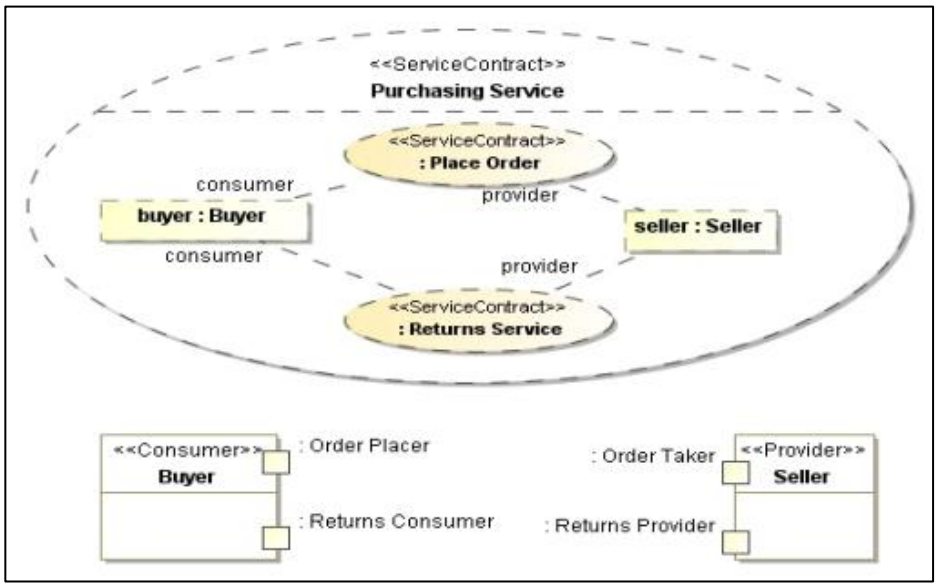

Figura 9. Ejemplo de Contrato de Servicio de [17]

4.2.3. Diagrama de Participantes como Clase

El diagrama de Participantes como Clase se utiliza para definir los distintos participantes y los puertos desde los cuales proveen o consumen servicios. Un Participant representa una parte o componente que provee y/o consume servicios. En el ámbito de negocios un Participant puede ser una persona, organización o sistema.

Un Participant tiene puertos los cuales utilizan el estereotipo Service y Request, y son el punto de interacción donde los servicios son ofrecidos o consumidos respectivamente. Un Participant que ofrece un servicio tiene un puerto Service cuyo tipo es una ServiceInterface de tipo Provider. Este participante implementa dicha interfaz de servicio. En cambio, un Participant que consume un servicio tiene un puerto Request cuyo tipo es una ServiceInterface de tipo Consumer. Este participante utiliza la interfaz de servicio provista por otro participante.

En la Figura 10 se puede ver un ejemplo de diagrama de Participantes como Clase, en donde el Participant *Dealer* tiene un puerto *Request* de tipo interfaz *Shipment status*. Esto quiere decir que el participante utiliza dicha interfaz. Por otro lado, el Participant *Shipper* tiene un puerto *Service* también de tipo interfaz *Shipment status*, lo que quiere decir que el participante provee dicha interfaz.

Figura 10. Ejemplo de diagrama de Participantes como Clase de [17]

4.2.4. Diagrama de Participantes como Componente

El diagrama de Participantes como componente sirve para especificar los participantes y las operaciones que involucran los servicios. Este diagrama también se puede utilizar para mostrar la reutilización de servicios existentes que se tienen que adaptar al contexto en el cual se quiere que funcione. La adaptación se hace definiendo un adaptador (Adapter), el cual se encargará de realizar cualquier cambio requerido en los datos, protocolo o proceso. En particular, se utiliza para unificar servicios *bottom-up* y *top-down*, en donde el *bottom-up* está acoplado a la tecnología que lo soporta y el *top-down* es más general y menos acoplado. El Adapter define la conexión entre los dos. En la Figura 11 se puede ver un ejemplo de diagrama de Participantes como Componente.

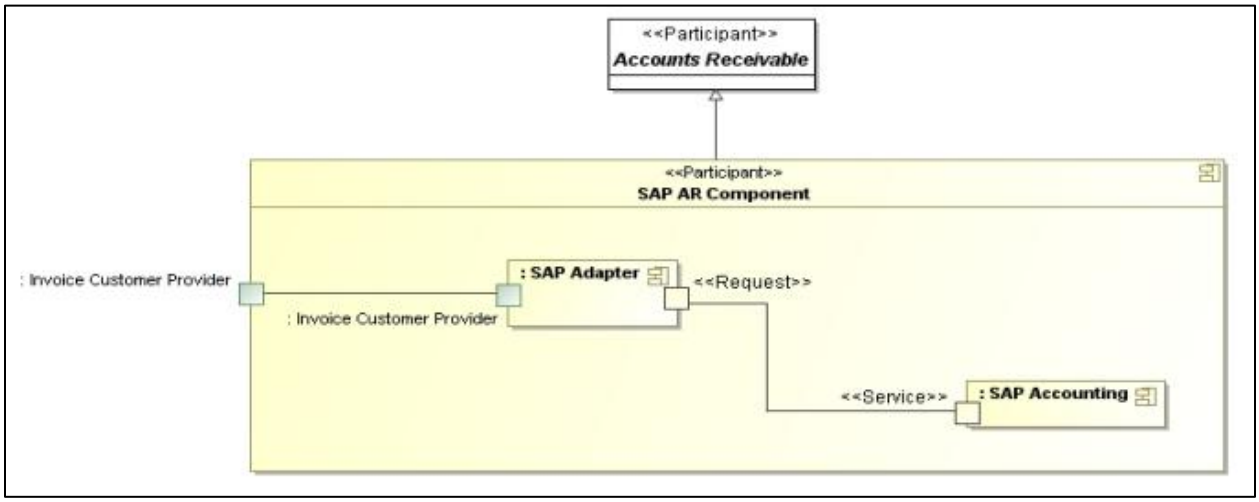

Figura 11. Ejemplo de diagrama de Participantes como Componente de [17]

4.2.5. Diagrama de Interfaces

El diagrama de interfaces permite modelar las distintas interfaces y las interfaces de servicios con las operaciones que proveen para realizar los servicios. También permite modelar las relaciones de dependencia entre las interfaces existentes. En la Figura 12 se puede ver un ejemplo de diagrama de Interfaces.

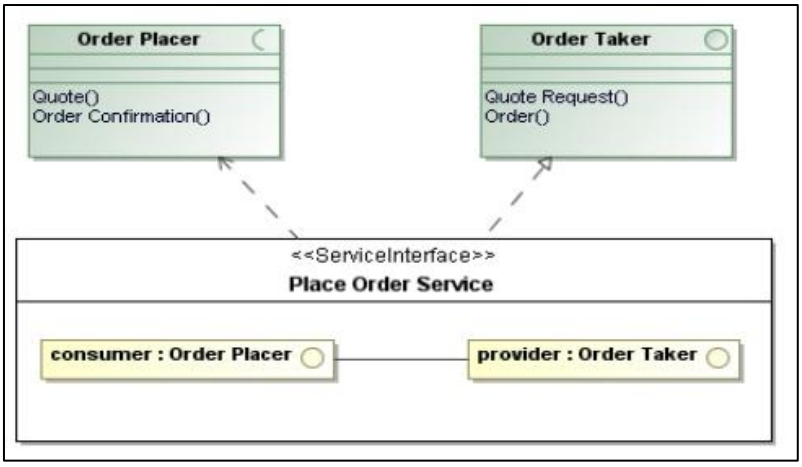

Figura 12. Ejemplo de diagrama de Interfaces de [17]

4.2.6. Diagrama de Capacidades

Un diagrama de Capacidades (Capability) es útil para expresar una arquitectura de servicios en términos de las capacidades lógicas de los servicios sin necesidad de conocer los participantes. Aunque los consumidores de servicios no deben preocuparse por cómo se implementa un servicio, es importante ser capaz de especificar el comportamiento de un servicio o la capacidad que realiza o implementa un ServiceInterface.

Las capacidades pueden ser utilizadas por sí mismas o en conjunto con los participantes para representar funcionalidades generales o habilidades que un participante debe tener. También se pueden utilizar una o más capacidades para especificar el comportamiento y la estructura necesaria para soportar una ServiceInterface. A su vez una ServiceInterface puede exponer capacidades a través de la dependencia *Expose*.

En la Figura 13 se puede ver un ejemplo de diagrama de Capacidades.

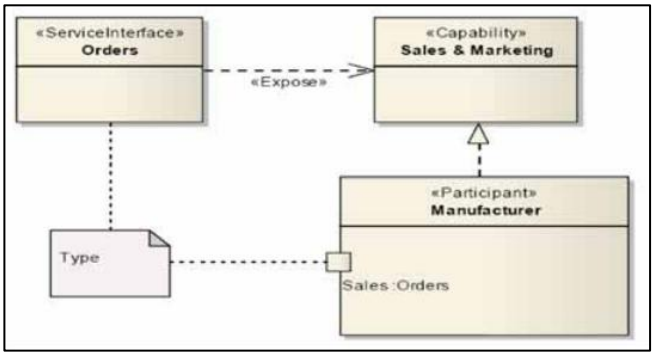

Figura 13. Ejemplo de diagrama de Capacidades de [17]

4.2.7. Diagrama de Mensajes

El diagrama de Mensajes tiene elementos que sirven para modelar los distintos tipos de datos. Se pueden definir *Message Type*, *Attachment* y *Property*. El tipo *MessageType* se utiliza para especificar la información intercambiada entre participantes que consumen y proveen un servicio, y puede ser definido como de tipo *Class*, *DataType* o *Enumeration.* El tipo *Attachment* es un componente de un

mensaje y está adjunto a él. El tipo *Property* es una característica estructural y puede ser designado como un identificador de propiedad, una propiedad que se puede utilizar para distinguir o identificar instancias del clasificador que las contiene. En la Figura 14 se puede ver un ejemplo de diagrama de Mensajes.

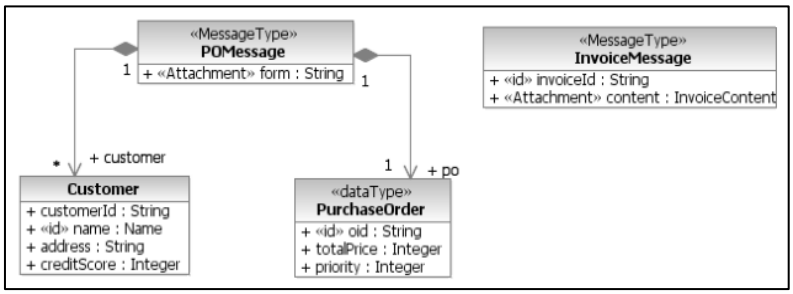

Figura 14. Ejemplo de diagrama de Mensajes de [17]

5. Calidad de servicios (QoS)

La calidad de servicio (*Quality of Service, QoS*) se puede definir como un conjunto de características perceptibles expresado en un lenguaje amigable con parámetros cuantificables que pueden ser subjetivos u objetivos [18]. QoS comprende atributos no funcionales importantes tales como las medidas de rendimiento (por ejemplo, tiempo de respuesta), atributos de seguridad, integridad transaccional, fiabilidad, escalabilidad y disponibilidad [14].

También el término calidad de servicio se utiliza a menudo para hacer referencia a la colección de requerimientos de calidad de un servicio. Podría pasar que diferentes servicios provean la misma funcionalidad para los consumidores, por lo tanto las especificaciones de QoS para estos servicios son esenciales a la hora de determinar el servicio más adecuado [19].

5.1.Atributos de calidad en SOAs

El diseño de la arquitectura de un sistema se ve directamente influenciado por los requerimientos de calidad y otros aspectos no funcionales, los cuales son importantes satisfacer para alcanzar los objetivos de negocios de la organización [20].

En [21] se especifican factores de calidad de servicios web, con la definición y explicación de los sub-factores. Los factores de calidad descritos en esta especificación son: valor de negocio, medición de nivel de servicio, interoperabilidad, procesamiento de negocio, gestionabilidad y seguridad. Estos factores pueden categorizarse en dos grupos: grupo de calidad de negocio y grupo de calidad de sistema. El primero solamente está compuesto por el factor calidad de nivel de negocio; el segundo está compuesto por una parte de calidad variante y otra de calidad invariante. La parte de calidad variante incluye factores de calidad cuyos valores pueden cambiar dinámicamente en tiempo de ejecución mientras el servicio se está usando. De forma contraria, la parte invariante refiere a los factores de calidad cuyos valores son determinados cuando el desarrollo del servicio es completado. En la Figura 15 se muestra la estructura y categorización de los factores de calidad.

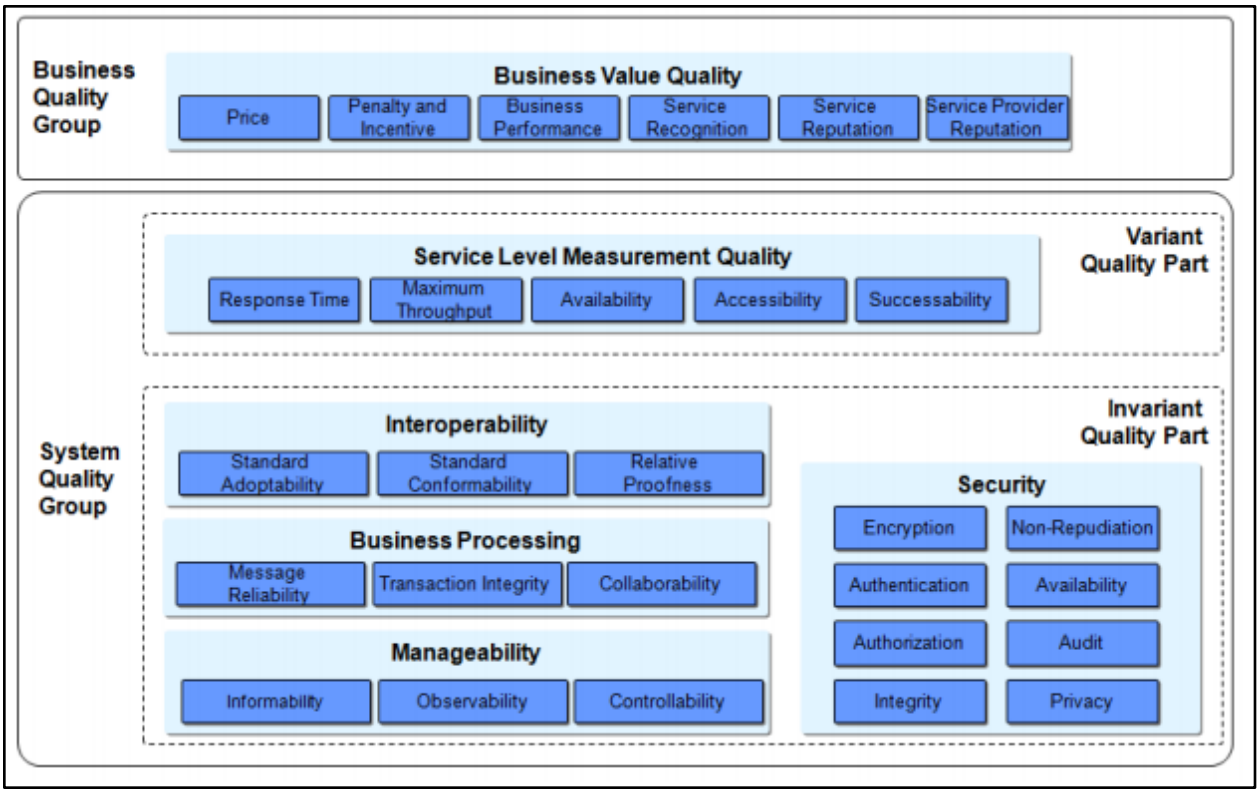

Figura 15. Factores y sub-factores de calidad de [21]

A continuación describiremos algunos de los factores y subfactores de calidad especificados en [21], y algunos atributos de calidad que según [20] hay que tener en cuenta a la hora de diseñar e implementar una arquitectura SOA.

5.1.1. Interoperabilidad

La interoperabilidad es la capacidad de una colección de entidades de compartir información específica y operar con dicha información bajo acuerdos semánticos operacionales [20].

En [21] se destaca la importancia de la interoperabilidad en los servicios web, dado que el servidor y el cliente pueden estar desplegados en plataformas distintas. La calidad de interoperabilidad es definida como un factor de calidad que evalúa si un sistema compuesto por servicios web está implementado utilizando los estándares correctos y de forma afín a la especificación.

5.1.2. Fiabilidad

La fiabilidad es la capacidad que tiene un sistema de mantenerse operativo en el tiempo. Cuando se habla de fiabilidad de servicios, se refiere a que los servicios no fallen o en caso de hacerlo reportan las fallas al usuario del servicio [20].

Generalmente, los servicios son accesibles dentro de una red con canales de comunicación confiables. Las conexiones se caen y los mensajes no son entregados o son entregados más de una vez o en la secuencia incorrecta. Aunque las técnicas para asegurar la entrega confiable de mensajes son razonablemente bien entendidas y están disponibles en algunos productos middleware de mensajería,

la fiabilidad en los mensajes sigue siendo un problema. Si la fiabilidad es manejada por los desarrolladores de servicios que están incorporando técnicas de fiabilidad directamente en los servicios y aplicaciones, no existen garantías de que van a tomar decisiones consistentes en los enfoques a adoptar. Esto puede resultar en que no se pueda garantizar la fiabilidad de mensajes de extremo a extremo. Incluso la utilización de productos middleware de diferentes vendedores puede excluir el intercambio fiable de mensajes entre aplicaciones y servicios. Existen dos especificaciones que abordan estos problemas y que definen protocolos que aseguran el intercambio fiable e interoperable de mensajes: WS-Reliability desarrollado por OASIS Consortium y WS-ReliableMessaging [20].

5.1.3. Disponibilidad

La disponibilidad es el grado en que un sistema o componente está operativo y accesible cuando es requerido para su uso [20]. Asimismo, la disponibilidad es uno de los subfactores de Seguridad en [21], ya que una de las formas de ataque a servicios web publicados en Internet es la denegación de servicio, que apunta a dejar al servicio inactivo.

5.1.4. Usabilidad

La usabilidad es una medida de calidad de la experiencia del usuario en la interacción con la información o servicios. Para mejorar la usabilidad en un sistema con una SOA, los proveedores de servicios deben considerar la granularidad en los datos, el desarrollo de servicios para soportar la usabilidad (por ejemplo, un servicio para cancelar un pedido o deshacer el último pedido), y funcionalidades que permitan el restablecimiento del contexto de un servicio en caso de pérdida de conexión (por ejemplo, en un carrito de compras online) [20].

5.1.5. Seguridad

La seguridad en los sistemas de software es asociada con cuatro principios [20]:

1. Confidencialidad: Solamente los sujetos autorizados pueden acceder a la información o servicio.
2. Autenticidad: Se puede confiar en que el autor o emisor de un mensaje es el responsable de la información del mensaje
3. Integridad: La información no está corrupta.
4. Disponibilidad: La información o servicio están disponibles de forma apropiada.

En una SOA, a menudo los mensajes contienen datos en algún formato de texto (XML por ejemplo) y esos datos podrían contener información personal. Una buena práctica es la encriptación de mensajes para preservar la privacidad de los datos [20]. Además, los servicios pueden tener un acceso restringido según el usuario del servicio por lo que se debe tener un mecanismo de autorización que permita configurar y otorgar permisos para un usuario específico, grupo de usuarios o rol [20].

5.1.6. Desempeño (Performance)

El desempeño está asociado al tiempo de respuesta (cuánto tarda el procesamiento de un pedido), rendimiento (cuántos pedidos pueden ser procesados por unidad de tiempo) y capacidad (cuánta demanda puede soportar el sistema cumpliendo con los requerimientos de tiempo de respuesta y rendimiento) [20].

5.1.7. Escalabilidad

La escalabilidad es la habilidad que tiene una SOA de funcionar correctamente cuando el sistema cambia de tamaño o en volumen con el fin de satisfacer las necesidades de los usuarios, sin que se vean degradados otros atributos de calidad [20].

5.1.8. Extensibilidad

La extensibilidad es la facilidad con la que las funcionalidades de los servicios pueden extenderse sin afectar otros servicios. El ambiente de negocio donde operan los sistemas de software está en continuo cambio y evolucionando, por lo que la extensibilidad en en SOA es muy importante. Estos cambios pueden darse en el sistema de software, en los usuarios de servicios, en los proveedores de servicios y en los mensajes que intercambian entre ellos. Extender en SOA significa hacer cambios que incluyen la extensión de la arquitectura, agregando servicios adicionales, o extendiendo servicios existentes [20].

5.1.9. Adaptabilidad

La adaptabilidad es la facilidad con la que un sistema puede cambiar para adaptarse a modificaciones en los requerimientos. Para alcanzar la adaptabilidad, los servicios deberán ser gestionados y monitoreados adecuadamente como una única solución cohesiva, como también debe ser gestionada la interacción entre los servicios y la aplicación subyacente [20].

5.1.10. Capacidad de testing

La capacidad de testing se define en [20] como el grado en que un sistema o servicio facilita el establecimiento de criterios de prueba y la realización de las mismas para determinar si se han cumplido con esos criterios. Probar un sistema que utiliza SOA puede ser complejo por varios motivos:

- Puede ser necesaria la interacción entre partes distribuidas del sistema.
- La organización puede no tener acceso al código fuente de los servicios (ya que estos servicios pueden ser externos) por lo que no podría identificar los casos de pruebas necesarios para probar.
- Hay servicios que se conocen en tiempo de ejecución por lo que es imposible predecir qué servicios o conjuntos de servicios se utilizan hasta que el sistema no esté en ejecución.

5.1.11. Auditabilidad

La auditabilidad es el grado en que un sistema apoya auditorías, ya sea financieras o legales, mediante el mantenimiento de registros de datos del sistema. Cuando se utiliza SOA, la auditoría puede ser difícil de realizar. Por ejemplo, si una aplicación que utiliza un enfoque SOA utiliza servicios externos dinámicamente, puede ser difícil rastrear qué servicio está utilizando actualmente [20].

5.1.12. Operabilidad y Capacidad de despliegue

La operabilidad y capacidad de despliegue refieren a la capacidad de los sistemas basados en SOA de operar en un ambiente de operaciones automatizadas y con recuperación automática [20].

5.1.13. Modificabilidad

La modificabilidad es la capacidad de realizar cambios en un sistema de forma rápida y rentable. Los servicios son autónomos, modulares y se acceden por interfaces cohesivas. Estas características

contribuyen a la creación en SOA del bajo acoplamiento donde hay poca dependencia entre los servicios [20].

5.1.14. Valor de negocio

El valor de negocio de un servicio web es el valor económico resultante de aplicar servicios web en un negocio y la calidad de éste proporciona una perspectiva del negocio que permite seleccionar correctamente los servicios, evaluando el valor de negocio de los servicios web [21].

5.2.Modelos de QoS para servicios

En esta sección presentaremos dos modelos de calidad de servicios: un modelo conceptual de calidad para servicios web propuesto por OASIS y otro de medición de la ejecución de procesos de negocio.

5.2.1. Quality Model for Web Services

En el 2005 el comité técnico Web Service Quality Model de la organización OASIS [22] publicó un modelo conceptual de calidad para servicios web [23], el cual consiste de tres componentes: factor de calidad (*Quality Factor*), actividad de calidad (*Quality Activity*) y rol de calidad (*Quality Associate*). Este modelo tiene como objetivo contribuir a que las personas involucradas en el ciclo de vida de los servicios web entiendan cómo describir y evaluar la calidad de éstos. El factor de calidad refiere a un grupo de elementos que representan las propiedades funcionales y no funcionales de los servicios web; el componente rol de calidad refiere a los roles, tareas de las organizaciones o personas relacionadas a los servicios web; y la actividad de calidad refiere a las acciones realizadas por los roles para mantener la estabilidad de la calidad de los servicios web. El modelo de calidad de servicios web ilustrado en la Figura 16 refiere a los componentes de calidad y sus relaciones. A continuación se describe brevemente cada uno.

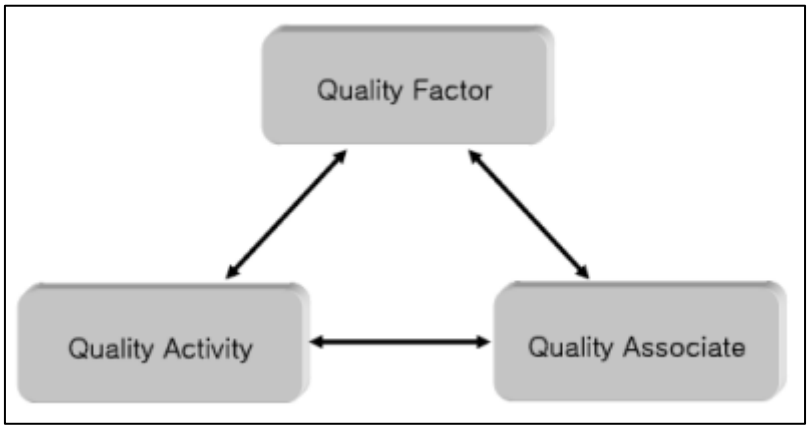

Figura 16. Modelo de Calidad de Servicios Web de [23]

5.2.1.1. Factores de calidad de servicios web

La calidad de los servicios web puede ser considerada en tres capas: capa de nivel de negocio, capa de nivel de servicio y capa de nivel de sistema; cada una de éstas tiene uno o más sub-factores de calidad. La capa de nivel de negocio es la calidad de representar el valor de negocio percibido por el usuario cuando utiliza los servicios web. La capa de nivel de servicio es la calidad mensurable de performance de los servicios web percibida por el usuario al utilizar los servicios web. Este factor de calidad incluye los

problemas de performance como estabilidad, escalabilidad y tiempo de respuesta. Por último, la capa de nivel de sistema puede ser dividida en la capa de interoperabilidad y la capa de gestión y seguridad. La primera determina si los servicios web desarrollados en distintos ambientes y por distintos desarrolladores pueden inter-operar de forma correcta, y la segunda indica la calidad de gestión y el nivel de las acciones de neutralización de ataques o accesos no autorizados.

5.2.1.2. Roles de calidad de servicios web

Se le llama rol de calidad a la función que cumple una persona que participa en algún paso del ciclo de vida de un servicio web. A continuación describiremos brevemente algunos de los roles especificados en el modelo:

- ***Stakeholder:*** Es quien solicita el desarrollo de un servicio web a un desarrollador, especificando los requerimientos de calidad con los que debe cumplir el servicio. Utiliza el modelo de calidad para testear si los servicios web cumplen con los niveles de calidad especificados en los requerimientos.
- ***Desarrollador:*** Es quien considera los requerimientos de calidad y diseña la estructura necesaria para cumplir con los requerimientos.
- ***Proveedor:*** Provee los servicios web implementados por el desarrollador. La calidad de los servicios web tiene una gran importancia del lado del proveedor, ya que éste puede perder ganancias si un competidor provee servicios web de mejor calidad.
- ***Consumidor***: Es el usuario que utiliza los servicios web. De existir múltiples servicios web que ofrecen las mismas funcionalidades, el consumidor elegirá el que ofrezca la mejor calidad de servicio.

En la Figura 17 se muestran todos los roles de calidad y cómo interactúan entre sí.

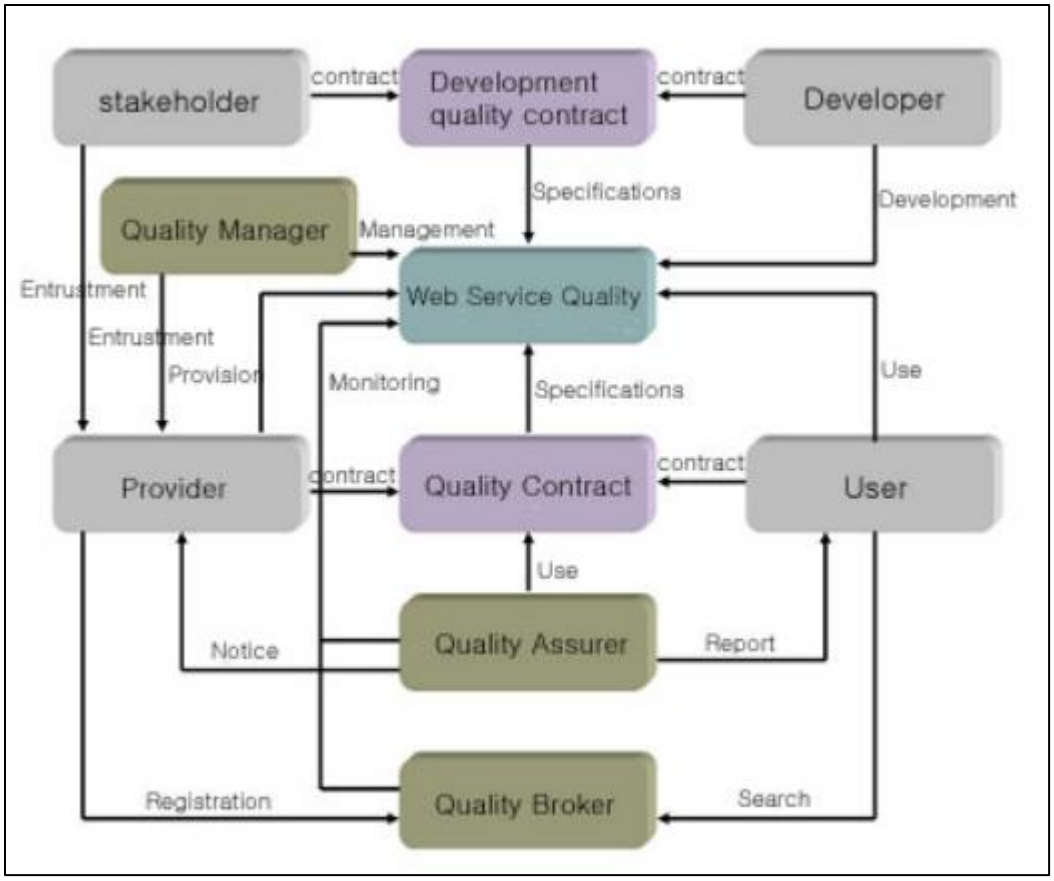

Figura 17. Roles de calidad de [23]

5.2.1.3. *Actividades de calidad de servicios web*

El componente Actividades de Calidad de este modelo consiste de varias actividades para mantener la estabilidad de los contratos de calidad de servicios web entre los roles de calidad. Existen distintos tipos de actividades, entre las que se encuentra la de tipo Contrato, la cual es la más importante y es la cooperación entre roles con los detalles de calidad de desarrollo, calidad de uso y calidad de gestión. Otras actividades son la búsqueda de detalles de calidad y búsqueda del servicio web con mejor calidad, desarrollo de servicios, publicación de detalles y niveles de calidad en un agente QoS y notificación de incumplimientos en los niveles de calidad requeridos.

El desarrollo de servicios web incluye el diseño, implementación, testing unitario e integración de servicios. Para asegurar que los servicios web son desarrollados con los niveles de calidad requeridos, los desarrolladores deben considerar la calidad en cada una de las tareas del desarrollo. Un stakeholder debe evaluar los servicios web que están siendo desarrollados e inspeccionarlos adecuadamente. Para esto, se realiza un contrato de calidad de desarrollo entre el desarrollador y el stakeholder, que debe contener los detalles de calidad que se deben alcanzar en el desarrollo y debe contener una lista de verificación para utilizarse en las inspecciones.

5.2.2. Modelo de Medición de la Ejecución de Procesos de Negocio (BPEMM)

El modelo de medición de ejecución de procesos de negocio [24] (*Business Process Execution Measurement Model,* BPEMM) es un modelo que integra medidas para procesos de negocios implementados con servicios. La medición de la ejecución de procesos de negocios provee una base para analizar el comportamiento real de una organización; ayuda a detectar deviaciones en los comportamientos planificados y a descubrir oportunidades de mejora en los procesos de negocio. Este modelo es parte de un framework llamado MINERVA [25], el cual aplica desarrollo dirigido por modelos y computación orientada a servicios para la mejora continua de procesos de negocio en las organizaciones.

Las medidas de BPEMM tienen tres elementos principales:

1. Objetivo: es la meta definida para la organización, proyecto o proceso, y es definida desde varios puntos de vista y modelos.
2. Pregunta: describe cómo será evaluado cada objetivo desde el punto de vista de una característica de calidad.
3. Medida: es un conjunto de datos objetivos o subjetivos, que es utilizado para responder cada pregunta de forma cuantitativa. Se utilizan tres tipos de medidas:
    - Medida Base: es la medida de un atributo, sin dependencias con otras medidas, y cuyo enfoque de medición es un método de medición.
    - Medida Derivada: es una medida derivada de medidas base y otras medidas derivadas, cuyo enfoque de medición es una función de medición.
    - Indicador: es una medida derivada de otras medidas, cuyo enfoque de medición es un modelo analítico con un criterio de decisión asociado.

Este modelo agrupa las medidas de ejecución de acuerdo a tres vistas específicas: procesos de negocio genéricos, ejecución eficiente (*Lean*), y ejecución de servicios. La vista de procesos de negocio genéricos incluye la duración de actividades, duración del proceso completo, costos, roles involucrados y la calidad percibida del usuario. La vista Lean define medidas de ejecución utilizadas para recolectar información para la detección de derroches en la ejecución de procesos de negocio. Apunta a encontrar actividades o caminos en el proceso que puedan guiar a una optimización y mejora del proceso de negocio. La vista de ejecución de servicio contiene medidas asociadas a la ejecución de los servicios que implementan el proceso de negocio, teniendo en cuenta requerimientos QoS. En este proyecto nos es de mayor interés la vista de ejecución de servicios, ya que las medidas definidas en esta vista están fuertemente ligadas al modelado de características QoS.

Para calcular las medidas de cada servicio invocado, la información de cada invocación debe ser registrada en un log. El servidor de aplicaciones que ejecuta el servicio puede registrar el tiempo, el origen y las credenciales de cada invocación del servicio.

En la Tabla 1 se muestran las medidas definidas en BPEMM para el tiempo de respuesta de servicio a modo de ejemplo.

Tabla 1. Medidas de Tiempo de Respuesta para la ejecución de servicios de [24]

Objetivo	**O1**	**Garantizar el tiempo de respuesta promedio del servicio en L1 segundos**
Pregunta	**P1**	**¿Cuál es el tiempo de respuesta promedio actual del servicio?**
Métricas	M1 (base)	Tiempo de invocación del servicio desde la actividad en el PN (IT = timestamp)
	M2 (base)	Tiempo de habilitación del servicio (ET = timestamp)
	M3 (base)	Tiempo de inicio del servicio (ST = timestamp)
	M4 (base)	Tiempo de terminación del servicio (CT = timestamp)
	M5 (base)	Tiempo de falla del servicio (FT = timestamp)
	M6 (base)	Tiempo de respuesta desde el servicio a la actividad en el PN (AT = timestamp)
	M7 (derivado)	Tiempo de procesamiento del servicio (SPoT = CT – ST)
	M8 (derivado)	Tiempo de latencia del servicio (SLaT = ST – ET)
	M9 (derivado)	Tiempo de respuesta del servicio (SRpT = SPoT + SLaT)
	M10 (derivado)	Tiempo de contestación desde el PN (SAnT = AT – IT)
	M11 (indicador)	Tiempo de procesamiento vs. Tiempo de latencia (STI = SLat/SPoT) Criterio de decisión = Index DC
	M12 (indicador)	Tiempo de respuesta promedio en todos los casos del PN (ASRpT = ∑SRpT/Tiempo total de ejecuciones en todos los casos del PN) Criterio de decisión = Index DC
	M13 (indicador)	Tiempo de contestación promedio en todos los casos del PN (ASAnT = SAnT/Tiempo total de ejecuciones en todos los casos del PN) Criterio de decisión = Index DC
Criterio de decisión	Index DC	R1: 0<=TTI<= L1 = ''LOW'' = GREEN; R2: L1<= TTI < L2 = ''MEDIUM'' = YELLOW; R3: L2<= TTI = ''HIGH'' = RED

En [24] se pueden ver todas las tablas de las medidas de ejecución definidas, que serán también explicadas en el capítulo del caso de estudio en que usamos BPEMM para especificar las QoS de los servicios ya modelados en SoaML para la implementación de un proceso de negocio seleccionado.

6. Modelado de calidad de servicios con QoS

La especificación *Quality of Service & Fault Tolerance* [18] define un conjunto de extensiones UML para representar conceptos de calidad de servicio y tolerancia a fallas, e integra estas extensiones en dos frameworks generales: framework de modelado QoS y framework de modelado de tolerancia a fallas. En este proyecto trataremos únicamente el framework general para modelado QoS, el cual provee la habilidad de asociar los requerimientos y propiedades de calidad a elementos del modelo, y así introducir aspectos no funcionales a los modelos UML.

El framework de modelado QoS define un metamodelo y un perfil UML. El metamodelo especifica un lenguaje abstracto de lenguajes de modelado de conceptos QoS, los cuales son representados como metaclases. El perfil define extensiones al metamodelo con el propósito de adaptarlo a una plataforma o dominio específico, sin cambiar su semántica [18].

Tanto el metamodelo como el perfil QoS se dividen en tres paquetes: el paquete *QoSCharacteristics,* que incluye los elementos de modelado que describen las características QoS, el paquete *QoSContraints,* que incluye los elementos de modelado que describen los contratos y restricciones QoS y el paquete *QoSLevels*, que incluye los elementos de modelado para la especificación de los niveles y transiciones QoS [18].

6.1.Elementos de QoS

A continuación se definen los principales elementos del metamodelo de QoS según [18]. En la Figura 18 se muestran las metaclases que extiende cada estereotipo del perfil.

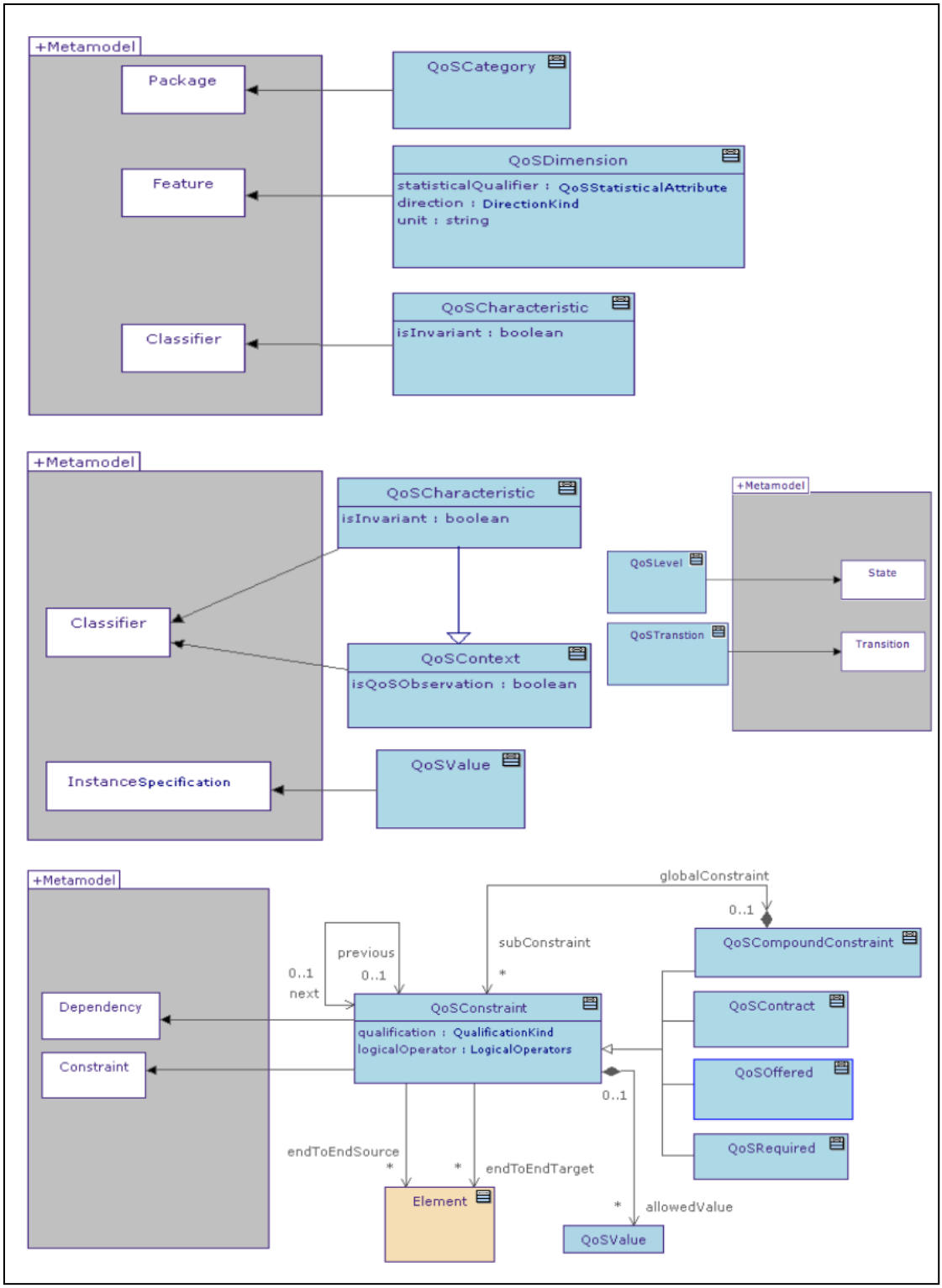

Figura 18. Perfil QoS de [18]

6.1.1. QoS Characteristic

Una *QoS Characteristic* representa una característica cuantificable de servicios y describe aspectos no funcionales. Por ejemplo, los atributos de calidad definidos en la sección 5.1 pueden ser modelados como elementos QoS Characteristic. Este elemento extiende la metaclase *Classifier* de UML.

6.1.2. QoS Dimension

Una *QoS Dimension* define una medida para cuantificar una QoS Characteristic. Por ejemplo, las medidas definidas en BPEMM [24] pueden ser modeladas como elementos QoS Dimension.

Una QoS Dimension contiene tres propiedades, los cuales se describen a continuación.

- ***Direction***: Indica de qué manera deben compararse dos valores distintos de una misma QoS Dimension. Éste es un enumerado cuyo valor puede ser creciente, decreciente o indefinido. Cuando el atributo toma el valor creciente, cuanto mayor es el valor, mejor es la calidad. En cambio, si el atributo toma el valor decreciente, cuanto menor es el valor, mejor es la calidad. Por ejemplo, si estamos midiendo el tiempo de respuesta de un sistema, será mejor aquel que tenga menor tiempo, por lo tanto la dirección será decreciente.
- ***Unit***: Indica la unidad de los valores de la dimensión. Por ejemplo, si se está midiendo el tiempo de respuesta, entonces la unidad podrá ser milisegundos.
- ***StatisticalQualifier***: Representa el tipo de clasificación estadística (valor máximo, valor mínimo, varianza, desviación estándar, entre otros). Por ejemplo, si se está midiendo el tiempo de respuesta, podría decirse que como máximo el servicio responderá en 1000 ms, es decir, el atributo statisticalQualifier será “valor máximo”.

Este elemento extiende la metaclase *Feature* de UML.

6.1.3. QoS Category

Una *QoS Category* agrupa características de calidad, permitiendo la categorización de éstas. En [18] se define un conjunto de categorías QoS, entre las que se encuentra Desempeño, la cual agrupa las características Latencia y Rendimiento. Este elemento extiende la metaclase *Package* de UML. En la Figura 19 se puede ver la relación entre los elementos QoS Category, QoS Characteristic y QoS Dimension dentro del paquete QoS Characteristics.

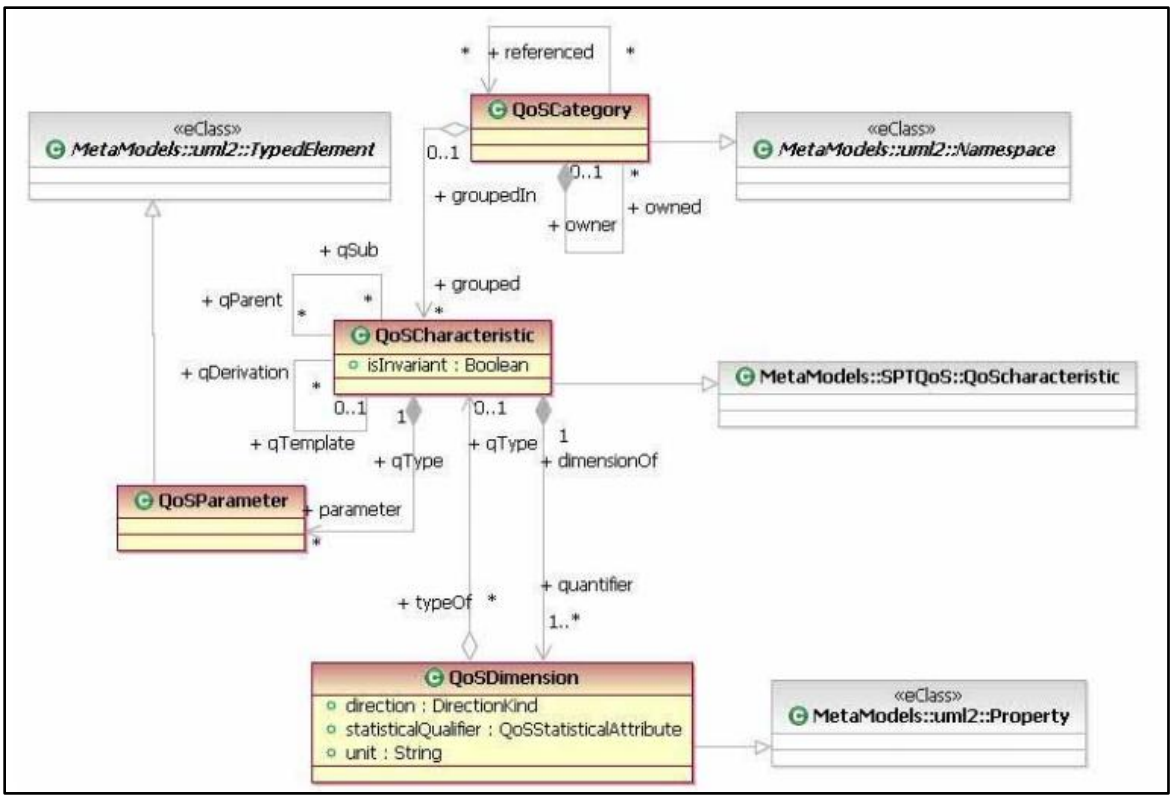

Figura 19. Diagrama de QoS Characteristics de [18]

6.1.4. QoS Value

Una *QoS Value* es una instancia de una QoS Characteristic y define los valores específicos para cada medida modelada con una QoS Dimension. Cuando se asocia un QoS Value a un elemento del modelo, se está caracterizando a ese elemento con los valores de calidad definidos. Por ejemplo, si se quiere modelar el tiempo de respuesta de un servicio, se le podrá asociar un QoS Value que instancie la característica Tiempo de Respuesta y así definir el tiempo máximo que el servicio podrá satisfacer. Este elemento extiende la metaclase *InstanceSpecification* de UML. En la Figura 20 se muestra el elemento QoS Value y sus relaciones con los demás elementos dentro del paquete QoS Values.

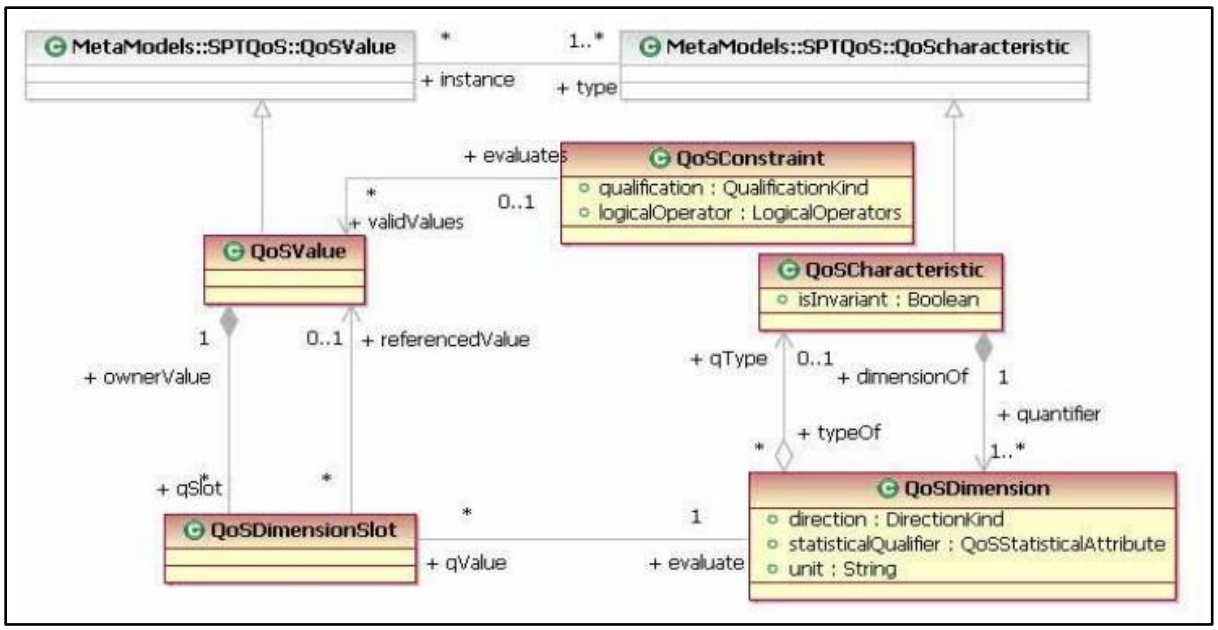

Figura 20. Diagrama de QoS Values de [18]

6.1.5. QoS Constraint

Una *QoS Constraint* permite caracterizar elementos del modelo con aspectos no funcionales. En [18] se especifican dos formas para modelar restricciones de calidad en los elementos del modelo: como una *Constraint* UML en la que se especifica una expresión OCL, o como una *Dependency* que asocia un elemento QoS Value con otro elemento del modelo. Debido a estas formas distintas de agregar restricciones de calidad, este elemento extiende las metaclases *Dependency* y *Constraint* de UML. El elemento QoS Constraint es un estereotipo abstracto, del cual existen tres especializaciones definidas como dependencias entre elementos QoS Value y otros elementos del modelo:

- ***QoS Offered***: Se utiliza para modelar las características de calidad ofrecidas. Cuando el proveedor define una dependencia QoS Offered entre un valor de calidad y uno de los servicios que provee, es él quien debe satisfacer dicha característica, y cuando es el consumidor quien la define, él debe satisfacerla al invocar el servicio.
- ***QoS Required***: Se utiliza para modelar las características de calidad requeridas. Cuando el proveedor define una dependencia QoS Required entre uno de los servicios que provee y un valor de calidad, el consumidor debe cumplir con dicha característica para garantizar la calidad ofrecida por el proveedor. Por ejemplo, el proveedor podría requerir una frecuencia máxima de invocaciones al servicio para satisfacer el tiempo de respuesta ofrecido. Cuando el consumidor define una dependencia QoS Required, es el proveedor del servicio el que deberá satisfacer los requerimientos de calidad del consumidor.
- ***QoS Contract***: Se utiliza para modelar las características de calidad acordadas entre un proveedor y un consumidor en un contrato de servicios. Los consumidores de servicios pueden tener diferentes requerimientos de calidad de servicio, por ejemplo distintos anchos de banda de bajada de datos. Por otro lado, los proveedores de servicios normalmente ofrecen distintos niveles de calidad (suscripciones preferenciales o premium) [26]. Antes de que un consumidor

se suscriba al proveedor, necesita establecer un contrato como un acuerdo mutuo con el proveedor, detallando los niveles de garantía de varias características QoS. Una vez creado el contrato, ambas partes deben atenerse a éste. Por ejemplo, el consumidor no debe excederse en la cantidad de solicitudes, y el proveedor debe cumplir con los niveles de performance acordados [26].

En la Figura 21 se pueden ver los elementos mencionados en esta sección.

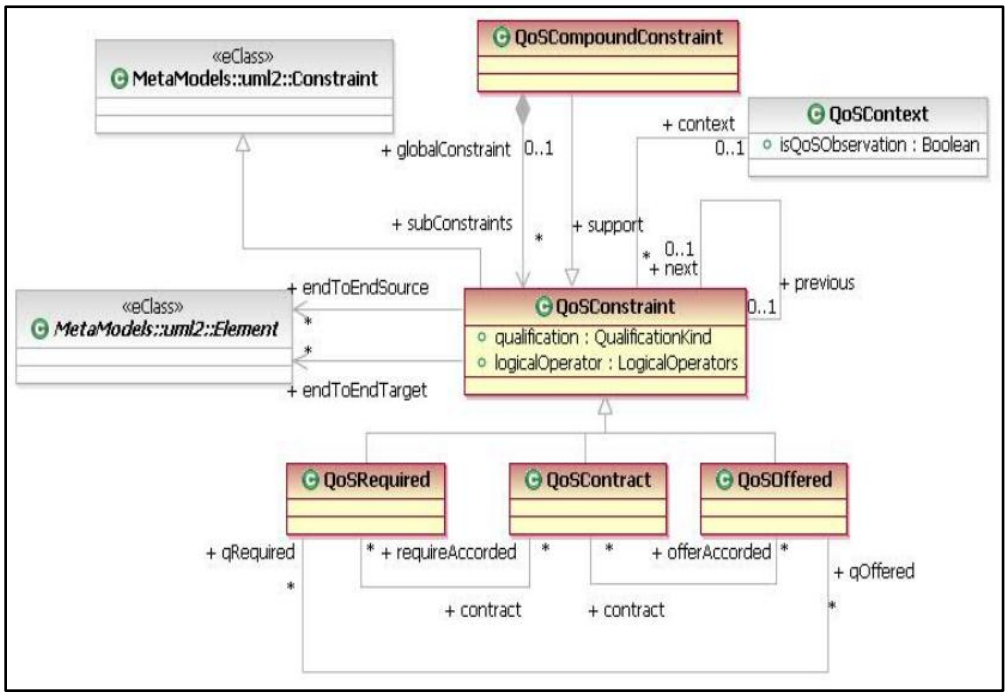

Figura 21. Diagrama de QoS Constraints de [18]

6.1.6. QoS Level

Un QoS Level representa un modo de ejecución del sistema. Cada uno de los modos posibles del sistema provee diferentes niveles de calidad para un mismo servicio y requiere diferentes recursos. El QoS Level actual de un sistema depende de los recursos disponibles, la calidad requerida, y de ciertos parámetros como variables de estado que identifican la configuración actual.

6.1.7. QoS Transition

Una QoS Transition modela las condiciones necesarias para que el sistema cambie de un QoS Level a otro. Estas transiciones de modos se pueden dar cuando un componente del sistema recibe muchos pedidos generando una sobrecarga en los recursos; los requisitos de calidad del QoS Level actual ya no se pueden satisfacer y es necesario realizar una transición a otro nivel.

6.2.Diagramas de QoS

A continuación se explicarán los distintos diagramas que se pueden definir en QoS según [18].

6.2.1. Diagrama de Categorías

El Diagrama de Categorías permite definir dependencias entre categorías. Para esto, se pueden definir elementos QoS Category que podrán ser agrupados dentro de otros elementos QoS Category y también podrán incluirse unos a otros. Esto se puede ver en la Figura 22.

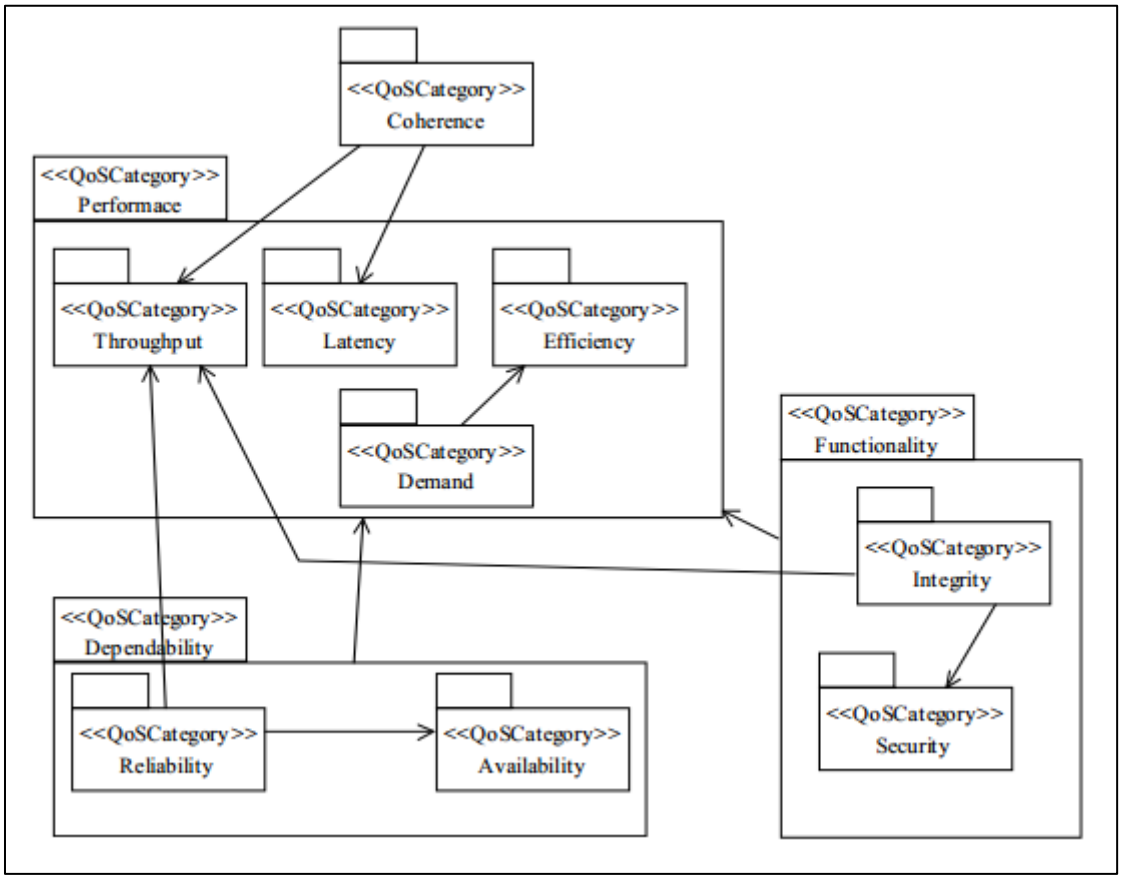

Figura 22. Ejemplo de Diagrama de Categorías de [18]

6.2.2. Diagrama de Características de Calidad

El Diagrama de Características de Calidad permite definir una taxonomía de características de calidad, especificando las dimensiones de cada característica y definiendo las relaciones entre éstas. En la Figura 23 se puede ver un ejemplo de un diagrama de este tipo.

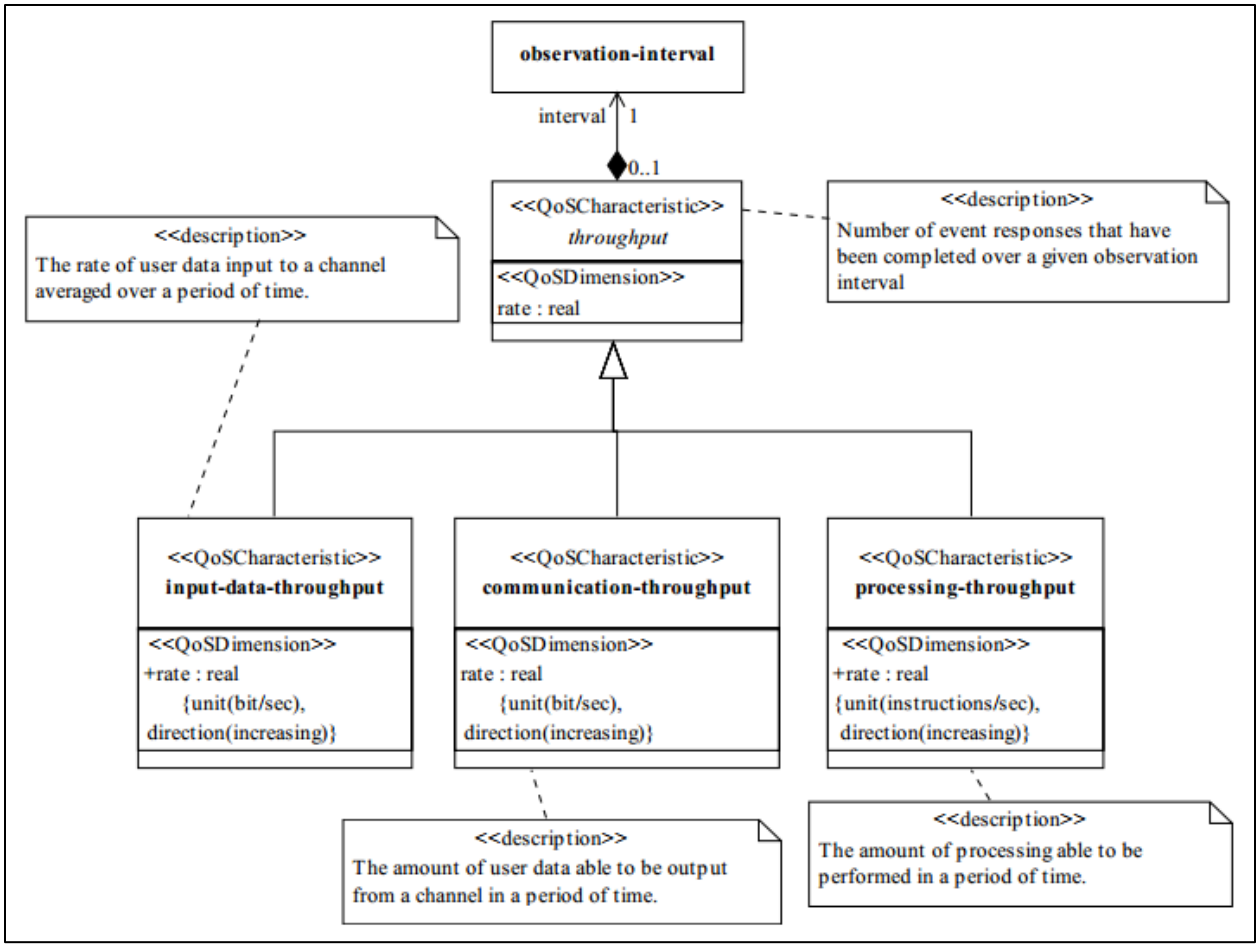

Figura 23. Ejemplo de Diagrama de Características de Calidad de [18]

6.2.3. Diagrama de Restricciones de Calidad

Los Diagramas de Restricciones de Calidad son diagramas UML de cualquier tipo, en el cual se aplican restricciones de calidad a los elementos del modelo. En estos diagramas se modelan los aspectos no funcionales del sistema, especificando los valores de las características de calidad que son ofrecidos, requeridos y negociados por los participantes. En la Figura 24 se puede ver el extracto de un diagrama de restricciones de calidad de [18].

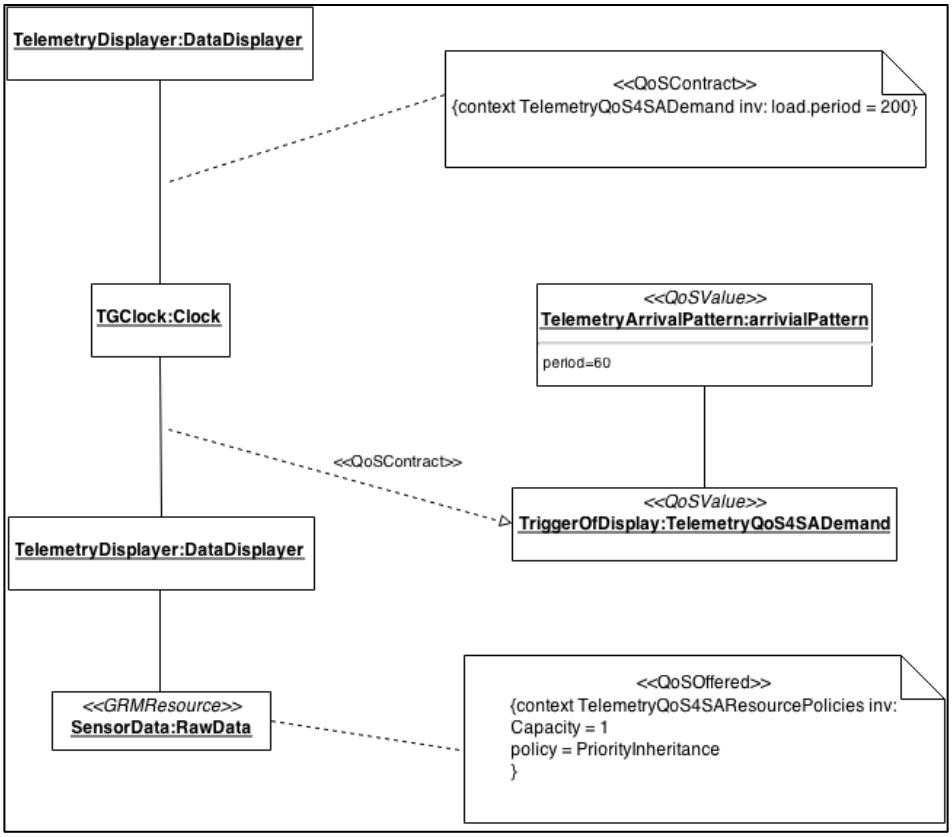

Figura 24. Ejemplo de Diagrama de Restricciones de Calidad de [18]

7. Modelado de SoaML con QoS

La especificación QoS [18] define un conjunto de elementos que pueden ser utilizados para modelar las características de calidad de un sistema, pero no especifica cómo incluir estas características en modelos de dominios específicos, como lo son los modelos SoaML. En [19], Schot investiga distintas opciones para modelar características de calidad en arquitecturas SOA en dos niveles de modelos: PIM y PSM.

Para los modelos PIM, utiliza la especificación SoaML [17] para el modelado de servicios y el estándar QoS para el modelado de características de calidad. Para modelar las características de calidad que ofrece un servicio, se decide asociar elementos *QoSValue* al puerto de tipo *Service* utilizando una dependencia *QoSOffered*. De forma análoga, se asocian elementos *QoSValue* al puerto de tipo *Request* mediante dependencias *QoSRequired* para modelar que un consumidor espera ciertos niveles de calidad del servicio que va a utilizar. Para modelar las características de calidad acordadas en un contrato entre el proveedor y consumidor de un servicio, se asocian elementos *QoSValue* a elementos *ServiceContract* utilizando dependencias *QoSContract*. En la Figura 25 se puede ver un ejemplo de estas asociaciones.

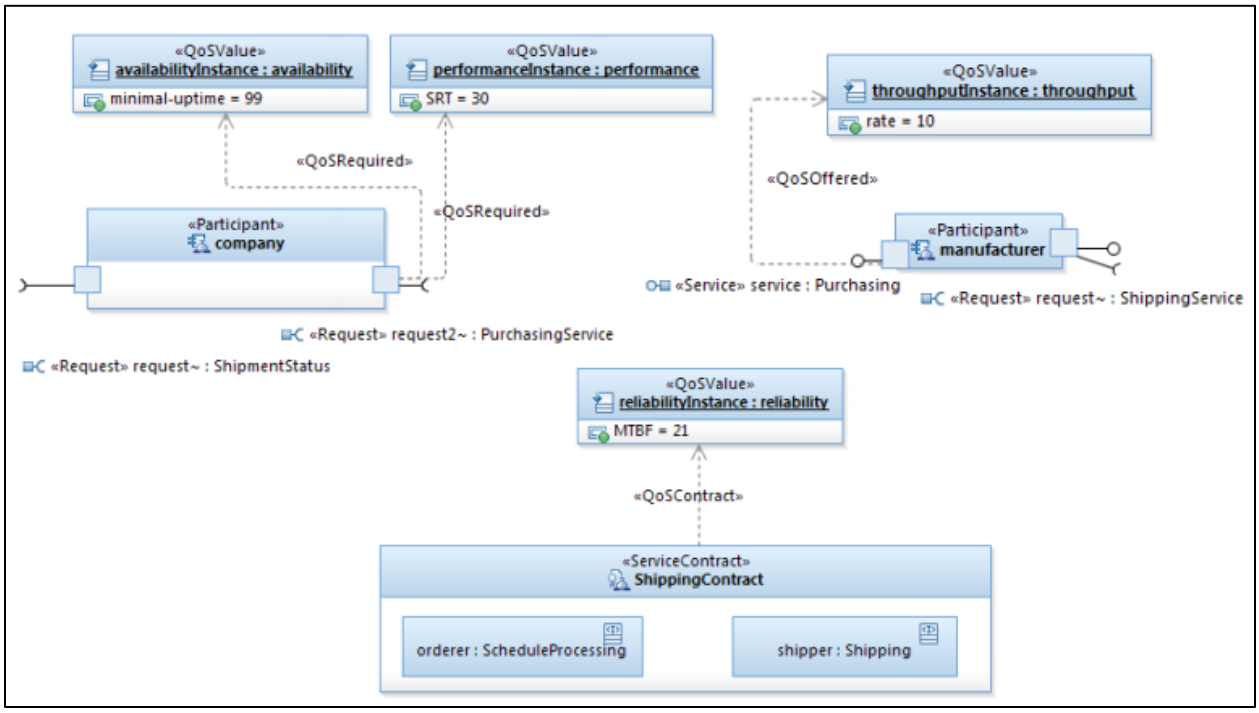

Figura 25. Ejemplo de características QoS aplicadas a modelos SoaML de [19]

A nivel de PSM, Schot [19] decide mapear las características QoS en modelos SoaML de dos formas distintas:

- Mapear los elementos QoSValue asociados a puertos de servicio y de pedido a políticas expresadas con el estándar WS-Policy [27], definiendo una gramática simple para modelar las aserciones de las políticas.
- Mapear los elementos QoSValue asociados a contratos de servicio a documentos WS-Agreement [28].

En la sección 9.4 se describen en detalle estos estándares. Los dos tipos de mapeos se describen en [19], indicando qué elementos del modelo origen deben ser transformados a los elementos del modelo destino. Los documentos WS-Agreement que se generan no están completos, ya que los modelos SoaML con características QoS no contienen toda la información que se requiere en el acuerdo.

8. Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma que es utilizado para distintas tareas, principalmente para desarrollar software. La arquitectura de Eclipse es un micro-núcleo que emplea módulos para proporcionar toda su funcionalidad, y permite añadir nuevas funcionalidades fácilmente [29]. Es una arquitectura basada en plugins. Un plugin es una unidad mínima de funcionalidad que permite escribir nuevas extensiones incrementando de este modo las funcionalidades provistas por el ambiente. Para esto, cada plugin Eclipse puede ofrecer puntos de extensión que permiten la posibilidad de agregar nuevas funcionalidades. En la Figura 26 se presenta un esquema de la arquitectura descripta.

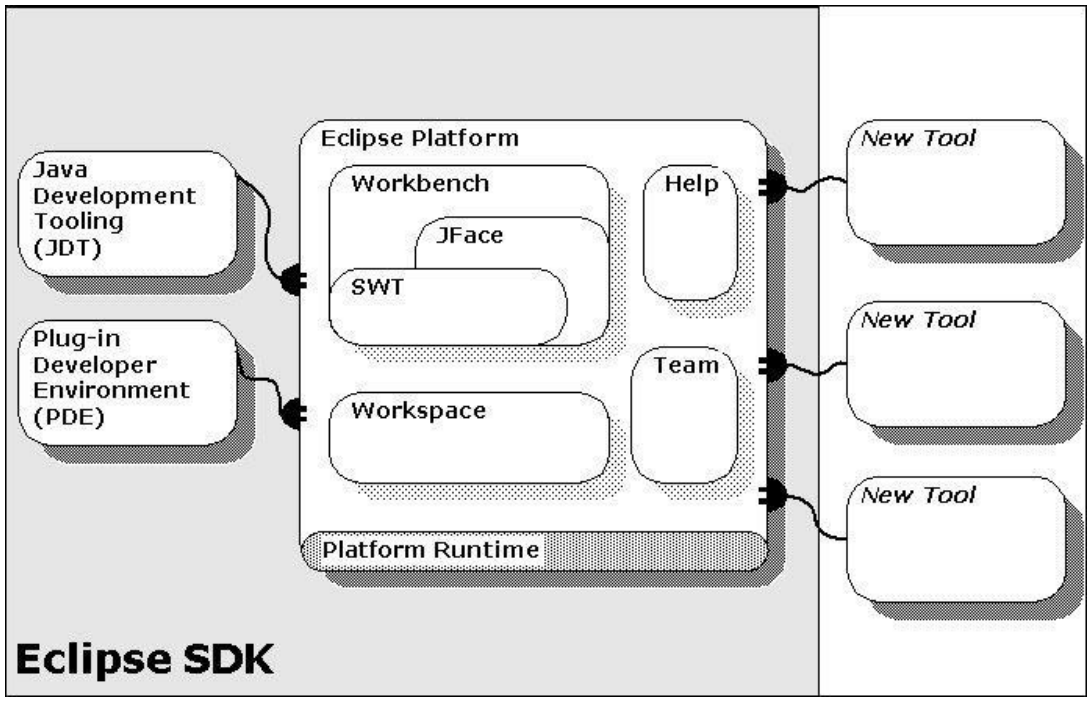

Figura 26. Arquitectura de Eclipse de [29]

9. Tecnología Web Services

Un servicio web es un sistema de software diseñado para apoyar la interoperabilidad entre diferentes aplicaciones, ejecutado en una variedad de plataformas, el cual tiene una interfaz descrita en un formato procesable por máquina (WSDL). Otros sistemas interactúan con el servicio web utilizando mensajes SOAP, los cuales son generalmente transportados utilizando HTTP con una serialización XML [30]. Existe también otro enfoque para servicios web, el enfoque REST, que es una técnica de arquitectura de software para sistemas distribuidos y es utilizada para describir cualquier interfaz web que utilice XML y HTTP [31].

La arquitectura tradicional de servicios web implica varias tecnologías, algunas de las cuales se pueden ver en la Figura 27.

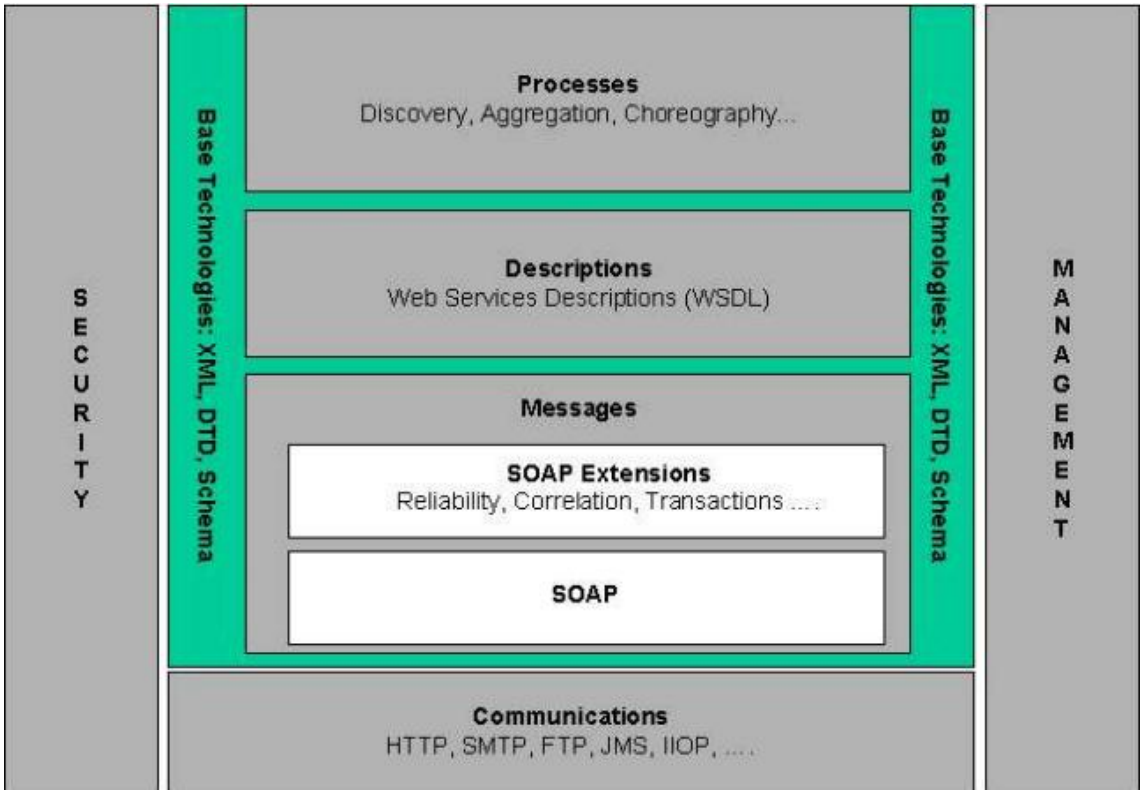

Figura 27. Arquitectura de Web Services de [30]

A continuación se describen algunos de los componentes relevantes de la arquitectura de un Web Service.

9.1.SOAP

Protocolo Simple de Acceso a Objetos, SOAP por su sigla en inglés, es un protocolo basado en XML que permite la interacción de múltiples objetos en diferentes procesos y tiene la capacidad de transmitir información compleja. Los datos pueden ser transmitidos a través de distintas tecnologías mostradas en la capa Communications de la Figura 27 (HTTP, SMTP, FTP, etc.). SOAP especifica el formato de los mensajes. El mensaje SOAP está compuesto por un *envelope* (sobre), cuya estructura está formada por los siguientes elementos: *header* (cabecera) y *body* (cuerpo) [32].

9.2.REST

REST (REpresentational State Transfer) es una técnica de arquitectura de software para sistemas distribuidos y es utilizado para describir cualquier interfaz web que utilice XML y HTTP. Es un protocolo de mensajería alternativo para SOAP.

Una implementación concreta de un Web Services REST sigue cuatro principios de diseño fundamentales:

1. La utilización de los métodos HTTP de manera explícita siguiendo el protocolo definido por la RFC 2616.
2. No mantener estados, es decir se deben enviar peticiones completas e independiente, que incluyan todos los datos necesarios para cumplir el pedido. De esta forma se mejora el rendimiento de los Web Services y se simplifica el diseño e implementación de los componentes del servidor, ya que se elimina la necesidad de sincronización de datos en el servidor.
3. Definir URIs con forma de directorio, las cuales deben ser intuitivas. Este tipo de URIs son jerárquicas con una única ruta raíz y va abriendo ramas a través de las sub rutas para exponer las áreas principales del servicio.
4. REST transfiere XML, JSON o ambos [31].

9.3.WSDL

WSDL (Web Services Description Language) es una especificación estándar basada en XML que se utiliza para describir servicios web. En éste se definen los formatos del mensaje, tipos de datos, protocolos de transporte y formatos de serialización de transporte que deben ser utilizados entre el consumidor y el proveedor. También especifica las ubicaciones de red en las que puede ser invocado, y puede proporcionar alguna información sobre el patrón de intercambio de mensajes que se espera [29].

Un documento WSDL se forma mediante un conjunto de componentes abstractos y componentes concretos. Los mismos deben ser definidos dentro del documento en un orden determinado. En la Figura 28 se muestra un esquema de la estructura de los componentes abstractos.

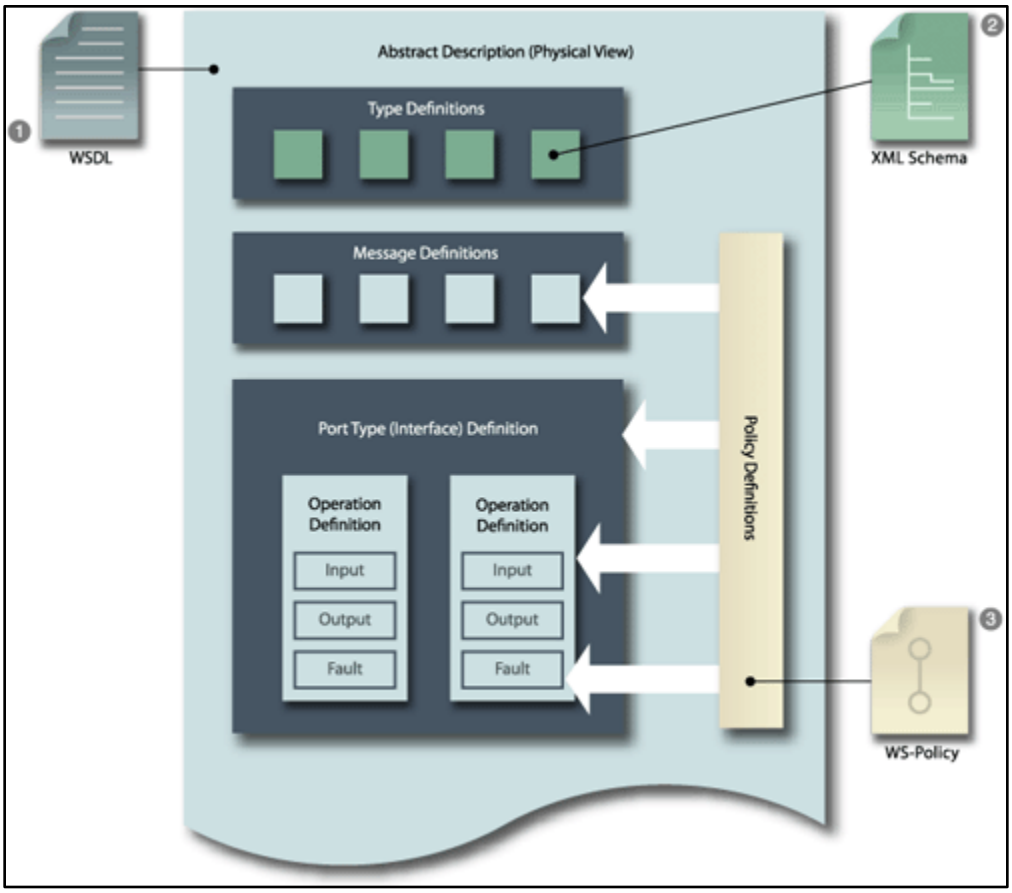

Figura 28. Descripción de los componentes abstractos en un WSDL de [33]

Los componentes abstractos que forman el WSDL son la definición de tipos (Type Definitions), la definición de mensajes de intercambio (Message Definitions) y la definición de interfaces que contienen los servicios (Port Type Interface Definitions).

En *Type Definitions* se definen los tipos de datos relevantes para el intercambio de mensajes. Para conseguir una mejor interoperabilidad se puede utilizar un XML Schema para definir los tipos.

Message Definitions representa un mensaje abstracto que el servicio intercambia con el consumidor. Se puede declarar cualquier cantidad de mensajes y los mismos deben contener un nombre único el cual actúa como identificador para poder ser referenciado por alguna operación.

PortType Definitions funciona como una interfaz donde se define el conjunto de operaciones que pueden ser realizadas y los mensajes involucrados tanto para la petición como para la respuesta.

Una vez definidos los elementos que forman la parte abstracta del documento WSDL, se deben definir los componentes concretos del mismo, como se muestra en la Figura 29.

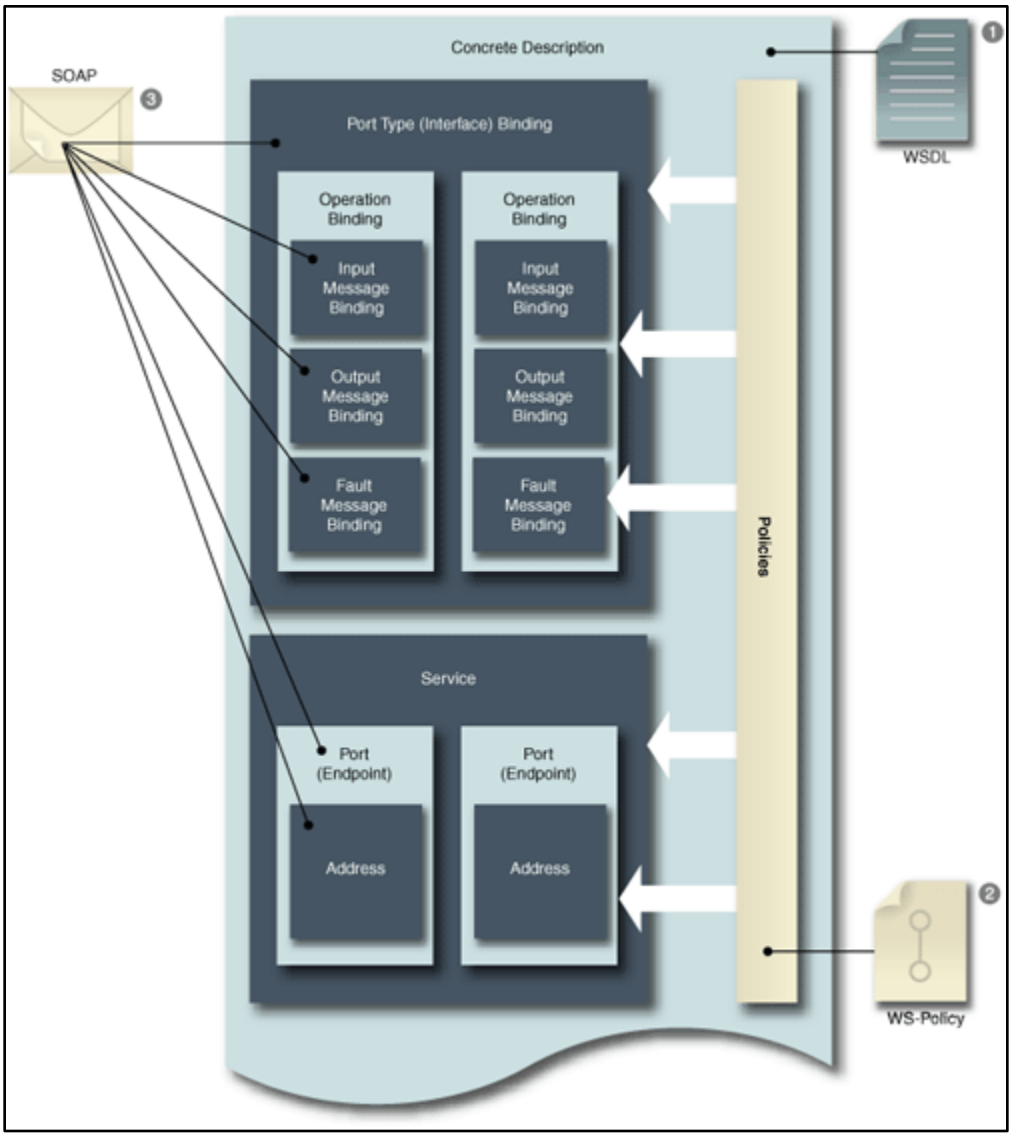

Figura 29. Descripción de los componentes concretos en un WSDL de [33]

Los componentes concretos que forman el WSDL son el enlace entre la definición abstracta y su implementación (PortType Binding) y la definición del servicio (Service).

PortType Binding especifica el protocolo utilizado y el formato de datos para cada *PortType*.

Luego de tener el bindeo, se define el *Service* donde se asigna una dirección física, la cual será usada por los consumidores para invocar el Web Service. Esto se define dentro del elemento *Port* (*Endpoint*), el cual referencia un único bindeo, por lo tanto un único *PortType*.

9.4.Implementación de WS

Para la implementación de Web Services existen dos enfoques, *Top Down* y *Bottom Up*. La primera consiste en crear el servicio web a partir de un archivo WSDL. En cambio, en la segunda, se comienza por

el código y se genera a partir de éste el WSDL correspondiente [34]. Una comparación de estos dos enfoques se puede ver en la Tabla 2.

Tabla 2. Ventajas y desventajas de los enfoques Top Down y Bottom Up de [34]

	Top Down	Bottom Up
Ventajas	1. Permite el desarrollo independiente y paralelo entre cliente y servicio. 2. Cuando se crean nuevos tipos, éstos pueden ser reutilizados por otros servicios. 3. No se generan dependencias con la implementación.	1. Forma rápida de exponer funcionalidades vía WS. 2. No requiere conocimiento de XMLSchema, WSDL.
Desventajas	1. Requiere conocimiento de XMLSchema y WSDL.	1. El esquema es embebido en el WSDL. 2. Los cambios en la interfaz son más difíciles de manejar. 3. En el WSDL hay dependencias con la implementación (paquetes y namespaces).

9.5.WS-Policy

El estándar WS-Policy [27], publicado por W3C, es utilizado para expresar políticas de Web services basándose en sus propiedades no funcionales. Éste define un framework y un modelo para expresar requerimientos y características generales de entidades en un sistema basado en servicios web, en formato XML.

9.5.1. Modelo de Políticas

En esta sección se describe un modelo abstracto de políticas y operaciones sobre políticas, que está compuesto por tres elementos principales: aserción de política (*Policy Assertion*), alternativa de política (*Policy Alternative*) y política (*Policy*).

9.5.1.1. Policy Assertion

Una aserción de políticas (*policy assertion*) representa un requerimiento, una capacidad o una propiedad. Cada aserción tiene un tipo, que está definido por las propiedades "*namespace*" y "*nombre local*" del elemento raíz que representa la aserción. Los autores de políticas pueden definir una aserción que contenga una expresión de políticas como uno de los hijos de la aserción. Se pueden definir parámetros de comportamiento en una aserción mediante la especificación de atributos o hijos en la aserción. Por ejemplo, una aserción puede requerir que se utilice cierto tipo de encriptado en los mensajes, o puede describir ciertas características de calidad del servicio expuesto [27].

9.5.1.2. Policy Alternative

Una alternativa de políticas (*policy alternative*) contiene una lista no ordenada de aserciones de políticas [27].

9.5.1.3. Policy

Una política (*policy*) es una lista no ordenada de alternativas de políticas, la cual es aplicada a una entidad que puede ser un mensaje, recurso, operación o *endpoint*. Cuando se aplican a un sistema basado en servicios web, las políticas son utilizadas para expresar condiciones en la interacción entre proveedores y consumidores. Por ejemplo, un consumidor puede decidir si utiliza un servicio o no dependiendo de las políticas expuestas por el proveedor [27].

Una aserción es soportada por una entidad si y sólo si la entidad satisface el requerimiento correspondiente a la aserción. Una alternativa de políticas es soportada por una entidad si y sólo si la entidad soporta todas las aserción dentro de la alternativa. Una política es soportada por una entidad si y sólo si la entidad soporta al menos una de las alternativas de la política. Notar que una entidad puede soportar una política incluso si la entidad no entiende el tipo de cada una de las afirmaciones de la política; la entidad necesita entender únicamente el tipo de las aserciones de la alternativa que soporta. Esta característica es fundamental para el versionado y despliegue incremental de nuevas aserciones, ya que esto permite a un proveedor incluir nuevas aserciones en nuevas alternativas, mientra que las entidades existentes continúan utilizando las alternativas anteriores (compatibilidad hacia atrás) [27].

9.5.2. Expresión de políticas

Una expresión de políticas es la representación en XML de una política, la cual puede ser normal o compacta. En la forma normal se enumeran todas las alternativas y todas las aserciones dentro de éstas; en la forma compacta se utiliza la menor cantidad de expresiones posibles [27]. En la Figura 30 se muestra un ejemplo de una expresión en forma normal de una política. Esta política contiene dos alternativas: si la primera alternativa es seleccionada, el cuerpo del mensaje tiene que ser firmado; si la segunda alternativa es seleccionada, el cuerpo del mensaje tiene que ser encriptado.

```
<wsp:Policy
   xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
   xmlns:wsp="http://www.w3.org/ns/ws-policy" >
  <wsp:ExactlyOne>
    <wsp:All>
       <sp:SignedParts>
          <sp:Body/>
       </sp:SignedParts>
    </wsp:All>
    <wsp:All>
       <sp:EncryptedParts>
          <sp:Body/>
       </sp:EncryptedParts>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Figura 30. Ejemplo de una expresión de política de [27]

Una expresión puede estar asociada con una IRI (Internationalized Resource Identifier). Este IRI sirve para identificar las políticas y referenciarlas desde otros elementos XML. En la Figura 31 se muestra un ejemplo de la definición de una política con un identificador y una referencia a esta política.

```
<wsp:Policy
    wsu:Id="P1"
    xmlns:wsp="http://www.w3.org/ns/ws-policy"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
    ....
</wsp:Policy>

<wsp:PolicyReference URI="#P1" />
```

Figura 31. Ejemplo de identificación y referencia de políticas

9.6.Q-WSDL

Q-WSDL es una extensión liviana de WSDL propuesta en [35] para permitir la descripción de características QoS de un servicio web, tales como rendimiento, confiabilidad, disponibilidad, seguridad, etc. En la especificación se introduce el metamodelo WSDL que es derivado del esquema XML de WSDL, al que se le agregan las características QoS y se lo llama Q-WSDL. Esta extensión sirve para:

- especificar requerimientos QoS de servicios web;
- agregar características QoS cuando se realizan consultas sobre registros de servicios web (por ejemplo UDDI);
- definir niveles de especificación de servicios (SLS) cuando se establecen niveles de acuerdos de servicios (SLA);
- agregar descripciones de QoS a la composición de servicios web;
- soportar el mapeo automático de documentos WSDL a Q-WSDL;
- soportar el mapeo automático de modelos UML a servicios en Q-WSDL.

Estos metamodelos son definidos utilizando MOF, y la producción de los esquemas se realiza con XMI. La transformación del metamodelo WSDL al metamodelo Q-WSDL se realizó agregando conceptos del perfil QoS&FT [18]. La Figura 32 ilustra el metamodelo Q-WSDL propuesto. Dentro de la caja de línea entrecortada se encuentra el metamodelo WSDL, y fuera de la caja están las metaclases y asociaciones QoS que extienden WSDL. El metamodelo no incluye elementos de gestión avanzada de QoS, como la especificación explícita de contratos o niveles QoS.

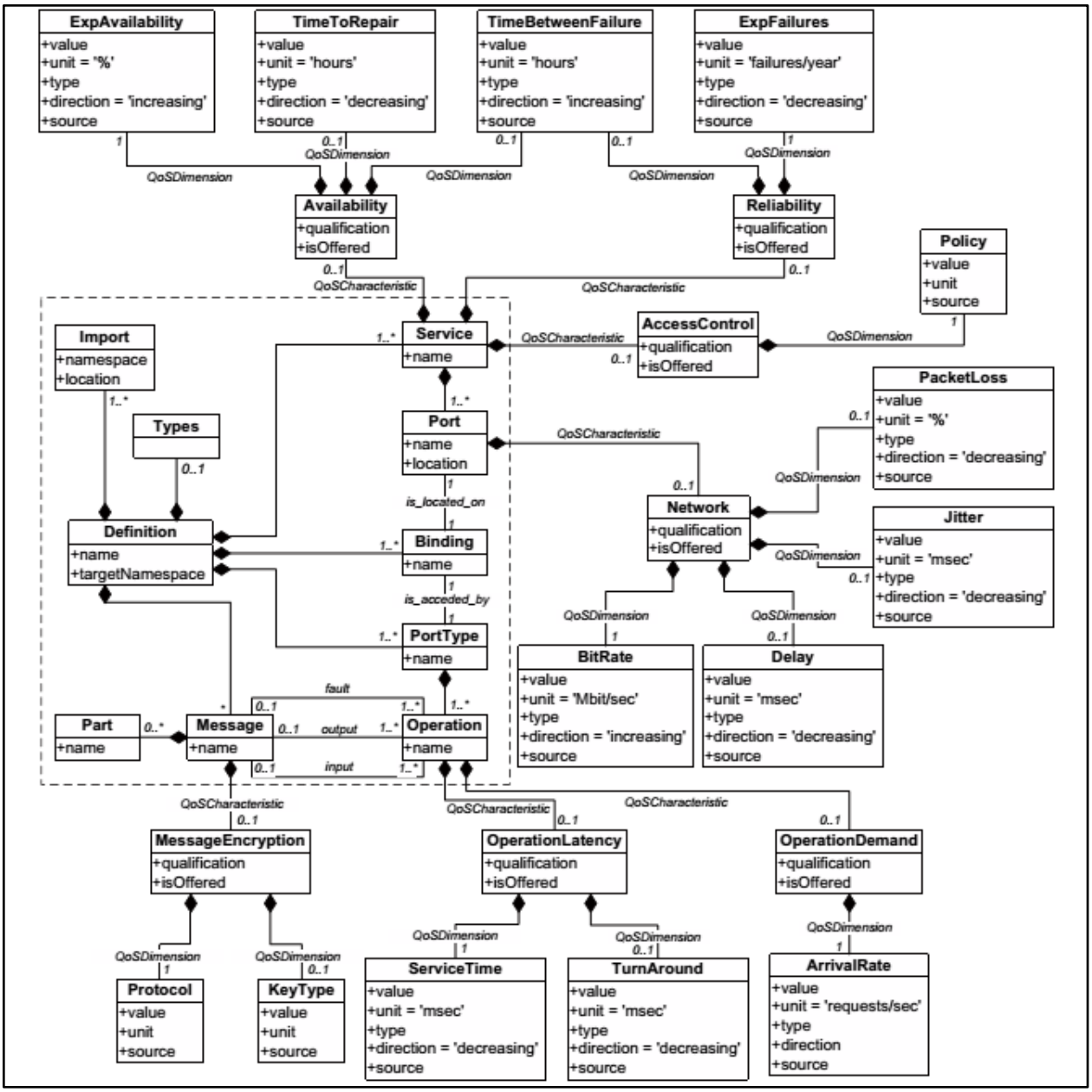

Figura 32. Metamodelo Q-WSDL de [35]

9.7.WS-Agreement

La especificación WS-Agreement [28] define un lenguaje y un protocolo para publicar las capacidades de los proveedores de servicios, crear acuerdos y monitorear el cumplimiento de los acuerdos en tiempo de ejecución.

Un acuerdo entre un proveedor y un consumidor de servicios especifica los requerimientos del consumidor y las garantías de disponibilidad de recursos y/o calidad de servicio del proveedor. Por ejemplo, un acuerdo puede proveer garantías sobre el tiempo máximo de respuesta y de la disponibilidad del servicio, así como también, garantías en la disponibilidad de recursos mínimos tales como memoria, CPU, almacenamiento, etc. [28].

Esta especificación provee un esquema para definir estructuras en general de documentos de acuerdos. Un acuerdo incluye información de las entidades participantes y de un conjunto de términos. Los términos pueden constar de al menos un término de servicios y puede contener de forma opcional términos de garantías que especifiquen objetivos de niveles de servicios y valores de negocio asociados a estos objetivos.

El proceso de creación de acuerdos normalmente comienza con una plantilla de acuerdos predefinida, que especifica aspectos del documento y reglas que deben seguirse para la creación de un acuerdo (restricciones de creación de acuerdos). Esta especificación define un esquema para una plantilla de acuerdos. La creación de un acuerdo puede ser iniciada por el consumidor de servicio o por el proveedor de servicio, y el protocolo provee mecanismos para esta simetría.

9.7.1. Estructura de un acuerdo

Un acuerdo está conceptualmente compuesto por varias partes distintas. En la Figura 33 se muestra la estructura de un acuerdo, y en la Figura 34 se encuentra la representación XML de un acuerdo.

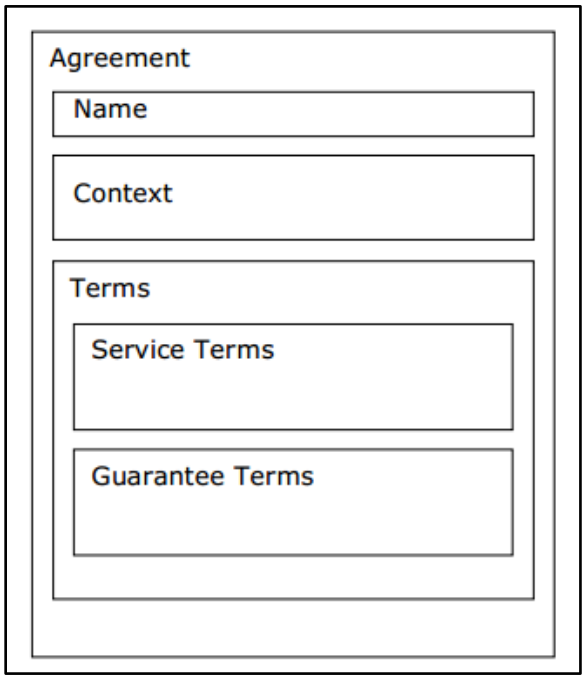

Figura 33. Estructura de un acuerdo de [28]

```
<wsag:Agreement AgreementId="xs:string">
   <wsag:Name>
       xs:string
   </wsag:Name> ?
   <wsag:AgreementContext>
      wsag:AgreementContextType
   </wsag:AgreementContext>
   <wsag:Terms>
       wsag:TermCompositorType
   </wsag:Terms>
</wsag:Agreement>
```

Figura 34. Representación XML de un acuerdo de [28]

9.7.1.1. Agreement

La etiqueta *Agreement* encapsula enteramente el acuerdo y su atributo *AgreementId* es un identificador obligatorio de esta versión particular del acuerdo. Este identificador ayuda a identificar la versión del acuerdo vigente. Si la instancia de un documento de acuerdos es modificada durante el ciclo de vida de un acuerdo, el identificador también debe ser reemplazado por un nuevo identificador único.

La etiqueta *Name* dentro de *Agreement* es un nombre opcional que puede ser dado a un acuerdo, el cual es independiente del nombre de la plantilla sobre la que está basado. El nombre no es un identificador único y puede ser utilizado para proveer un nombre entendible por humanos.

9.7.1.2. Agreement Context

El alcance de un acuerdo está asociado a su contexto, el cual normalmente incluye las partes involucradas. Adicionalmente, el contexto contiene metadata sobre el acuerdo, como la duración del acuerdo, y opcionalmente, el nombre de la plantilla sobre la que fue creada el acuerdo. En la Figura 35 se encuentra el esquema del elemento *Context*.

```
<wsag:Context xs:anyAttribute>
  <wsag:AgreementInitiator>xs:anyType</wsag:AgreementInitiator> ?
  <wsag:AgreementResponder>xs:anyType</wsag:AgreementResponder> ?
  <wsag:ServiceProvider>wsag:AgreementRoleType</wsag:ServiceProvider>
  <wsag:ExpirationTime>xs:DateTime</wsag:ExpirationTime> ?
  <wsag:TemplateId>xs:string</wsag:TemplateId> ?
  <wsag:TemplateName>xs:string</wsag:TemplateName> ?
  <xs:any/> *
</wsag:Context>
```

Figura 35. Esquema de Agreement Context de [28]

El elemento *AgreementInitiator* es un elemento opcional que identifica quién inició el proceso de creación del acuerdo. El elemento opcional *AgreementResponder* identifica a la entidad que respondió el pedido de creación del acuerdo. La valor dentro de estos elementos puede ser una URI, un wsa:EndpointReference de WS-Addressing o puede identificar al iniciador por un tipo más abstracto de nombre.

El elemento *ServiceProvider* identifica al proveedor de servicios, y puede ser *AgreementInitiator* o *AgreementResponder*.

ExpirationTime es un elemento opcional que especifica el tiempo de expiración del acuerdo. El iniciador del acuerdo puede utilizar este mecanismo para especificar el tiempo de vida del acuerdo.

El elemento opcional *TemplateID* refiere a la versión específica de la plantilla sobre la que fue creada el acuerdo. El elemento opcional *TemplateName* especifica el nombre de esta plantilla, y es útil para modificaciones futuras del acuerdo.

Se pueden agregar otros elementos o atributos de dominio específico siempre y cuando no contradigan la semántica del elemento padre.

9.7.1.3. Terms

Un término expresa derechos u obligaciones definidas para un participante. Cada término en un acuerdo tiene un tipo, que puede ser “término de servicio” o “término de garantía”. Los términos de servicio proveen la información necesaria para instanciar o identificar a un servicio perteneciente al acuerdo y al cual aplican los términos de garantía. Los términos de garantía especifican los niveles de servicios acordados por las partes participantes. Los sistemas de gestión pueden utilizar los términos de garantía para monitorear el servicio y hacer cumplir el acuerdo.

Dentro del elemento *Terms* se pueden utilizar elementos especiales para combinar términos. Esto permite la especificación de ramas alternativas con anidación potencialmente compleja de los términos del acuerdo. Debe existir al menos un término de servicio y cero o más términos de garantía. En la Figura 36 se describe la estructura recursiva de los términos.

```
<wsag:Terms>
    <wsag:All>
        {
            <wsag:All>
                wsag:TermCompositorType
            </wsag:All>       |
            <wsag:OneOrMore>
                wsag:TermCompositorType
            </wsag:OneOrMore> |
            <wsag:ExactlyOne>
                wsag:TermCompositorType
            </wsag:ExactlyOne> |
            <wsag:ServiceDescriptionTerm>
                wsag:ServiceDescriptionTermType
            </wsag:ServiceDescriptionTerm> |
            <wsag:ServiceReference>
                wsag:ServiceReferenceType
            </wsag:ServiceReference> |
            <wsag:ServiceProperties>
                wsag:ServicePropertiesType
            </wsag:ServiceProperties> |
            <wsag:GuaranteeTerm>
                wsag:GuaranteeTermType
            </wsag:GuaranteeTerm>
        } +
    </wsag:All>
</wsag:Terms>
```

Figura 36. Esquema de Terms de [28]

Los elementos *All*, *OneOrMore* y *ExactlyOne* son operadores lógicos del tipo AND, OR y XOR respectivamente, y son utilizados para agrupar términos de forma lógica.

Los términos de descripción de servicio (*ServiceDescriptionTerm* o SDT) son un componente fundamental de un acuerdo. La provisión de este servicio puede estar condicionada a restricciones en tiempo de ejecución y objetivos de niveles de servicio adicionales sobre cómo debe funcionar el servicio; los SDTs definen la funcionalidad que será distribuida bajo un acuerdo. El contenido de la descripción del servicio es dependiente de cada dominio particular. Un SDT consiste de tres partes:

- El nombre del término.
- El nombre del servicio que se está describiendo parcial o completamente.
- Una descripción específica de dominio de la funcionalidad ofrecida o requerida.

Una referencia de servicio (*ServiceReference*) apunta a un servicio, por ejemplo proveyendo una referencia a un extremo. Ambas partes entienden la semántica del servicio al que se está refiriendo, o conocen cómo buscar las propiedades del servicio.

El elemento ServiceProperties es utilizado para definir propiedades expuestas y mensurables asociadas al servicio, como por ejemplo tiempo de respuesta y rendimiento. Estas propiedades son utilizadas para expresar los objetivos de niveles de servicio.

Los términos de garantía (*GuaranteeTerm*) definen la garantía sobre calidad de servicio. Un acuerdo puede garantizar límites en la disponibilidad de recursos tales como memoria, CPU o almacenamiento. Estos límites son referidos como objetivos de niveles de servicios (SLO). Una expresión de garantía también puede incluir condiciones en factores externos, tales como la hora del día u otras condiciones que el consumidor del servicio debe cumplir. Por ejemplo, el tiempo promedio del tiempo de respuesta del servicio de un banco puede ser garantizado si la tasa de pedidos está dentro de un umbral especificado durante días de semana. Un término de garantía también incluye especificación de valores de negocio asociados a un SLO. Un acuerdo contiene cero o más términos de garantía, donde el elemento de cada término consiste de las siguientes partes:

- *Obligated*: la parte obligada a cumplir con el término.
- *ServiceScope*: la lista de servicios a la que aplica el término de garantía.
- *QualifyingCondition*: una condición opcional que se debe cumplir (cuando es especificada) para hacer cumplir con la garantía.
- *ServiceLevelObjective*: una aserción expresada sobre las descripciones de servicio.
- *BusinessValueList*: uno o más valores de negocio asociados con el objetivo.

9.8.WS-Security

La especificación WS-Security [36] propone un conjunto de extensiones SOAP que pueden ser utilizadas para asegurar la integridad y confidencialidad de los mensajes.

La especificación provee tres mecanismos principales: tokens de seguridad como parte del mensaje, integridad de mensajes y confidencialidad de mensajes. Un token de seguridad es un conjunto de declaraciones (*Claims*) hechas por una entidad, como por ejemplo nombre, identidad, clave, privilegios, etc. Las firmas son utilizadas para verificar el origen y la integridad de los mensajes, asegurando que cualquier modificación en los mensajes pueda ser detectada. Para mantener la confidencialidad de los mensajes SOAP se utilizan mecanismos de encriptación.

El formato y semántica de los tokens no están definidos en esta especificación, sino que son definidos en documentos de perfiles asociados. Por ejemplo, en el perfil Username Token [37] se especifica el formato de tokens utilizados para enviar un usuario y contraseña en los mensajes SOAP. En la Figura 37 se muestra un ejemplo de un mensaje SOAP con un token que especifica un nombre de usuario y contraseña.

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="...">
   <S11:Header>
      ...
      <wsse:Security>
         <wsse:UsernameToken>
            <wsse:Username>Zoe</wsse:Username>
            <wsse:Password>IloveDogs</wsse:Password>
         </wsse:UsernameToken>
      </wsse:Security>
      ...
   </S11:Header>
   ...
</S11:Envelope>
```

Figura 37. Ejemplo de UsernameToken de [37]

La especificación WS-SecurityPolicy [38] define un conjunto de aserciones de políticas de seguridad basadas en las características de seguridad de WS-Security y expresadas con el framework WS-Policy. La especificación define distintos tipos de aserciones: de integridad, de confidencialidad, de elementos requeridos, y de tokens de seguridad (por ejemplo Kerberos y X.509). En la Figura 38 se muestra un ejemplo de una política en la que se requiere que el emisor del mensaje envie la contraseña en un UsernameToken.

```
<wsp:Policy>
    <sp:SupportingTokens>
        <wsp:Policy>
            <sp:UsernameToken sp:IncludeToken=
            "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient"
        </wsp:Policy>
    </sp:SupportingTokens>
</wsp:Policy>
```

Figura 38. Ejemplo de aserción de UsernameToken

10. Plataforma Java EE

La plataforma Java EE es una colección de especificaciones que definen una infraestructura con componentes estandarizados y servicios para desarrollar aplicaciones distribuidas, en una arquitectura multicapa.

Java EE tiene como objetivo principal permitir que el desarrollador se centre en el diseño e implementación del sistema, delegando a la infraestructura del servidor de aplicaciones Java EE las cuestiones de más bajo nivel ajenas a la aplicación. Para nuestro proyecto utilizamos la versión Java EE 7.

Arquitectura JEE

La plataforma Java EE utiliza un modelo de aplicación distribuida de varios niveles. La lógica de aplicación se divide en distintos componentes según su función y los componentes de la aplicación que constituyen una aplicación Java EE son instalados en diferentes máquinas dependiendo del nivel del ambiente JEE al que pertenecen.

La Figura 39 muestra dos aplicaciones multinivel Java EE divididas en los niveles que se describen a continuación:

- Los componentes a nivel de cliente (*Client-tier*) se ejecutan en la máquina cliente.
- Los componentes de nivel Web (*Web-tier*) se ejecutan en el servidor Java EE.
- Los componentes de nivel de negocio (*Business-tier*) se ejecutan en el servidor Java EE.
- El sistema de información de la empresa (EIS, por su sigla en inglés) se ejecuta en el servidor EIS.

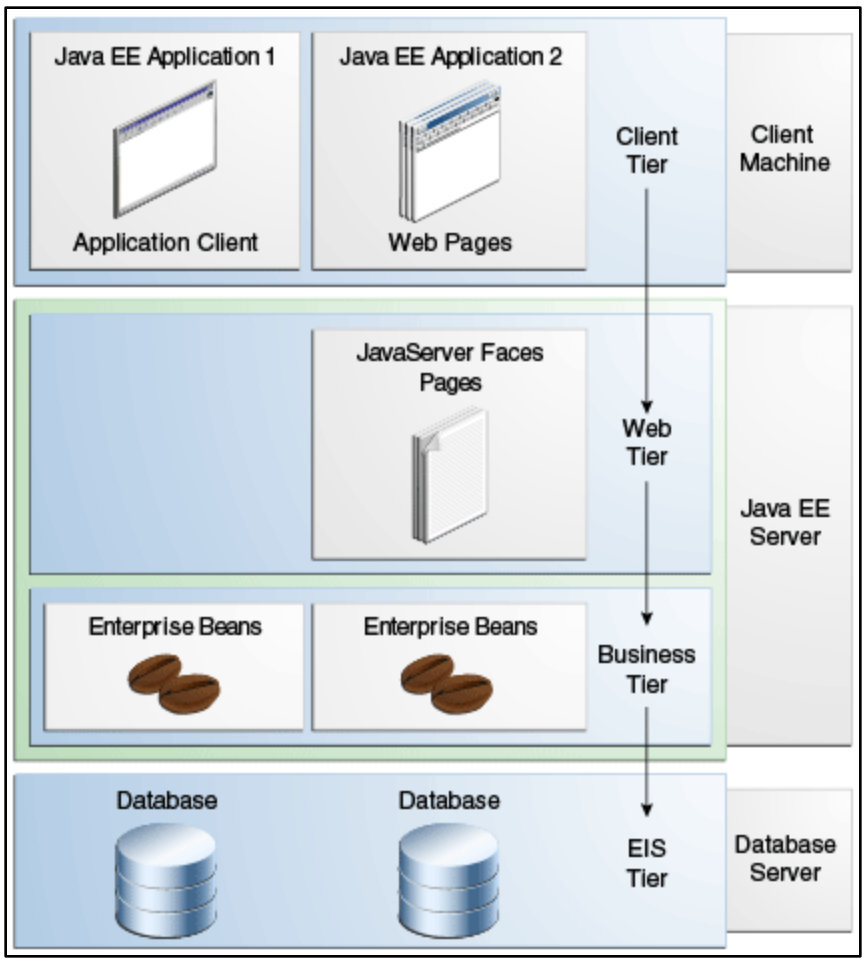

Figura 39. Aplicaciones multinivel de [39]

Una aplicación Java EE consiste de las capas de software mostradas en la Figura 39, y son consideradas generalmente como aplicación de tres niveles debido a que se distribuyen en tres lugares: las máquinas cliente, la máquina servidor Java EE y la base de datos o máquinas existentes en el extremo posterior.

11. Desarrollos previos

En este capítulo se describen los plugins de Eclipse realizados por proyectos de grado que trabajaron en SoaML y generación de código en años anteriores, y que utilizamos en este proyecto como base para el desarrollo del plugin integrado SoaML Toolkit.

11.1. Plugin SoaML para Eclipse

El plugin de Eclipse SoaML, realizado en el marco de un proyecto de grado en el año 2010, es un plugin que implementa el estándar SoaML (la versión beta 2 de la especificación) y permite la generación de diagramas de forma gráfica y la importación y exportación de los modelos generados en formato XMI (XML Metadata Interchange) para permitir la interoperabilidad con otras herramientas.

Actualmente existen muy pocas herramientas que ofrezcan el modelado de servicios con el estándar SoaML [17], la mayoría de las cuales son comerciales por lo que este plugin al encontrarse en el contexto de Eclipse y del proyecto Papyrus, provee a la comunidad una herramienta para el modelado de servicios con SoaML como soporte para desarrollos orientados a servicios, lo cual fue de gran aporte.

La solución que se implementó para la realización de este plugin extiende las funcionalidades del plugin Papyrus para el modelado UML, permitiendo de esta forma reutilizar tanto los diagramas que se implementaron para el mismo, como otras funcionalidades provistas a nivel de interfaz gráfica. Se aprovecharon también, ciertas facilidades que Papyrus ofrecía para la definición de editores de perfiles de UML y mecanismos de personalización de los mismos. El plugin SoaML provee la mayoría de los diagramas definidos por el estándar SoaML para el modelado de: Arquitectura de Servicios, Participantes, Servicios y su especificación mediante contratos de servicio, interfaces, operaciones con parámetros de entrada y salida. Puntualmente brinda siete diagramas: diagrama de Arquitectura de Servicios, diagrama de Contratos de Servicios, diagrama de Participantes como clases y como componentes, diagrama de Mensajes y diagrama de Capacidades. En la Figura 40 se puede ver el esquema de la arquitectura Eclipse con las extensiones para el plugin SoaML.

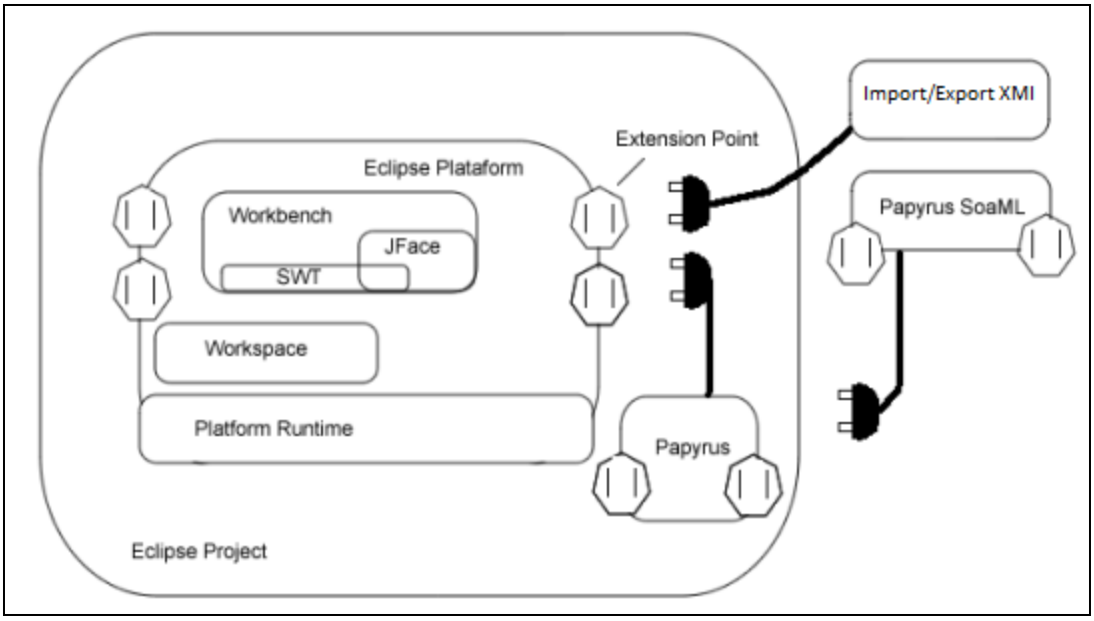

Figura 40. Esquema de la arquitectura Eclipse con las extensiones para el plugin SoaML

11.2. Plugin SoaML2Code para Eclipse

El plugin de Eclipse denominado SoaML2Code, que fue desarrollado en un proyecto de grado en el año 2012, es una extensión del plugin SoaML. Éste permite que a partir de un modelo SoaML representado por los diagramas realizados en el plugin SoaML, genere código ejecutable en lenguaje Java que represente dicho modelo, y exponga los servicios provistos en el modelo como Web Services.

La solución implementada utiliza el framework ya existente Apache CXF, el cual es un framework para servicios de código abierto que ayuda a construir y desarrollar servicios, incluyendo entre sus principales características el soporte a múltiples estándares de Web Services, además de soporte para una variedad de lenguajes de programación en los clientes y una gran variedad de protocolos de transporte.

El plugin toma en cuenta para la generación de código la mayoría de los elementos que forman parte del estándar SoaML y que son representados por los diagramas definidos por el plugin SoaML. Los servicios son expuestos como Web Services SOAP (Simple Object Access Protocol, SOAP) bajo la API de implementación Java para la creación de Web Services JAX- WS RI y JAX-WS + Spring, permitiendo a los usuarios de la comunidad agregar fácilmente alguna otra implementación que deseen.

La herramienta es de fácil utilización y al encontrarse en el contexto de Eclipse provee a la comunidad de un plugin para la generación de código ejecutable que permite la exposición de servicios a partir de modelos SoaML, funcionando correctamente de manera independiente de la versión de Eclipse que se esté utilizando y brindando una herramienta de distribución gratuita y de código abierto como soporte para desarrollos orientados a servicios. En la Figura 41 se puede observar cómo interactúa el plugin SoaML2Code con el plugin SoaML y Eclipse.

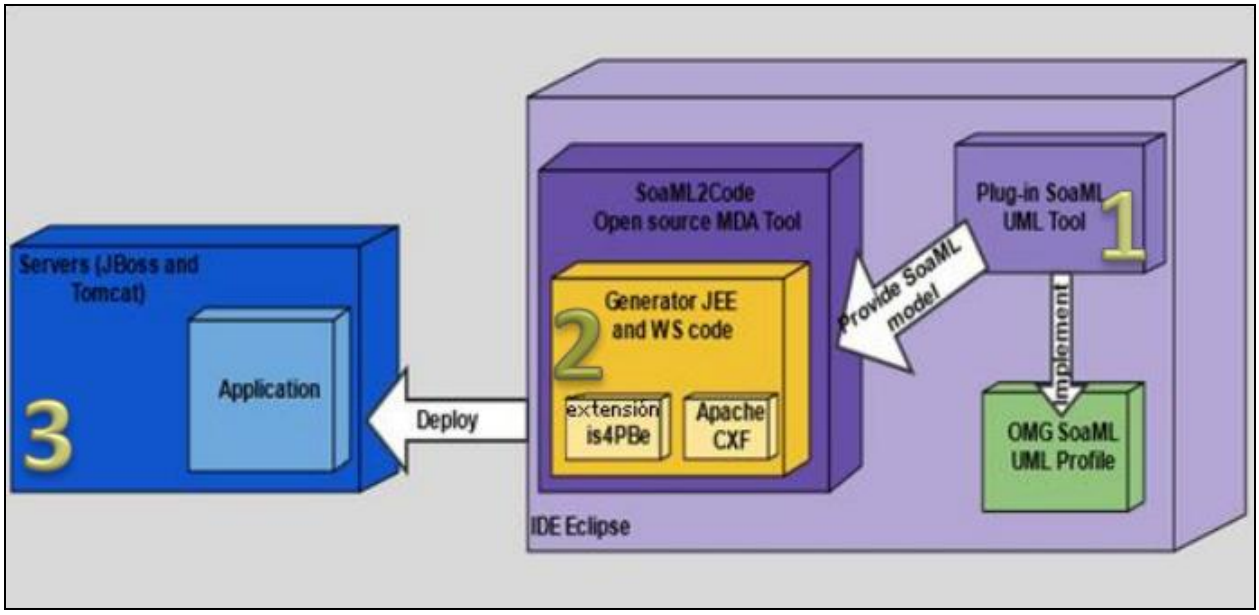

Figura 41. Interacción de componentes para la generación en plugin SoaML2Code

12. Referencias

[1] Stephen J. Mellor, Tony Clark, and Takao Futagami. Model-driven development: Guest editors' introduction. *IEEE software*, 20 (5): 14–18, 2003.

[2] Douglas C. Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39 (2): 25, 2006.

[3] Bran Selic. The pragmatics of model-driven development. *IEEE software*, 20 (5): 19–25, 2003.

[4] Colin Atkinson and Thomas Kuhne. Model-driven development: a metamodeling foundation. *Software, IEEE*, 20 (5): 36–41, 2003.

[5] Mariano Belaunde and Carol Burt. MDA guide. *Object Management Group, Tech. Rep. omg/2003-06-01*, 2003.

[6] Anneke G. Kleppe, Jos B. Warmer, and Wim Bast. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional, 2003.

[7] Juan Bernardo Quintero and Raquel Anaya. MDA y el papel de los modelos en el proceso de desarrollo de software. *Revista EIA*, (8), 2007.

[8] OMG Unified Modeling Language version 2.5, 2013.

[9] OMG Meta Object Facility (MOF) Core Specification version 2.4.2, 2014.

[10] Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, 2011.

[11] MOF Model to Text Transformation Language, 2008.

[12] XML Metadata Interchange Specification, 2014.

[13] Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall, 2005. ISBN 9780131858589.

[14] MP Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service oriented computing: State of the art and research challenges. *Computer*, 40 (11): 38–45, 2007.

[15] Mike P. Papazoglou and Willem-Jan Van Den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB journal*, 16 (3): 389–415, 2007.

[16] Grace A. Lewis, Edwin Morris, Soumya Simanta, and Lutz Wrage. Common misconceptions about service-oriented architecture. In *Commercial-off-the-Shelf (COTS)-Based Software Systems, 2007. ICCBSS'07. Sixth International IEEE Conference on*, pages 123–130. IEEE, 2007.

[17] Service oriented architecture Modeling Language (SoaML) Specification, v1.0.1, 2012.

[18] UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, v1.1, 2008.

[19] Niels Schot. QoS-aware model-driven SOA using SoaML. 2012.

[20] Liam O'Brien, Len Bass, and Paulo Merson. Quality attributes and service-oriented architectures. technical note. Technical report, CMU/SEI-2005-TN-014, 2005.

[21] Web Services Quality Factors Version 1.0, .

[22] OASIS, URL https://www.oasis-open.org/. [Último acceso: 07/12/2014]

[23] Quality Model for Web Services, 2005.

[24] Andrea Delgado, Barbara Weber, Francisco Ruiz, Ignacio García-Rodríguez de Guzmán, and Mario Piattini. An integrated approach based on execution measures for the continuous improvement of business processes realized by services. 2013.

[25] Andrea Delgado, Francisco Ruiz, Ignacio García-Rodríguez de Guzmán, and Mario Piattini. MINERVA: Model driven and sErvice oRiented framework for the continuous business processes improVement & relAtedtools. *5th Workshop on Engineering Service-Oriented Applications (WESOA'09) in 7th Int. Conf. on Service Oriented Computing (ICSOC'09)*, Noviembre 2009.

[26] Changzhou Wang, Guijun Wang, Haiqin Wang, Alice Chen, and Rodolfo Santiago. Quality of service (QoS) contract specification, establishment, and monitoring for service level management. In *Enterprise Distributed Object Computing Conference Workshops, 2006. EDOCW'06. 10th IEEE International*, pages 49–49. IEEE, 2006.

[27] A. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, and Ü Yalçinalp. Web Services Policy 1.5. *Framework*, 2007.

[28] Alain Andrieux and Karl Czajkowski. Web services agreement specification (WS-Agreement).

[29] *Eclipse Luna Documentation*. URL http://help.eclipse.org/luna/index.jsp. [Último acceso: 26/11/2014]

[30] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture, W3C Working Group Note, 11 February 2004. *World Wide Web Consortium*, 2004. URL http://www.w3.org/TR/ws-arch. [Último acceso: 30/11/2014]

[31] Sameer Tyagi. RESTful Web Services. 2006. URL http://www.oracle.com/technetwork/articles/-javase/index-137171.html. [Último acceso: 26/11/2014]

[32] Marc Hadley, Noah Mendelsohn, J. Moreau, H. Nielsen, and M. Gudgin. SOAP Version 1.2 Part 1: Messaging Framework. *W3C REC REC-soap12-part1-20030624, June*, pages 240–8491, 2003.

[33] Thomas Erl, Anish Karmarkar, Priscilla Walmsley, Hugo Haas, L. Umit Yalcinalp, Kevin Liu, David Orchard, Andre Tost, and James Pasley. *Web service contract design and versioning for SOA*. Prentice Hall, 2009.

[34] LINS/FING/UDELAR. Web services. 2012. URL http://www.fing.edu.uy/inco/cursos/tsi/TSI2/-2012/teorico/tsi2-07-web-services.pdf. [Último acceso: 26/11/2014]

[35] A D'Ambrogio. A Model-driven WSDL Extension for Describing the QoS of Web Services. In *Web Services, 2006. ICWS '06. International Conference on*, pages 789–796, Sept 2006.

[36] Web Services Security 1.1, 2004.

[37] Web Services Security UsernameToken Profile 1.1, 2006.

[38] WS-SecurityPolicy 1.2, 2007.

[39] Oracle. Distributed multitiered applications. URL http://docs.oracle.com/javaee/7/tutorial/doc/-overview003.htm. [Último acceso: 29/09/2014]

Proyecto de grado

Generación automática de Arquitecturas Orientadas a Servicios (SOAs) con SoaML y especificación de QoS

Guía de Instalación

Federico Bertolini, Sofía Pérez, María Noel Quiñones

Tutor
Andrea Delgado

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay
Diciembre 2014

Contenido

1. JDK

Descargar la JDK versión 1.7.x de la siguiente dirección: http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html.

2. Eclipse

Descargar el Eclipse Modeling Tools, en su versión Luna (4.4.0), de la siguiente dirección: https://www.eclipse.org/downloads/packages/eclipse-modeling-tools/lunasr1

3. Papyrus

Instalar Papyrus yendo al menú *Help* y eligiendo la opción *Install Modeling Components*, para luego seleccionar Papyrus.
Descargar el código fuente de http://git.eclipse.org/c/papyrus/org.eclipse.papyrus.git/tree/plugins/editor/org.eclipse.papyrus.editor.perspectiveconfiguration, compilar y generar el plugin, el cual deberá ser copiado en la carpeta plugins del Eclipse.

4. Plugin SoaML Toolkit

El plugin SoaML Toolkit engloba los siguientes plugins:

- SoaML Modelling Tool
- QoS Modelling Tool
- SoaML+QoS Modelling Tool
- Import/Export XMI
- SoaML2Code

Dentro de los entregables se encuentra el directorio *SoaMLToolkit_update*, que contiene lo necesario para instalar el plugin desde Eclipse. Para esto se debe ir al menú *Help*, seleccionar la opción *Install New Software* y agregar luego el directorio mencionado. Allí se mostrará el plugin, el cual deberá ser seleccionado para instalarlo, siguiendo los pasos del *wizard*.

Dentro de los entregables también se encuentran los directorios SoaMLModelling_update, SoaMLQoSModelling_update, SoaMLToJava_update, ImportExportXMI_update y QoSModelling_update, los cuales permiten instalar los plugins mencionados anteriormente de forma independiente.

Otra forma de instalación es copiar los *jars* que corresponden al plugin dentro de la carpeta *plugins* del Eclipse (con el Eclipse cerrado).

A continuación se enumeran los archivos que deben ser copiados por plugin:

SoaML Modelling Tool

- org.eclipse.papyrus.soaml.diagram.capabilities.jar

- org.eclipse.papyrus.soaml.diagram.common.jar
- org.eclipse.papyrus.soaml.diagram.interfacesdef.jar
- org.eclipse.papyrus.soaml.diagram.message.jar
- org.eclipse.papyrus.soaml.diagram.participantcomp.jar
- org.eclipse.papyrus.soaml.diagram.pl.jar
- org.eclipse.papyrus.soaml.diagram.sa.jar
- org.eclipse.papyrus.soaml.diagram.servicecontract.jar

QoS Modelling Tool

- org.eclipse.papyrus.qos.diagram.common.jar
- org.eclipse.papyrus.qos.diagram.qoscharacteristic.jar

SoaML+QoS Modelling Tool

- org.eclipse.papyrus.soamlqos.diagram.capabilities.jar
- org.eclipse.papyrus.soamlqos.diagram.common.jar
- org.eclipse.papyrus.soamlqos.diagram.interfacesdef.jar
- org.eclipse.papyrus.soamlqos.diagram.message.jar
- org.eclipse.papyrus.soamlqos.diagram.participantcomp.jar
- org.eclipse.papyrus.soamlqos.diagram.pl.jar
- org.eclipse.papyrus.soamlqos.diagram.qoscharacteristic.jar
- org.eclipse.papyrus.soamlqos.diagram.sa.jar
- org.eclipse.papyrus.soamlqos.diagram.servicecontract.jar

Import/Export XMI

- importexportXMI.jar

SoaML2Code

- SoaMLToJava.jar.

Para desinstalar los plugins, si se instalaron de la primera forma mencionada, se debe ir al menú *Help* y seleccionar la opción *Installation Details*. Allí se desplegarán todos los plugins instalados, por lo que se deben seleccionar los que se desea desinstalar, luego presionar *Uninstall* y seguir los pasos. Si se instalaron de la segunda forma mencionada entonces, con el Eclipse cerrado, se deben eliminar de la carpeta plugins todos los *jars* copiados en la instalación.

5. Servidores

Se deben instalar y configurar los servidores JBoss y Tomcat.

5.1.JBoss

Descargar JBoss versión 7.1.1 de http://jbossas.jboss.org/downloads.html (JBoss AS 7.1.1.Final).
Para configurarlo se debe instalar primero la herramienta *JBossAS Tools* yendo al menú *Help* e *Install New Software*, utilizando la dirección http://download.jboss.org/jbosstools/updates/release/luna/. Luego ir a la opción *Window* del menú y seleccionar *Preferences.* Dentro de la opción *Server/Runtime*

Environments hacer click en *Add* y seleccionar *JBoss 7.1 Runtime* para luego definir la ruta en donde fue instalado. Una vez hecho esto, se debe ir a la vista *Servers*, hacer click derecho y *New/Server* y seleccionar el servidor configurado anteriormente.

5.2.Tomcat

Descargar Tomcat 8.0 de http://tomcat.apache.org/download-80.cgi.
Ir al menú *Help/Install New Software* y descargar de http://download.eclipse.org/releases/luna/ los plugins *JST Server Adapters* y *JST Server Adapters Extentions*.
Para configurarlo sólo es necesario ir a la vista *Servers*, hacer click derecho y *New/Server* y seleccionar la opción *Tomcat v8.0 Server* para luego especificar el directorio donde se encuentra instalado.

6. Generación de Dynamic Web Project

6.1.JAX-WS RI

Descargar JAX-WS RI, versión 2.2.8, de https://jax-ws.java.net/2.2.8/.
Descomprimir el archivo descargado y copiar la carpeta en la carpeta plugins del Eclipse. La carpeta copiada deberá mantener el nombre "jaxws-ri".

6.1.1. JBoss

Para que compilen los proyectos generados para JBoss se debe definir la variable JBOSS_HOME apuntando al directorio de instalación de JBoss, haciendo click en *Window/Preferences* y luego en *Java/Build Path/Classpath Variables*.

6.2.Apache CXF

Descargar Apache CXF 3.0.1 de http://archive.apache.org/dist/cxf/3.0.1/.
En el Eclipse se debe setear la variable de definición de CXF yendo al menú *Window/Preferences* y luego seleccionando *Web Services/CXF 2.x Preferences.* Luego se debe hacer click en *Add* para ingresar la ubicación de la carpeta descargada.
Instalar plugin *CXF Web Services* yendo a *Help/Install New Software.*

7. Plugins Extra

Además de los plugins instalados anteriormente es necesario instalar los siguientes:

- Eclipse Java EE Developer Tools
- Eclipse Java Web Development Tools

Ambos se instalan yendo a *Help/Install New Software* y utilizando la dirección http://download.eclipse.org/releases/luna/.

Proyecto de grado

Generación automática de Arquitecturas Orientadas a Servicios (SOAs) con SoaML y especificación de QoS

Manual de Usuario

Federico Bertolini, Sofía Pérez, María Noel Quiñones

Tutor
Andrea Delgado

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay
Diciembre 2014

Resumen

Este documento tiene como propósito explicar cómo se trabaja correctamente con el plugin SoaML Toolkit.

Contenido

1. Descripción de los diagramas disponibles

A continuación se describirán los diagramas disponibles incorporados al plugin de modelado con la inclusión de QoS. Para ver el resto de los diagramas, referirse al Manual de Usuario del plugin SoaML.

1.1.Diagrama de Características QoS

Este diagrama permite definir las distintas características de calidad, las dimensiones que cuantifican dichas características así como también la categoría a la cual pertenece y las asociaciones que pueden haber entre distintas características. Se podrá agregar elementos tales como características, dimensiones y categorías así como también asociaciones y comentarios.

1.2.Diagrama de Contrato de Servicios con QoS

En este diagrama podemos construir contratos de servicios. Se pueden agregar elementos tales como contratos de servicios, roles y usos de contratos de servicios. También se pueden agregar asociaciones entre roles y especificar los roles que cumplen ciertos roles en contratos de servicios asociados. El usuario también puede agregar características y valores de calidad de servicios, y asociaciones entre valores de calidad y contratos de servicios.

1.3.Diagrama de Participantes como Clases con QoS

En este diagrama se pueden especificar participantes, agregando elementos de tipo Participant y puertos de tipo Service y Request. El usuario también puede agregar características y valores de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios.

1.4.Diagrama de Participantes como Componentes con QoS

Permite al usuario crear un diagrama que indica la implementación de participantes como componentes. Se pueden agregar componentes participante y dentro de ellos elementos tales como proveedores de servicios, puertos, asociaciones y canales de servicio entre los puertos, capacidades y sus dependencias con los participantes existentes en el diagrama, usos de arquitecturas de servicios y los correspondientes rolebindings con los proveedores de servicios asociados. El usuario también puede agregar características y valores de calidad de servicios, así como también restricciones de calidad de servicios y asociarlos a puertos de servicios.

2. Pasos para construir una taxonomía de características de calidad

Desde el explorador, crear un nuevo proyecto *Papyrus*, elegir un nombre para el proyecto y seleccionar el lenguaje QoS, como se muestra en las Figuras 1 y 2.

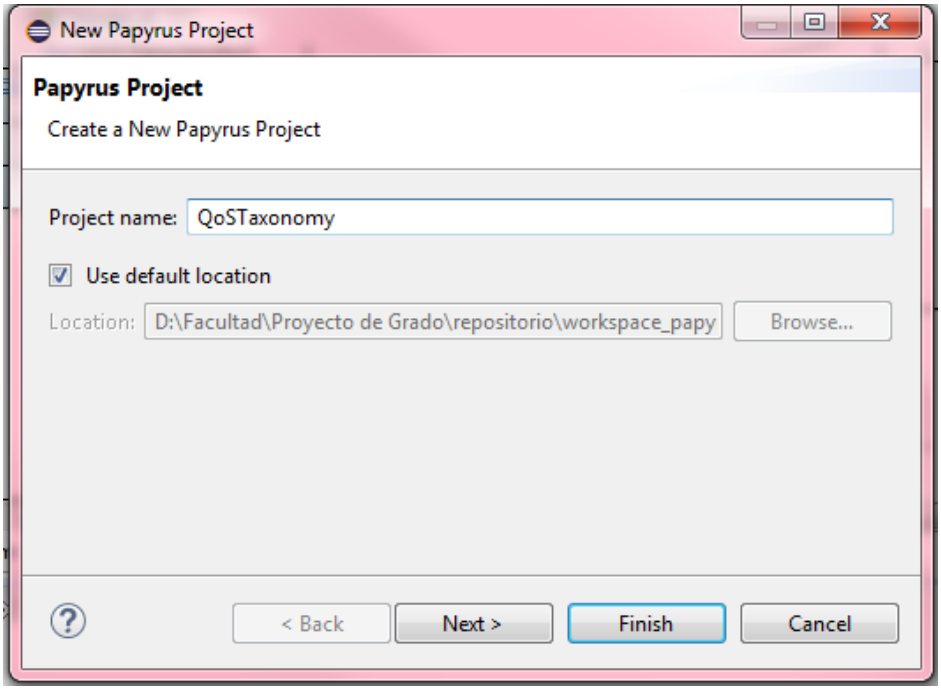

Figura 1. Proyecto Papyrus QoS - Paso 1

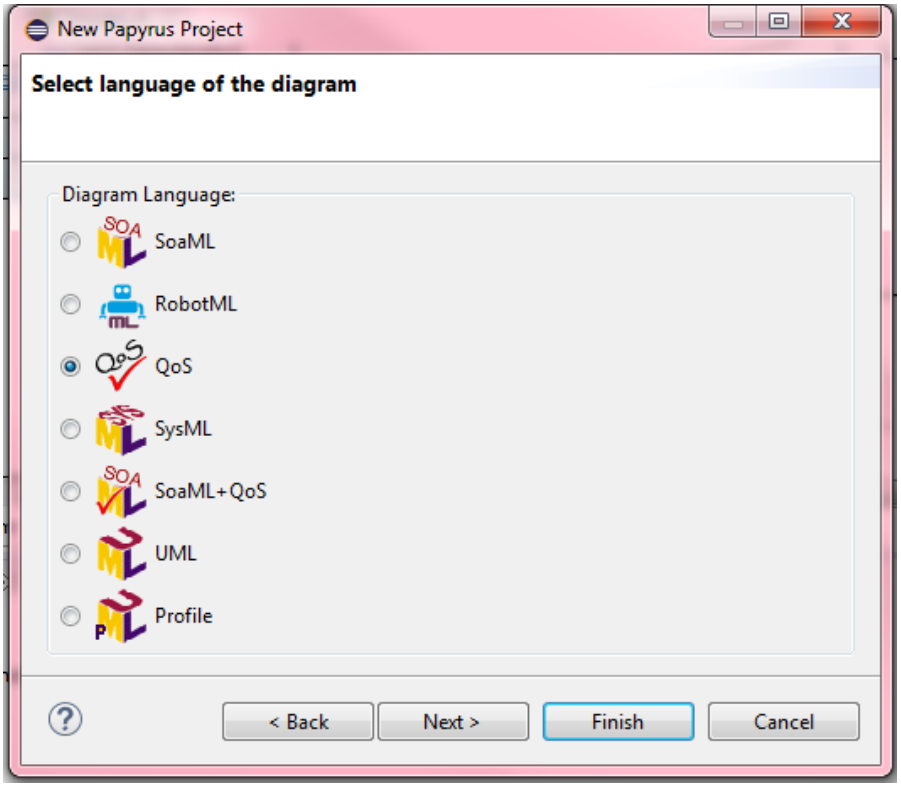

Figura 2. Proyecto Papyrus QoS - Paso 2

A continuación se seleccionan los diagramas que se desean generar y en este caso el único disponible es el de QoS Characteristics. Esto se puede ver en la Figura 3.

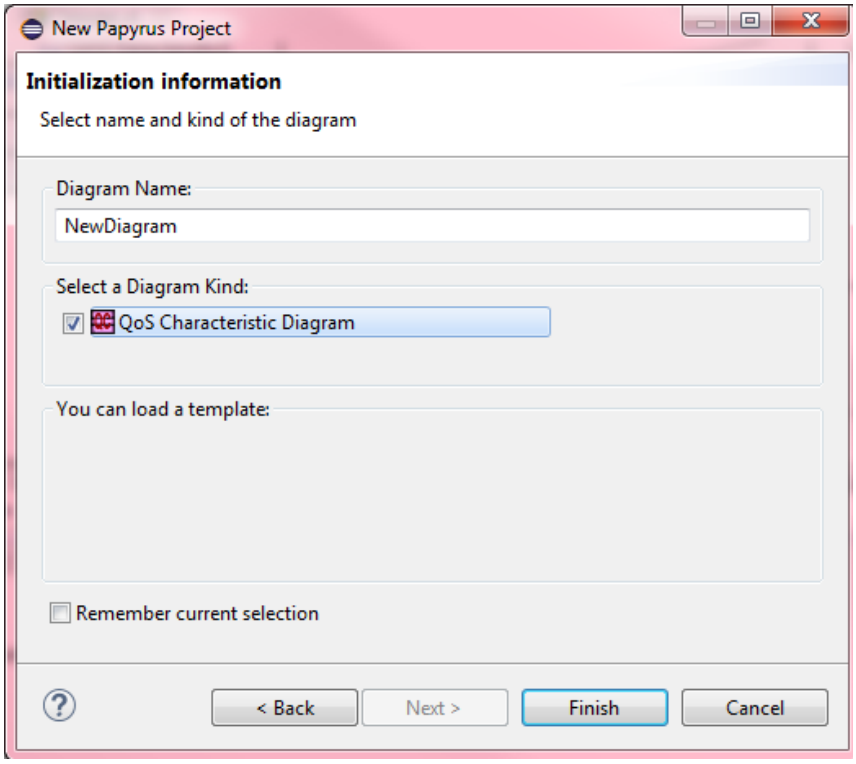

Figura 3. Proyecto Papyrus QoS - Paso 2

Una vez finalizado, se creará el diagrama seleccionado en el modelo recientemente creado y se aplicará por defecto al modelo el perfil QoS. El asistente de creación preguntará si desea cambiar a la perspectiva Papyrus. Se recomienda aceptar esta sugerencia.

Teniendo el proyecto creado, en todo momento se pueden agregar al modelo cuantos diagramas de QoS Characteristic se desee. Esto se logra haciendo *click* derecho sobre el modelo en la vista *Model Explorer* y seleccionando la opción *New diagram*, como se muestra en la Figura 4.

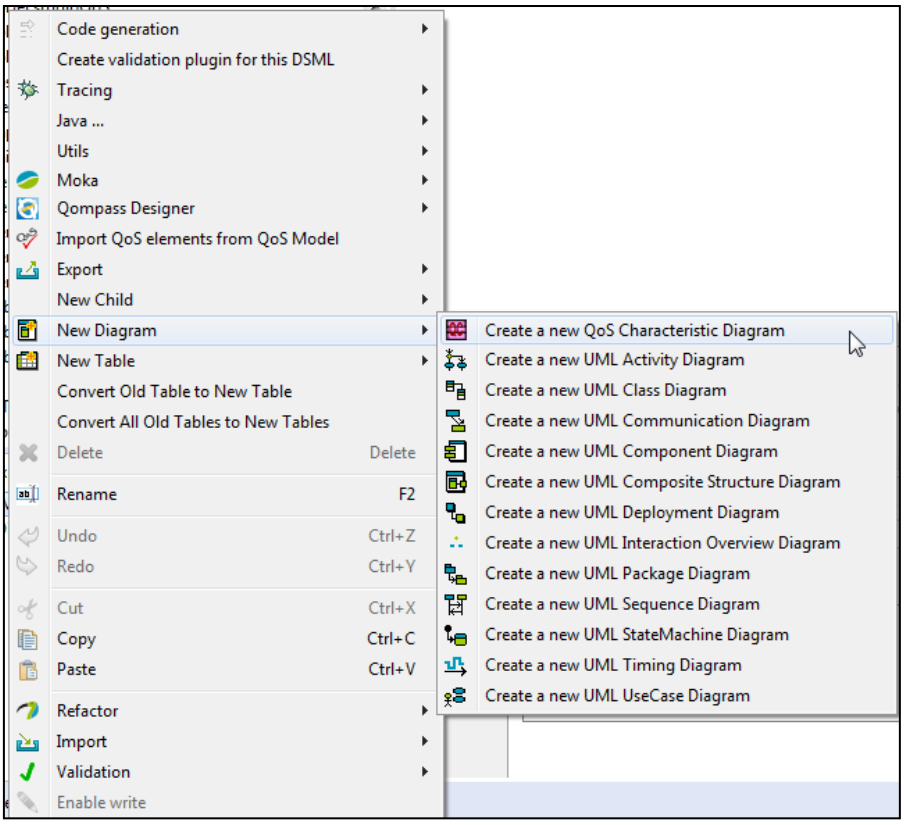

Figura 4. Crear nuevo diagrama en el modelo QoS

2.1.Diagrama de Características QoS

Desde la paleta de herramientas del editor seleccione el elemento *QoSCategory*, haga *click* en el elemento y luego *click* en el lienzo. Para cambiar el nombre por defecto del elemento haga *click* en éste o seleccione el elemento *QoSCategory* en el lienzo, vaya a la pestaña UML de la vista *Properties* y modifique el campo *Name*. Si dicha vista no está visible entonces debe hacer *click* derecho sobre el elemento y seleccionar *Show Properties View*. Una vez insertado el elemento sobre el lienzo es posible modificar su tamaño.

Dentro del elemento *QoSCategory* es posible insertar elementos del tipo *QoSCharacteristic*. Para esto debe hacer *click* sobre el elemento en la paleta de herramientas y luego *click* dentro de la *QoSCategory* agregada anteriormente al lienzo. Esto permite agrupar características de calidad según distintos criterios. De todos modos, es posible insertar elementos del tipo *QoSCharacteristic* directamente sobre el lienzo, sin agruparlos dentro de una *QoSCategory*. Del mismo modo que para la *QoSCategory* es posible cambiar el nombre de la *QoSCharacteristic* y también es posible modificar su tamaño.

Dentro del elemento *QoSCharacteristic* es posible agregar tantos elementos *QoSDimension* como se desee. Para esto se debe seleccionar el elemento *QoSDimension* de la paleta de herramientas y luego hacer *click* sobre la *QoSCharacteristic* sobre la que se quiere agregar la dimensión. Inicialmente ésta es insertada con un nombre por defecto (que puede ser modificado al igual que en los otros casos) y con un tipo seteado como *<Undefined>*. Dado que la *QoSDimension* no tiene sentido sin un tipo definido, es necesario modificarlo. Para esto debe ir a la vista de propiedades y dentro de la pestaña UML ir al campo *Type.* Allí deberán

hacer click sobre el botón con la etiqueta "...", la cual permite seleccionar un tipo definido en el propio modelo o en paquetes de tipos, como se muestra en la Figura 5.

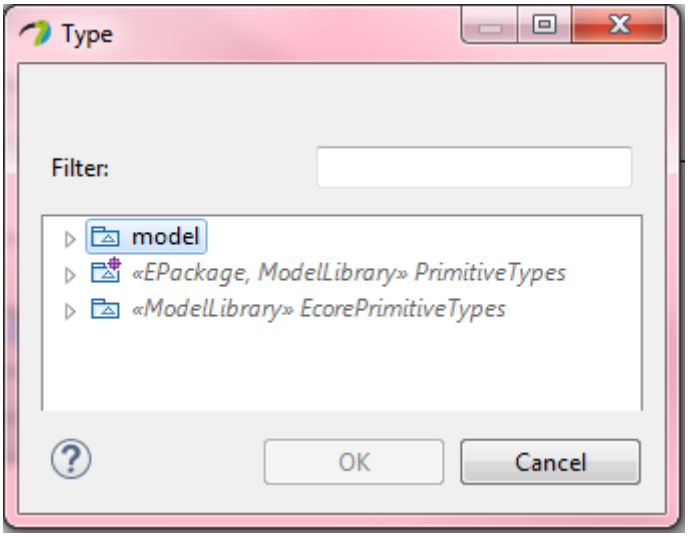

Figura 5. Selección de tipo

Dentro de este diagrama también es posible agregar comentarios. Para esto se debe seleccionar el elemento *Comment* dentro de la paleta de herramientas y luego hacer *click* sobre el lienzo. Los comentarios pueden ser enlazados a los demás elementos mediante el uso del conector *Link*. Para realizarlo se debe seleccionar el elemento *Link* de la paleta y luego hacer click, primero sobre el comentario y después sobre el elemento al que se quiere asociar el comentario.

También es posible conectar dos categorías mediante el conector *PackageImport*. Para esto es necesario hacer *click* sobre el elemento *PackageImport* en la paleta y luego hacer *click*, primero sobre la categoría A y luego sobre la B. Esto implicaría que la categoría A importa los elementos contenidos en la categoría B.

Por último, es posible agregar una generalización entre dos elementos del tipo *QoSCharacteristic*. Para esto es necesario hacer *click* sobre el elemento *Generalization* en la paleta y luego hacer *click*, primero sobre la característica A y luego sobre la B. Esto implicaría que la característica A hereda de la característica B.

3. Pasos para construir una arquitectura de servicios en SoaML con QoS

Desde el explorador, crear un nuevo proyecto *Papyrus*, elegir un nombre para el proyecto y seleccionar el lenguaje SoaML+QoS, como se muestra en las Figuras 6 y 7.

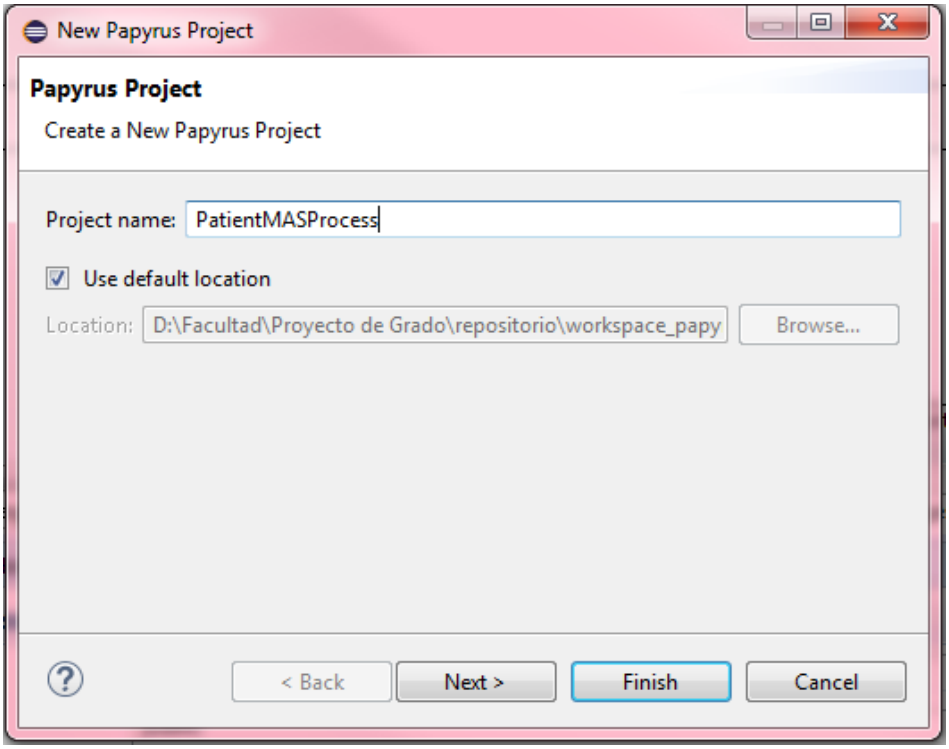

Figura 6. Proyecto Papyrus SoaML+QoS - Paso 1

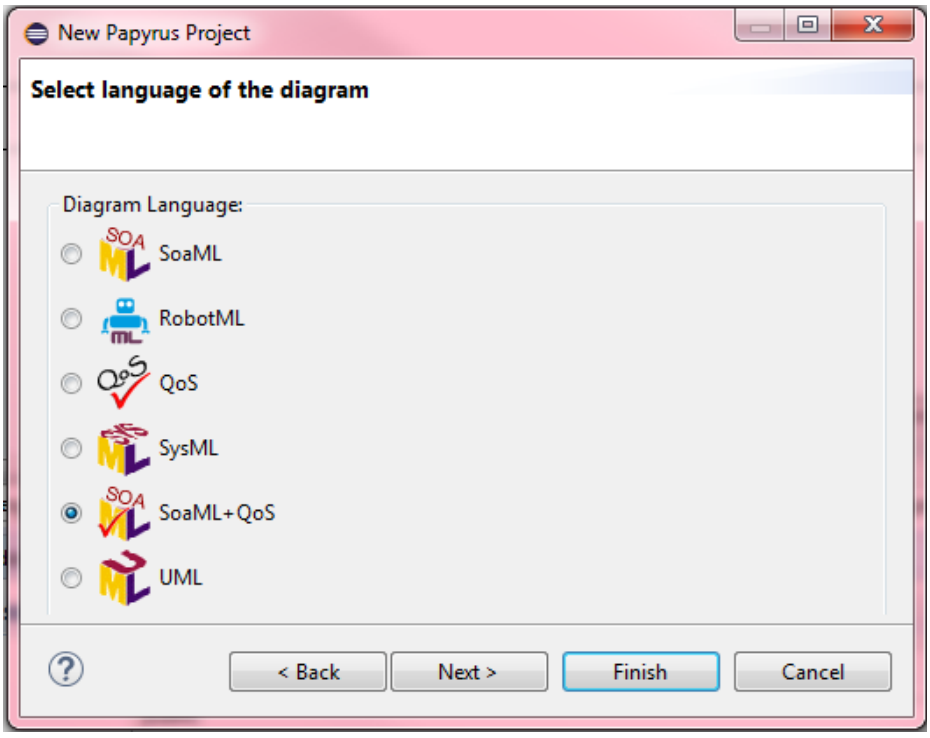

Figura 7. Proyecto Papyrus SoaML+QoS - Paso 2

A continuación se seleccionan los diagramas que se desean generar. Para el propósito de este manual se seleccionarán los diagramas incorporados en este plugin. Nuevamente, para ver el resto de los diagramas, referirse al Manual de Usuario del plugin SoaML. En la Figura 8 se puede ver la selección de los diagramas QoS Characteristic, Participant Class con QoS, Participant Component con QoS y Service Contract con QoS.

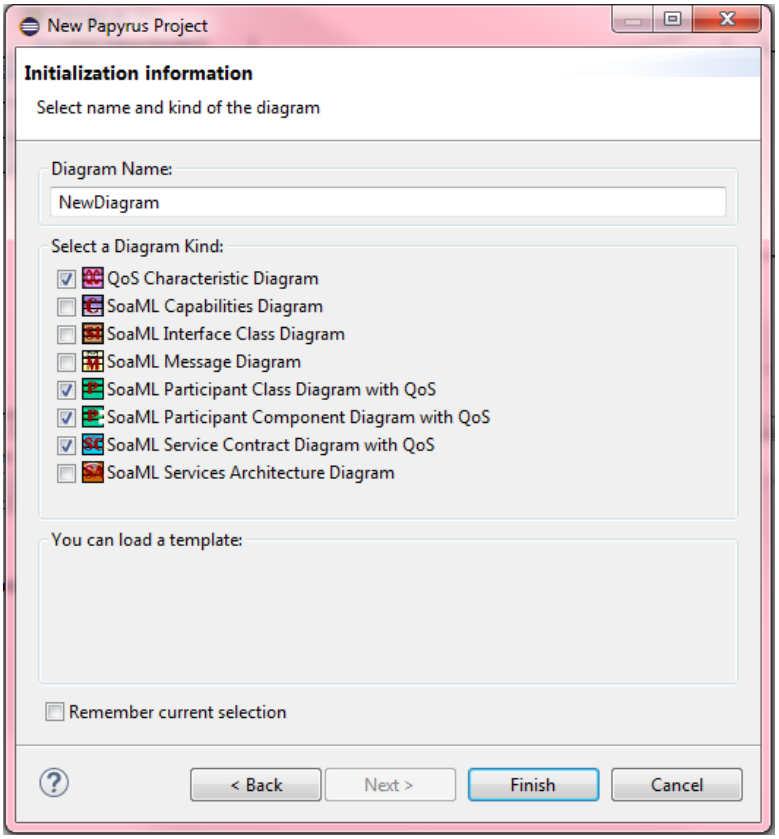

Figura 8. Proyecto Papyrus SoaML+QoS - Paso 3

Una vez finalizado, se crearán los diagramas seleccionados en el modelo recientemente creado y se aplicará por defecto al modelo el perfil SoaMLQoS. El asistente de creación preguntará si desea cambiar a la perspectiva Papyrus. Se recomienda aceptar esta sugerencia.

Teniendo el proyecto creado, en todo momento se pueden agregar nuevos diagramas al modelo. Esto se logra haciendo *click* derecho sobre el modelo en la vista *Model Explorer* y seleccionando la opción *New diagram*, como se muestra en la Figura 9.

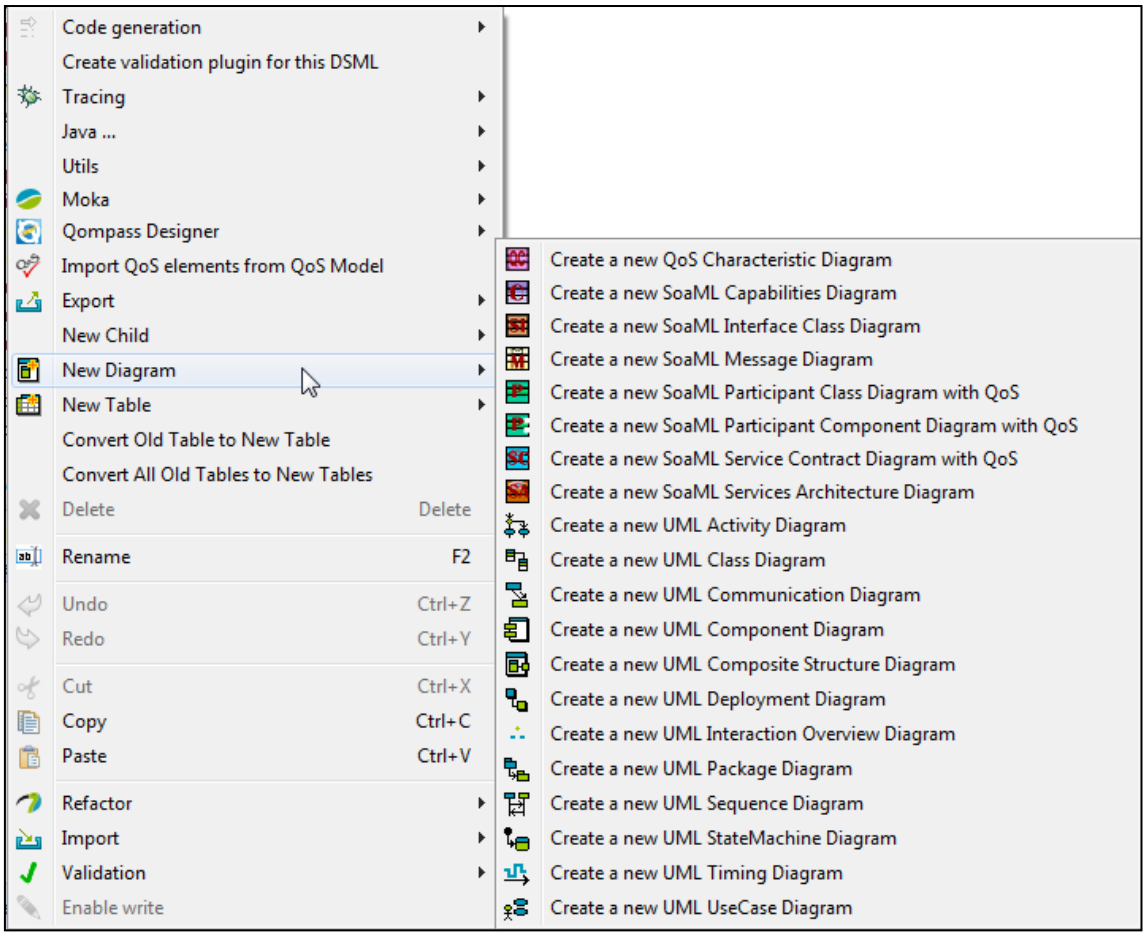

Figura 9. Crear nuevo diagrama en el modelo

3.1.Diagrama de Características QoS

Hay dos posibilidades a la hora de modelar las características de calidad en un modelo SoaML+QoS. Una forma es crear el diagrama de características de calidad siguiendo los pasos descriptos en la sección 2.1. De esta manera se tiene una taxonomía de características de calidad asociada directamente al modelo SoaML, la cual sólo podrá ser utilizada en el modelo actual. La otra forma es importando los elementos QoS de un modelo QoS definido de forma independiente, como se explicó en la sección 2. Esto tiene la gran ventaja de que se puede definir una taxonomía de características de calidad que podrá ser reutilizada en todos los modelos SoaML+QoS que se desee. Para realizar la importación se debe hacer *click* derecho sobre el modelo en la vista *Model Explorer* y seleccionar la opción *Import QoS elements from QoS Model*, como se muestra en la Figura 10.

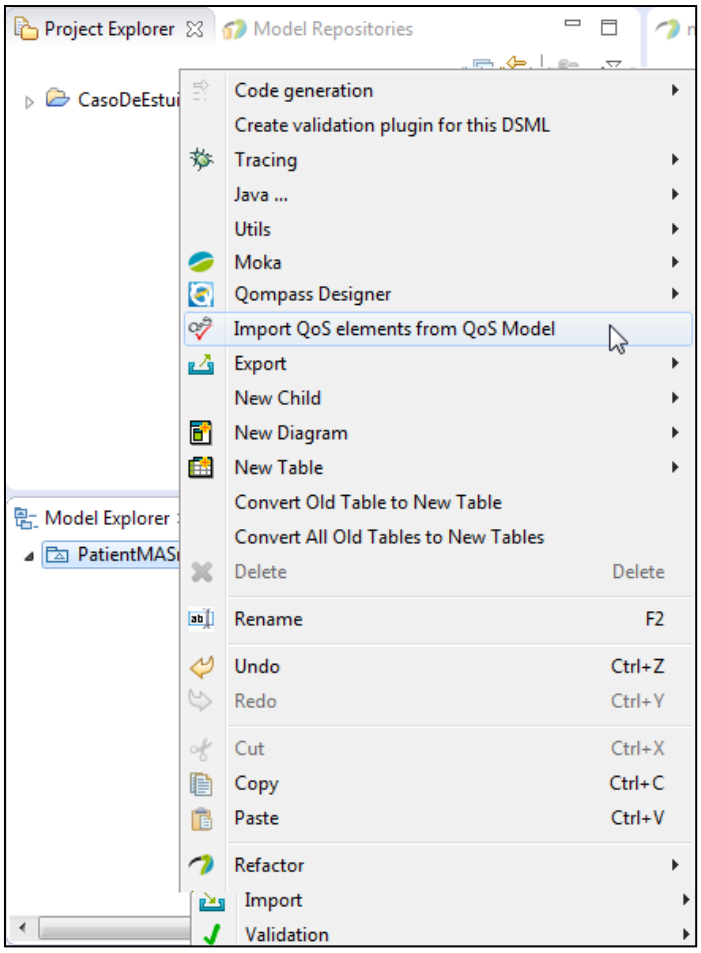

Figura 10. Importar elementos QoS

Cabe aclarar que esta funcionalidad sólo importa los elementos definidos en el modelo QoS y no los diagramas. Si se desee reproducir el diagrama se deben arrastrar los elementos desde la vista *Model Explorer* al lienzo.

3.2.Diagrama de Contrato de Servicios con QoS

Una vez seguidos los pasos del manual de usuario de SoaML para la creación del diagrama de contrato de servicios y sus elementos correspondientes, se pasa a explicar la utilización de los elementos de características de calidad en este diagrama.

A esta altura se deben tener definidas características de calidad, ya sea en un diagrama creado en el propio modelo o habiendo importado los elementos de un modelo QoS.

En este diagrama se podrán agregar elementos *QoSValue* que podrán ser asociados a elementos *ServiceContract*. Para esto, desde la paleta de herramientas del editor seleccione el elemento *QoSValue*, haga *click* en el elemento y luego *click* en el lienzo. Para cambiar el nombre por defecto del elemento haga *click* en éste o seleccione el elemento *QoSValue* en el lienzo, vaya a la pestaña UML de la vista *Properties* y modifique el campo *Name.* Una vez insertado el elemento sobre el lienzo es posible modificar su tamaño. El elemento *QoSValue* debe instanciar un elemento *QoSCharacteristic*. Para esto existen dos posibilidades. Para la primera opción se debe seleccionar el elemento *QoSValue* e ir a la pestaña UML de la vista *Properties*. Allí se debe hacer *click* sobre el botón con la etiqueta "+" en el campo *Classifier* y se

desplegará un cuadro de diálogo donde se deberá elegir el elemento *QoSCharacteristic* a instanciar, como se muestra en la Figura 11.

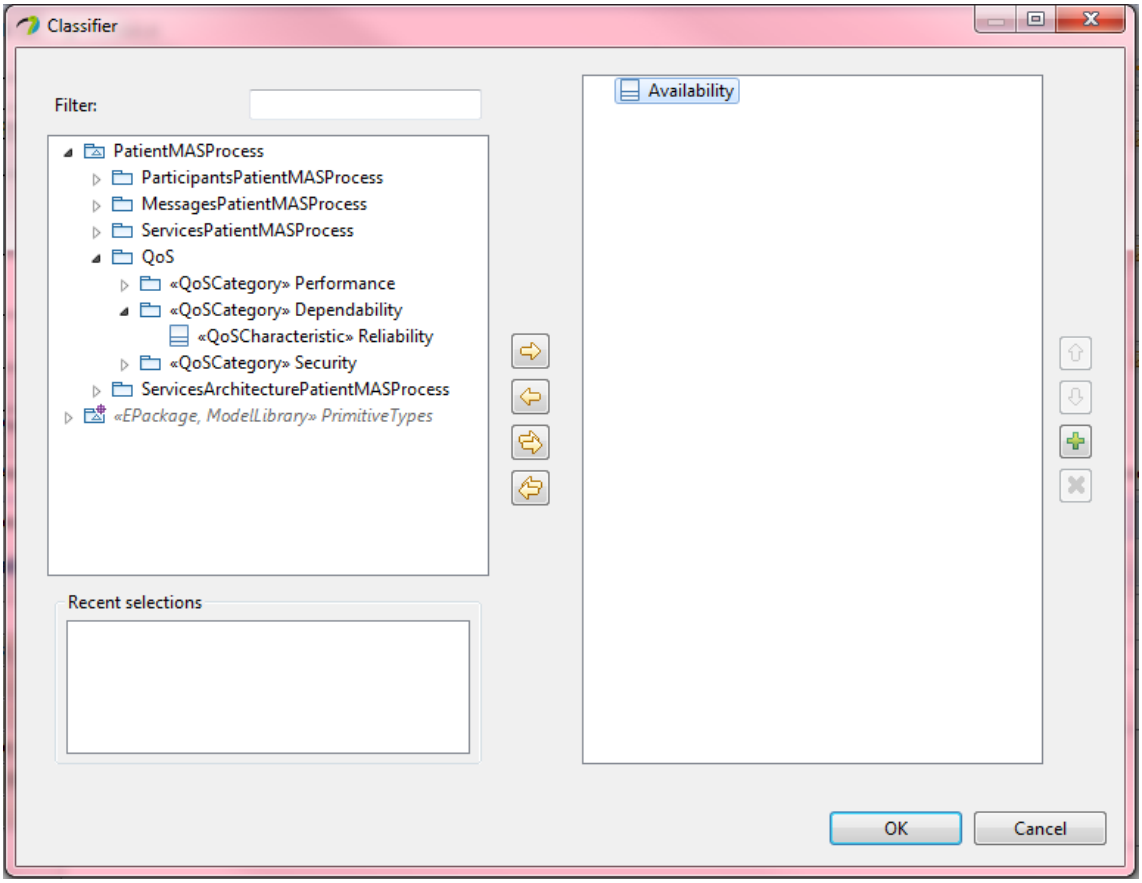

Figura 11. Selección de classifier QoSCharacteristic

Luego, se deberá agregar un elemento *Slot* por cada elemento *QoSDimension* de la *QoSCharacteristic* que se desee instanciar. Esto se realiza haciendo *click* sobre el elemento *Slot* en la paleta y luego *click* sobre la *QoSValue* previamente definida. Así, se agrega un slot con la etiqueta <UNDEFINED>, la cual debe ser definida. Para esto se debe hacer *click* sobre el slot, ir a la pestaña UML de la vista *Properties* y hacer *click* sobre el botón con la etiqueta "...". Allí se desplegará el cuadro de diálogo que se muestra en la Figura 12, en donde se deberá seleccionar la *QoSDimension*, que pertenece a la *QoSCharacteristic* seleccionada previamente, que instanciará el *Slot*.

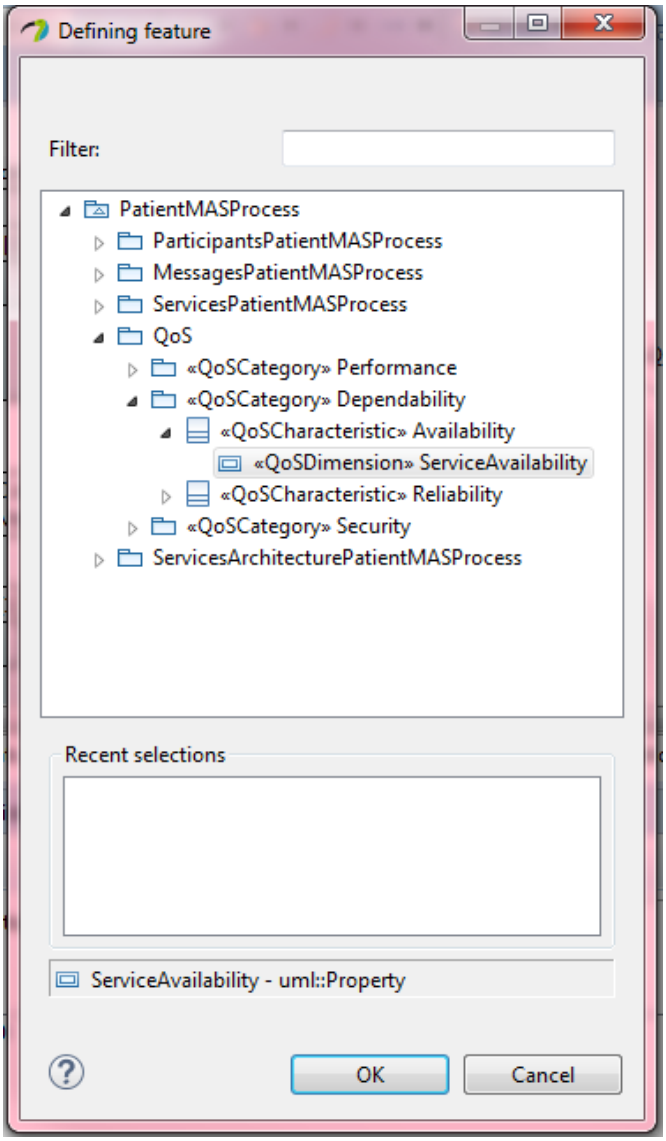

Figura 12. Selección de defining feature QoSDimension

Por último, se deberá asignar un valor a la *QoSDimension*. Para esto se debe nuevamente hacer *click* sobre el slot, ir a la pestaña UML de la vista *Properties* y hacer *click* sobre el botón "+" correspondiente al campo *Value*. Esto desplegará un menú en donde se deberá seleccionar el tipo del valor, como se muestra en la Figura 13.

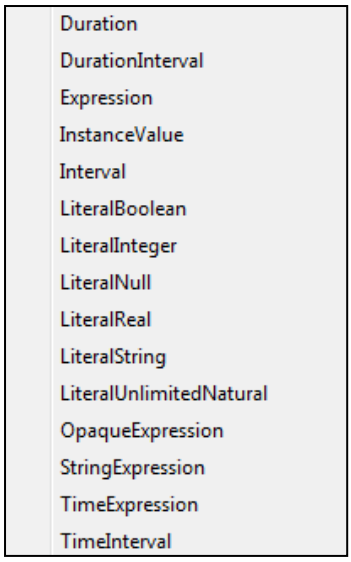

Figura 13. Selección de tipo de QoSDimension value

El tipo seleccionado dependerá del tipo definido para la *QoSDimension* en el diagrama de *QoS Characteristics*. Por ejemplo, si el tipo seleccionado fue *String*, entonces en el menú se deberá seleccionar *LiteralString*. Si en cambio el tipo seleccionado fue *Integer*, entonces en el menú se deberá seleccionar *LiteralInteger*.

Una vez seleccionado el tipo del *Slot*, se desplegará un cuadro de diálogo donde se deberá ingresar el valor para éste y opcionalmente el nombre. Esto se puede observar en la Figura 14.

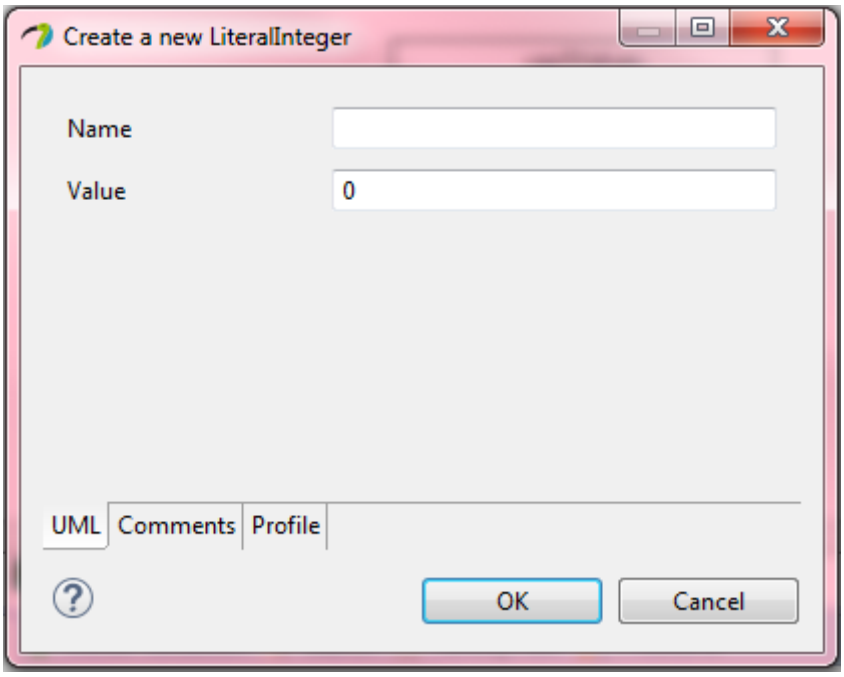

Figura 14. Definir valor para Slot

La segunda opción para definir la *QoSCharacteristic* que instanciará la *QoSValue* es la recomendada. Ésta consiste en seleccionar la *QoSCharacteristic* en la vista *Model Explorer* y arrastrala hasta la *QoSValue*. Con esto ya se estaría definiendo el *classifier* y es equivalente a lo que se muestra en la Figura 11. La ventaja de este método es que, una vez arrastrada la *QoSCharacteristic* se despliega un cuadro de diálogo con las *QoSDimensions* definidas para

ésta, de modo que se podrá seleccionar para cuáles *QoSDimensions* se desea crear *Slots*. Esto se puede ver en la Figura 15.

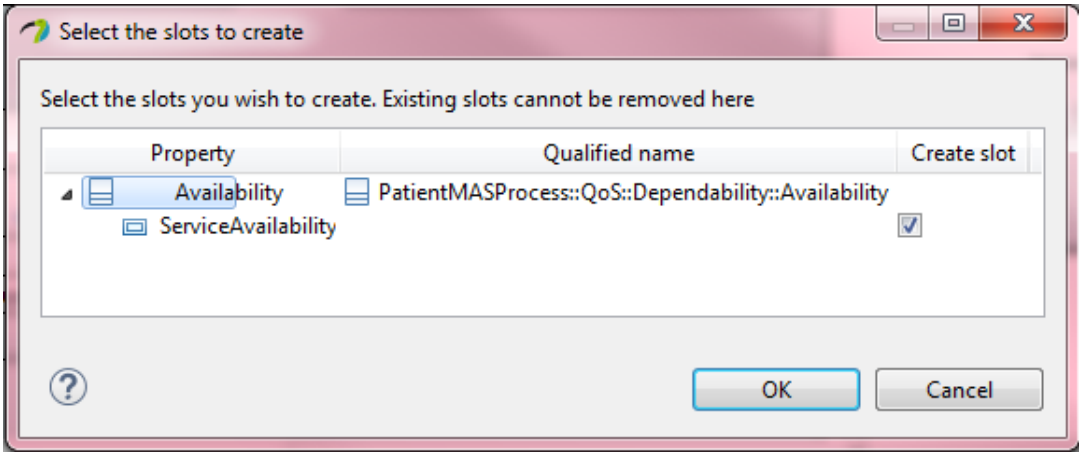

Figura 15. Selección de slots a crear

Por una limitación de *Papyrus* el elemento *Slot* no es mostrado en el lienzo, dentro de la *QoSValue*. Si se desea agregar para completar el modelo alcanza con arrastrarlo desde la vista *Model Explorer* a la *QoSValue*.

La definición del tipo y el valor se realiza de la misma forma que para la primera opción.

Por último, resta asociar la *QoSValue* al *ServiceContract*. Para esto se debe hacer *click* sobre el elemento *QoSContract* en la paleta, luego hacer click sobre el elemento *ServiceContract* en el lienzo y después sobre el elemento *QoSValue* al que se quiere asociar.

3.3.Diagrama de Participantes como Clases con QoS

En este diagrama también se permiten agregar elementos del tipo *QoSValue* que serán asociados a puertos de participantes. Para su creación se pueden seguir los mismos pasos descriptos en la sección 12.

Como se mencionó antes, estos *QoSValues* pueden ser asociados a puertos de tipo *Service* y *Request* y esto dependerá de lo que se quiera modelar.

Si lo que se quiere modelar es la oferta de una cierta característica de calidad, por ejemplo "Este servicio tiene un tiempo de respuesta de 5 segundos", entonces se deberá asociar un puerto del tipo *Service* con una *QoSValue* que modele lo anterior mediante el uso de un conector del tipo *QoSOffered.* Para esto se debe hacer *click* sobre el elemento *QoSOffered* en la paleta y luego sobre el puerto de tipo *Service* y sobre la *QoSValue*.

Si lo que se quiere modelar es alguna característica requerida por el servicio, por ejemplo, la provisión de usuario y contraseña para controlar el acceso al servicio, entonces se deberá asociar un puerto del tipo *Service* con una *QoSValue* que modele lo anterior mediante el uso de un conector del tipo *QoSRequired.* Para esto se debe hacer *click* sobre el elemento *QoSRequired* en la paleta y luego sobre el puerto de tipo *Service* y sobre la *QoSValue*.

Si se quieren modelar características requeridas por el cliente, se deberá asociar un puerto del tipo *Request* con una *QoSValue* mediante el conector *QoSRequired.*

3.4.Diagrama de Participantes como Componentes con QoS

En este diagrama se permiten agregar elementos con tipo *QoSValue*, *QoSRequired, QoSOffered* y *Slot*, de igual forma que para el *Diagrama de Participantes como Clases con QoS*, como se explicó en la sección 3.3.

4. Pasos para generar código asociado a modelo SoaML+QoS

Para generar el código asociado a un modelo SoaML+QoS se debe ir al menú SoaML Toolkit en la barra de herramientas de Eclipse y luego hacer *click* en el submenú SoaML2Code, como se muestra en la Figura 16.

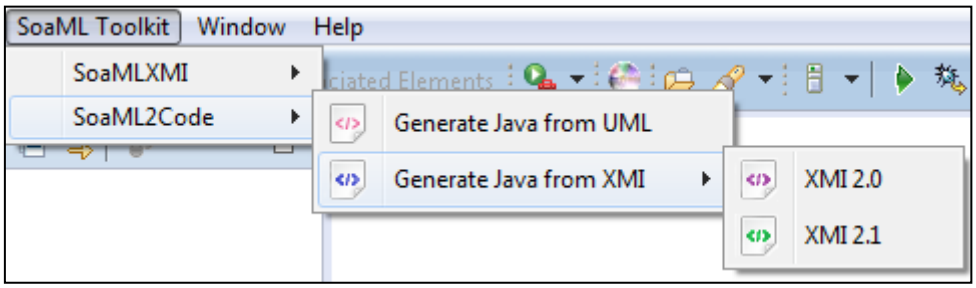

Figura 16. Menú SoaML Toolkit - SoaML2Code

Una vez allí se tienen dos opciones: generar el código a partir de un archivo UML o hacerlo desde un archivo XMI en sus dos versiones, 2.0 o 2.1.

En caso de seleccionar la opción *Generate Java from UML*, se muestra un cuadro de diálogo donde se debe seleccionar un proyecto del *workspace* de Eclipse actual que contenga un archivo UML, como se muestra en la Figura 17.

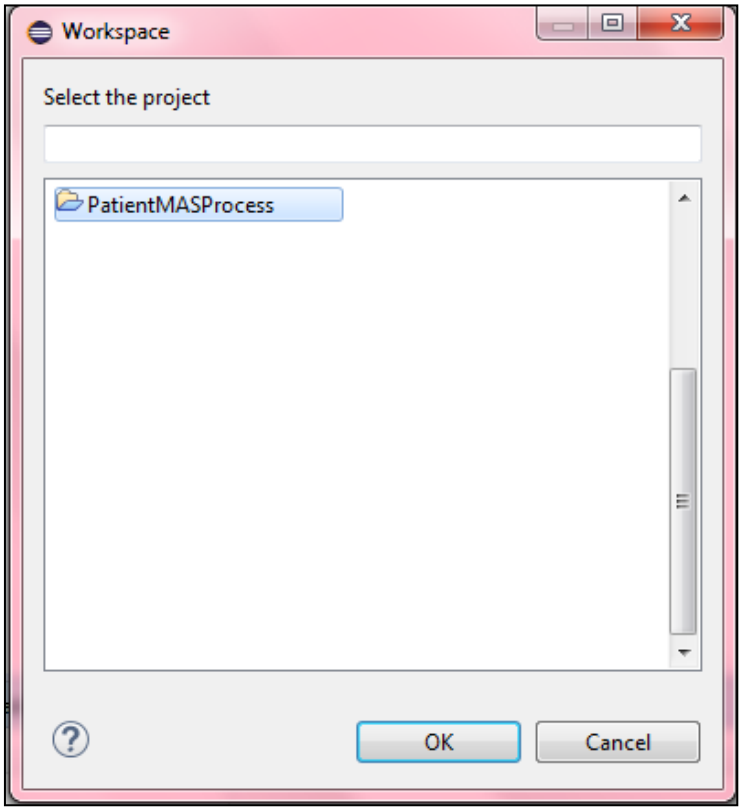

Figura 17. Selección de proyecto con archivo UML

Si se selecciona la opción *Generate Java from XMI*, independientemente de la versión seleccionada, se abre un cuadro de diálogo donde se deberá seleccionar un archivo XMI, el cual podrá estar en cualquier directorio. Esto se puede observar en la Figura 18.

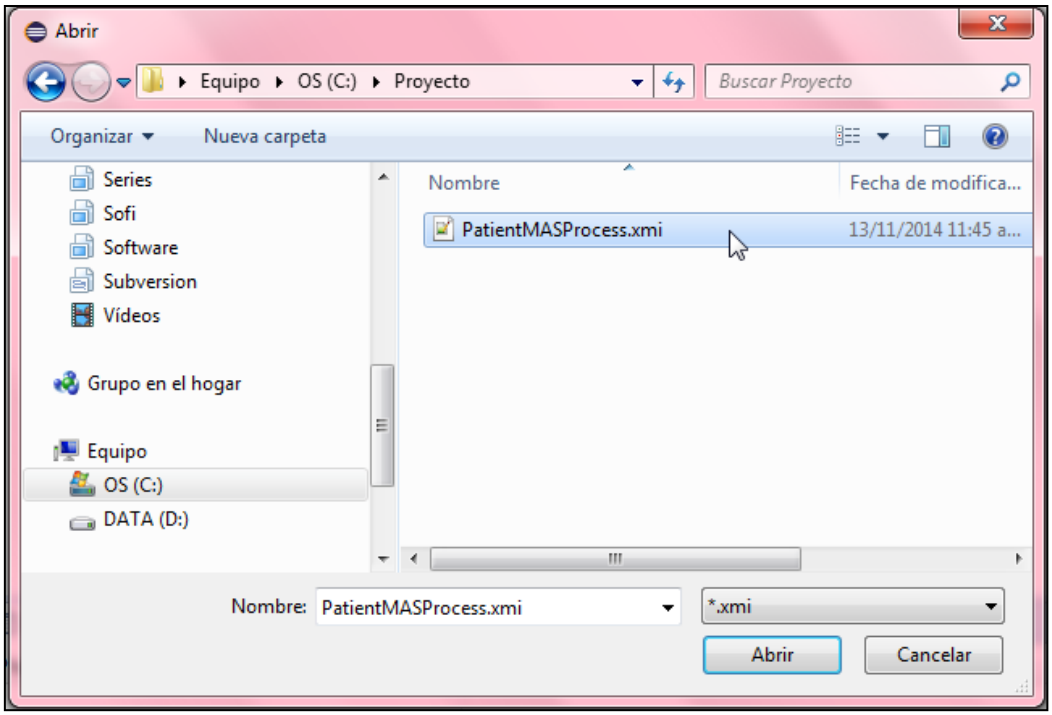

Figura 18. Selección de archivo XMI

Es importante destacar que el formato del archivo de entrada debe cumplir con la estructura general presentada en la Figura 19. En particular, dentro del paquete QoS (que debe aparecer en cuarta posición respecto a los demás paquetes) deben estar todos los elementos QoS del modelo.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI>
    <uml:Model name="PatientMASProcess" xmi:id="0">
        <packagedElement name="Participants">
            <packagedElement name="Participant1"> ... </packagedElement>
            ..
            <packagedElement name="ParticipantN"> ... </packagedElement>
        </packagedElement>
        <packagedElement name="Messages">
            <packagedElement name="Message1"/>
            ..
            <packagedElement name="MessageN"/>
        </packagedElement>
        <packagedElement name="Services">
            <packagedElement name="Service1">
                <packagedElement>
                    <ownedOperation name="Operation1">
                        <ownedParameter name="parameter"/>
                    </ownedOperation>
                </packagedElement>
                ..
                <packagedElement>
                    <ownedOperation name="OperationN">
                        <ownedParameter name="parameter"/>
                    </ownedOperation>
                </packagedElement>
            </packagedElement>
            ..
            <packagedElement name="ServiceN">
                <packagedElement>
                    <ownedOperation name="Operation1">
                        <ownedParameter name="parameter"/>
                    </ownedOperation>
                </packagedElement>
                ..
                <packagedElement>
                    <ownedOperation name="OperationN">
                        <ownedParameter name="parameter"/>
                    </ownedOperation>
                </packagedElement>
            </packagedElement>
        </packagedElement>
        <packagedElement name="QoS">
            ...
        </packagedElement>
        ..
</xmi:XMI>
```

Figura 19. Estructura general del modelo de entrada

Una vez confirmado el archivo de entrada, se muestra un *wizard* en donde se deberá seleccionar las características que tendrá el o los proyectos generados.

Como se muestra en la Figura 20, la primera página del *wizard* muestra los participantes encontrados en el modelo de entrada y permite seleccionar para cuáles se desea generar un proyecto.

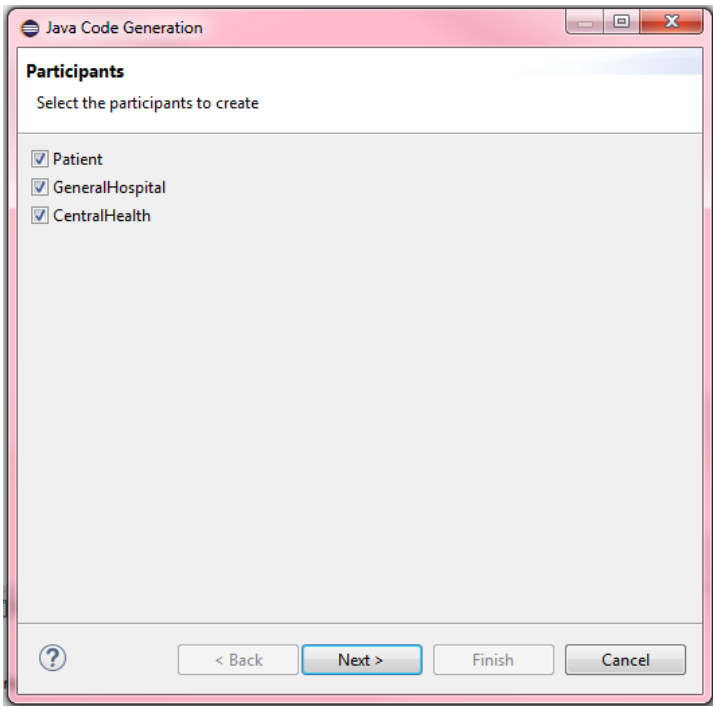

Figura 20. Selección de participantes

Luego de seleccionar los participantes y hacer *click* en *Next,* se muestra la página con las opciones de servidor y cliente a generar, como se ve en la Figura 21.

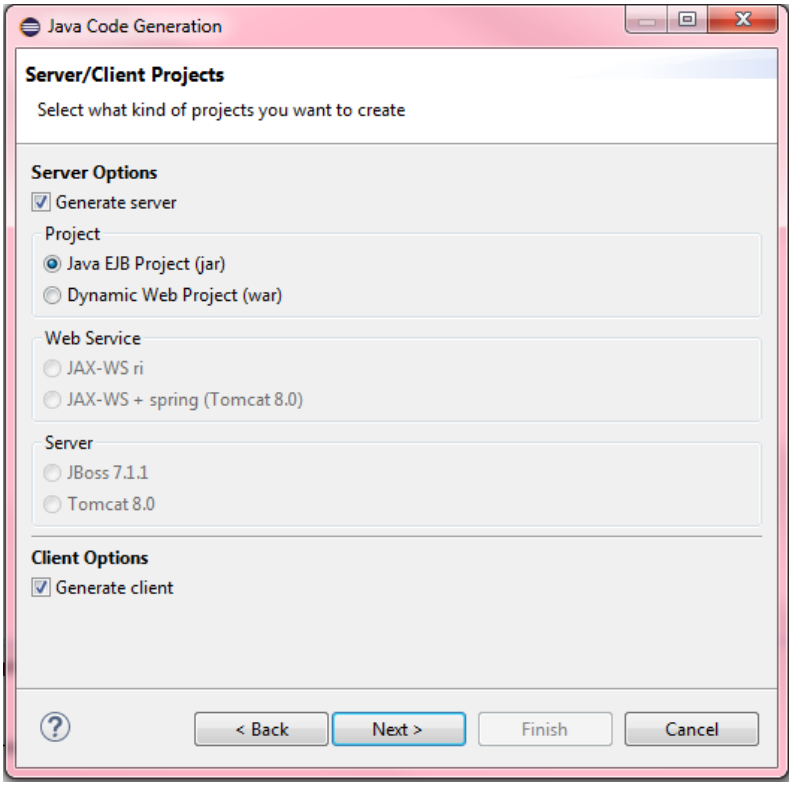

Figura 21. Selección de servidor y cliente

Para interiorizarse sobre lo generado dependiendo del servidor seleccionado, referirse al Manual de Usuario de SoaML2Code, en su primera versión.

Como se muestra en la Figura 22, la siguiente página del *wizard* permite seleccionar la IP y puerto en que se publicará o consumirá el servicio.

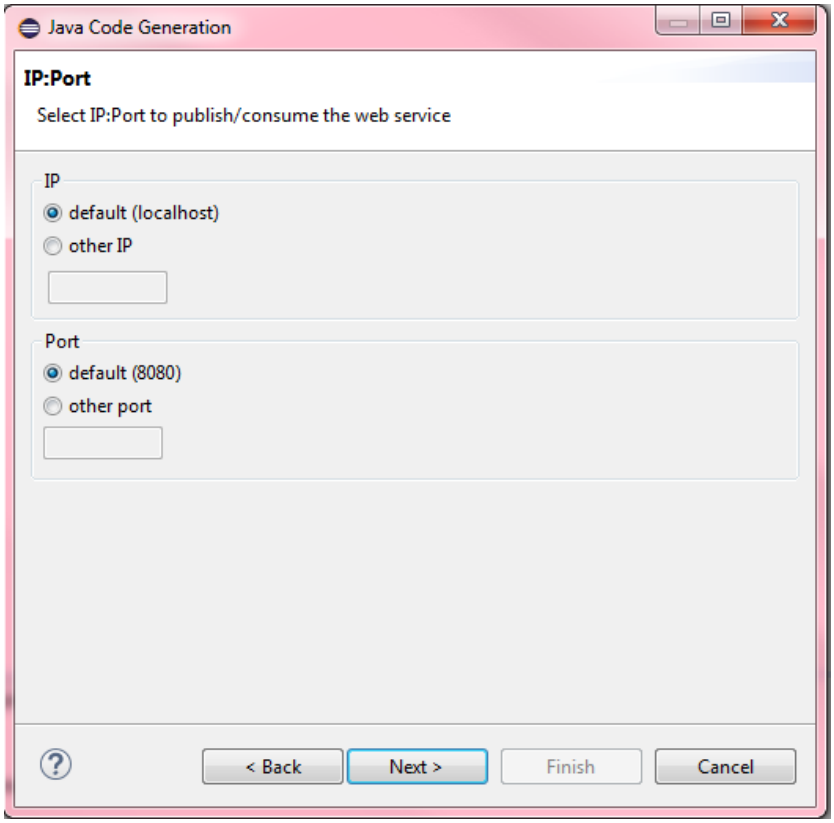

Figura 22. Selección de IP y puerto

En la cuarta y última página del *wizard* se muestran las opciones de generación de características de calidad, como se ve en la Figura 23. Ésta es mostrada únicamente si se encuentran elementos QoS en el modelo de entrada.

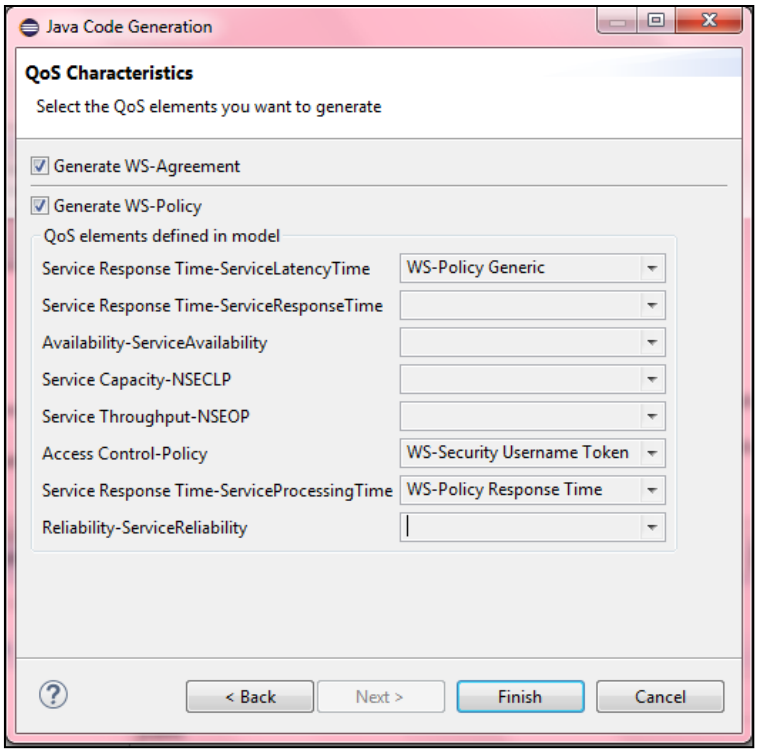

Figura 23. Selección de WS-Agreement y WS-Policy

Existe la posibilidad de que el usuario seleccione no generar nada de QoS, por lo que el proyecto generado será igual al generado para un modelo SoaML.

Una vez configuradas las características de los proyectos a generar y finalizado el *wizard*, el plugin procederá a realizar la generación. A continuación se explican las distintas opciones de lo generado, dependiendo de lo que se seleccionó en la página de servidor/cliente y en la de características de calidad.

4.1.Generación de WS-Agreement

Si se selecciona que se desea generar documentos WS-Agreement, para cada participante elegido se obtienen todos los elementos del tipo *ServiceContract* que contengan al participante (ya sea como proveedor o consumidor) y que estén asociados a al menos un elemento *QoSValue*. Para cada uno de los contratos, se genera un documento WS-Agreement que contiene parte de la información del contrato y del servicio implementado, y en donde se especifican las características de calidad asociadas al contrato. El contrato para el cual el participante está como proveedor se genera en el proyecto del servidor, y en el proyecto del cliente se genera el contrato para el cual el participante está como consumidor. La Figura 24 muestra un ejemplo de un documento WS-Agreement generado por esta herramienta.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:Agreement
    AgreementId="ReceiveSurgerydateReservationServiceContract1" xmlns:wsag=
    "http://schemas.ggf.org/graap/2007/03/ws-agreement">
    <wsag:Name>ReceiveSurgerydateReservationServiceContract</wsag:Name>
    <wsag:Context>
        <wsag:AgreementInitiator>GeneralHospital</wsag:AgreementInitiator>
        <wsag:AgreementResponder>Patient</wsag:AgreementResponder>
        <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
    </wsag:Context>
    <wsag:Terms>
        <wsag:All>
            <wsag:ExactlyOne><wsag:All>
                    <wsag:ServiceDescriptionTerm
                        wsag:ServiceName="ReceiveSurgerydateReservationService" wsag:name=
                        "ReceiveSurgerydateReservationService_SDT"/>
                </wsag:All></wsag:ExactlyOne>
            <wsag:GuaranteeTerm wsag:Name="ServiceAvailability_GT"
                wsag:Obligated="ServiceProvider" xmlns:wsag=
                "http://schemas.ggf.org/graap/2007/03/ws-agreement">
                <wsag:ServiceScope>
                    <wsag:ServiceName>ReceiveSurgerydateReservationService</wsag:ServiceName>
                </wsag:ServiceScope>
                <wsag:ServiceLevelObjective>
                    <wsag:KPITarget>
                        <wsag:KPIName>ServiceAvailability</wsag:KPIName>
                        <wsag:Target>99.5 %</wsag:Target>
                    </wsag:KPITarget>
                </wsag:ServiceLevelObjective>
                <wsag:BusinessValueList/>
            </wsag:GuaranteeTerm>
        </wsag:All>
    </wsag:Terms>
</wsag:Agreement>
```

Figura 24. Ejemplo de WS-Agreement generado

El documento WS-Agreement generado es un esqueleto que el usuario deberá completar para su correcta definición.

4.2.Generación de WS-Policy

Si se selecciona generar WS-Policy, el wsdl es modificado para incluir las distintas políticas disponibles. En caso de haber elegido WS-Policy Generic o WS-Policy Response Time, el resultado en el wsdl es el mismo. Se puede ver un ejemplo en la Figura 25.

```
<wsdl:service name="ReceiveSurgerydateReservationService">
    <wsdl:port binding="serv:ReceiveSurgerydateReservationServiceBinding"
        name="ReceiveSurgerydateReservation">
        <soap:address
            location="http://localhost:8080/PatientServer/ReceiveSurgerydateReservationService" />
        <wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy">
            <wsp:All>
                <ServiceLatencyTime wsp:Optional="true"
                    direction="decreasing" unit="ms" value="10" />
                <ServiceResponseTime direction="decreasing"
                    unit="ms" value="45" />
            </wsp:All>
        </wsp:Policy>
    </wsdl:port>
</wsdl:service>
```

Figura 25. Ejemplo de policy genérica

En caso de seleccionar WS-Security Username Token el resultado se puede ver en la Figura 26.

```
<wsdl:service name="ReceiveSurgerydateReservationService">
    <wsdl:port
        binding="serv:ReceiveSurgerydateReservationServiceBinding" name="ReceiveSurgerydateReservation">
        <soap:address location="http://localhost:8080/PatientServer/ReceiveSurgerydateReservationService"/>
        <wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy">
            <wsp:All>
                <sp:SupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
                    <wsp:Policy>
                        <sp:UsernameToken sp:IncludeToken=
                        "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                            <wsp:Policy>
                                <sp:WssUsernameToken11/>
                            </wsp:Policy>
                        </sp:UsernameToken>
                    </wsp:Policy>
                </sp:SupportingTokens>
            </wsp:All>
        </wsp:Policy>
    </wsdl:port>
</wsdl:service>
```

Figura 26. Ejemplo de policy Username Token

Si se seleccionan distintos tipos de políticas, éstas se anidan como se muestra en la Figura 27.

```
<wsdl:service name="ReceiveSurgerydateReservationService">
    <wsdl:port
        binding="serv:ReceiveSurgerydateReservationServiceBinding" name="ReceiveSurgerydateReservation">
        <soap:address location="http://localhost:8080/PatientServer/ReceiveSurgerydateReservationService"/>
        <wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy">
            <wsp:All>
                <ServiceLatencyTime direction="decreasing" unit="ms" value="10"/>
                <sp:SupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
                    <wsp:Policy>
                        <sp:UsernameToken sp:IncludeToken=
                        "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                            <wsp:Policy>
                                <sp:WssUsernameToken11/>
                            </wsp:Policy>
                        </sp:UsernameToken>
                    </wsp:Policy>
                </sp:SupportingTokens>
            </wsp:All>
        </wsp:Policy>
    </wsdl:port>
</wsdl:service>
```

Figura 27. Ejemplo de varias policies

Según lo que se haya seleccionado en el *wizard* para QoS se generarán distintos elementos en el código resultante, dependiendo también de qué tipo de proyecto se decidió generar y para qué servidor. Vale aclarar que para el caso de haber seleccionado un proyecto Java EJB, los archivos wsdl contienen las políticas de calidad y de control de acceso, pero los proyectos no contienen las clases Java necesarias para validar el token de seguridad o la configuración para ignorar las políticas de calidad cuando se recibe un mensaje SOAP.