2012

Proyecto de Grado

Extensión del plug-in de Eclipse SoaML para generación de código JEE y WS desde modelos SoaML

Victor Fuica - Alejandro Goss - Natalia Maurizio

Informe Final

Tutor
Andrea Delgado
Cotutor
Laura González

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

Resumen

La orientación a servicios se ha convertido actualmente en una de las formas más utilizada para la informatización de los procesos de negocios de una organización. El paradigma de Computación Orientada a Servicios (Service Oriented Computing, SOC) promueve la descomposición de la lógica de la solución en unidades pequeñas de trabajo autónomo especializadas en resolver una porción específica del problema. Dichas unidades de trabajo son los llamados servicios que pueden ser usados y combinados para el desarrollo de nuevas aplicaciones, permitiendo un desarrollo rápido, con gran flexibilidad y agilidad ante cambios en el funcionamiento a nivel de negocio y en consecuencia una baja en los costos. SoaML (SOA Modeling Language, SoaML) es el estándar propuesto por la OMG (Object Management Group, OMG) para el modelado de servicios con UML. Es un perfil que extiende el lenguaje de modelado UML (Unified Modeling Language, UML) incorporando conceptos específicos a este dominio, y define un metamodelo que especifica la sintaxis y semántica correcta para este estándar.

Este trabajo se centró en el desarrollo de un plug-in para el entorno de desarrollo Eclipse que extendiera el plug-in SoaML, desarrollado en un proyecto previo, que implementa el estándar SoaML permitiendo la generación de diagramas de forma gráfica, la importación y exportación de los modelos generados en formato XMI (XML Metadata Interchange, XMI) para permitir la interoperabilidad con otras herramientas. El plug-in generado en este proyecto permite que a partir de un modelo SoaML representado por los diagramas realizados en el plug-in SoaML, genere código ejecutable en lenguaje Java que represente dicho modelo, y exponga los servicios provistos en el modelo como Web Services. Actualmente existen muy pocas herramientas que ofrezcan el modelado de servicios con el estándar SoaML en conjunto con la generación de código que represente dicho modelo, la mayoría de las cuales son comerciales y cuyas funcionalidades se analizaron como parte de los requerimientos para el desarrollo del plug-in. Luego del análisis de diferentes alternativas existentes para la generación de código ejecutable que permitan la exposición de Web Services, se diseñó e implementó una solución utilizando 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 plug-in de Eclipse denominado SoaML2Code que fue desarrollado en este proyecto de grado, 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 plug-in SoaML. Puntualmente se decidió exponer los servicios 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 obtenida es de fácil utilización, y al encontrarse en el contexto de Eclipse provee a la comunidad de un plug-in 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.

Palabras Claves: Service Oriented Computing (SOC), Servicios, Modelado, SoaML, Plugins, Eclipse,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. Entre los objetivos se incluye el estudio del marco teórico de la orientación a servicios (Service Oriented Computing, SOC) y su relación e integración con Procesos de Negocio (Business Process Management, BPM) y desarrollo dirigido por modelos (Model Driven Development, MDD). El desarrollo dirigido por modelos basa el desarrollo de software en modelos, utilizando como artefactos de primer orden metamodelos, modelos y lenguajes que permiten transformaciones entre estos. Estas transformaciones convierten sucesivamente un modelo en otro modelo del mismo sistema, hasta llegar al código asociado. A su vez, el estándar SoaML de OMG es un perfil y metamodelo UML liberado en el año 2009 que permite modelar servicios con elementos UML y estereotipos específicamente definidos. En el año 2010 fue desarrollado en el marco de un proyecto de grado, el Eclipse SoaML plug-in [1] basado en Papyrus[1] (UML2), el cual implementa el estándar SoaML permitiendo la generación de modelos SoaML. En este contexto, este documento presenta el desarrollo del Eclipse SoaML2Code plug-in, el cual proporciona una herramienta para la generación de código ejecutable Java a partir de modelos SoaML.

1.1. Motivación

El uso de modelos en disciplinas tradicionales de ingeniería es una práctica aceptada y tiene una amplia historia. Hoy en día la mayor parte de las metodologías usadas en Ingeniería de Software (IS) utilizan modelos como herramienta principal para desarrollar software, pero con foco en la documentación. Por otro lado, hay una tendencia a seguir un enfoque dirigido por modelos en los que éstos son los artefactos principales del desarrollo [2]. Las organizaciones están generalizando el uso de Web Services y poniendo foco en transformar sus sistemas en una Arquitectura Orientada a Servicios (SOA). Con este nuevo enfoque y con la ayuda de estándares de las tecnologías de la información (Java EE, XML, Web Services), las organizaciones pueden independizarse de la tecnología en que se encuentran implementadas las distintas soluciones, desarrollar procesos de negocio de forma más rápida y flexible y así reaccionar ágilmente a los cambios de negocio adaptando e integrando procesos de negocio y sus tecnologías.

La arquitectura orientada a servicios (Service Oriented Architecture, SOA) es un estilo de arquitectura de software que propone la construcción de aplicaciones mediante el ensamblado de bloques reusables, débilmente acoplados y altamente interoperables, cada uno de los cuales es representado como un servicio. Los mismos pueden encontrarse distribuidos y pertenecer potencialmente a diferentes propietarios. Es por ello que dicha arquitectura es utilizada para la integración de aplicaciones empresariales (Enterprise Application Integration, EAI) y comercio electrónico entre empresas (Business-to-business, B2B). Para poder especificar arquitecturas orientadas a servicios de forma clara y ágil, es necesario contar con un lenguaje de modelado particular a este dominio. Un avance en este tema fue la definición del estándar SoaML[3] (Soa Modeling Language, SoaML) de la OMG (Object Managment Group, OMG). Este estándar extiende la notación UML [4] [5] (Unified Modeling Language, UML) mediante la definición de un metamodelo y un perfil UML. SoaML puede ser integrado para su utilización con el enfoque MDA (Model Driven Architecture, MDA) [6] para poder diseñar SOA, y también ser utilizado para generar código a partir de dichos modelos.

[1] Papyrus, herramienta de código abierto para modelado gráfico UML2. http://www.papyrusuml.org/

La especificación del proyecto plantea como parte de la motivación para el mismo, que en la actualidad no existen muchas herramientas que implementan el estándar SoaML y que además generen código a partir de modelos SoaML. Además, el producto generado por este proyecto, nos provee una herramienta que soporta MDD y permite automatizar la generación de código desde modelos. De esta manera, se logra realizar un aporte dentro de las herramientas de generación de código a partir del modelado de servicios utilizando el estándar SoaML, que podrá ser liberada a la comunidad como software libre.

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 construir un plug-in que extienda el plug-in de Eclipse SoaML para generar código Java EE y Web Services a partir de modelos SoaML.

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

- (OE1) Estudiar fundamentos de la orientación a servicios con foco en modelado, conceptos de metamodelado, lenguaje de modelado SoaML, perfiles UML y el formato de intercambio de modelos XML (XML Metadata Interchange, XMI), además de la plataforma de desarrollo empresarial Java EE y la tecnología de Web Services.
- (OE2) Evaluar las posibles estrategias y alternativas de implementación para generar código en el entorno de desarrollo Eclipse, así como las herramientas de generación de código a partir de modelos SoaML existentes en la comunidad -tanto comerciales como libres-, para identificar características deseables y características mejorables.
- (OE3) Desarrollar componentes de software para la implementación del plug-in que permitan la generación de aplicaciones en lenguaje Java a partir de la interpretación de modelos SoaML, así como la ejecución de las mismas sobre un servidor de aplicaciones.
- (OE4) Verificar el funcionamiento del producto generado mediante un caso de estudio que permita probar las características de la solución elaborada y evaluar los resultados obtenidos.

1.3. 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, el tiempo insumido por cada una y las desviaciones del cronograma inicial.

Gestión de Configuración

Se utilizó un repositorio del sitio assembla [7] gratuito en el que se almacenaron los archivos fuente y documentos del proyecto. El repositorio está estructurado con los directorios que se muestran en la Figura 1.1.

- **Branch:** contiene las distintas ramas que se abrieron a lo largo del proyecto para congelar distintas versiones de la implementación del plug-in.
- **Documentos:** contiene las distintas versiones de la documentación, entre ellas se encuentra el documento de casos de uso, estado del arte, los documentos de verificación por iteración, el informe final, además de todos los papers y documentos que se usaron en el transcurso del proyecto como materiales.
- **SoaMLToJava:** contiene la versión final del código desarrollado para el plug-in.
- **Trunk:** se encuentran los distintos prototipos que se realizaron durante el proyecto como por ejemplo para el parser de los archivos XMI a objetos Java.

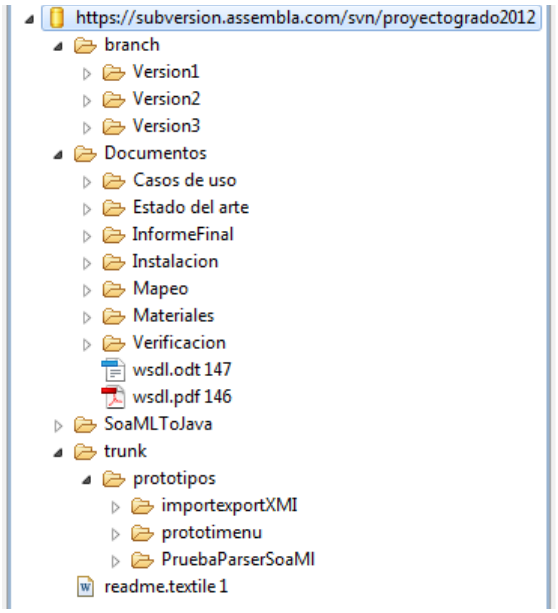


Figura 1.1. Estructura del repositorio utilizado para el proyecto

El uso del repositorio fue práctico para el manejo de los distintos recursos y versiones tanto de los documentos como de los archivos fuente generados a lo largo del proyecto. Facilitó la división de trabajo entre los integrantes del grupo y la aplicación de un desarrollo iterativo incremental tanto para la implementación del plug-in, así como también para la generación de la documentación.

Cronograma previsto y cronograma real

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

- Estudio del estado del arte en orientación a servicios, en particular en lo que refiere a la etapa de modelado, requisitos y características asociadas, estándares existentes.
    - Estudio de la teoría de metamodelado, perfiles UML, estándar XMI para intercambio de modelos
    - Estudio del estándar SoaML: metamodelo y perfil (estereotipos)
- Estudio de plataforma Java EE y tecnología de Web Services
    - Estudio de estándares de Web Services
    - Estudio de plataforma Java EE
- Implementación de la extensión del plug-in SoaML para Eclipse
    - Requerimientos (Casos de Uso)
    - Priorización CU, implementación arquitectura, pruebas
    - implementación 2dos. CU, pruebas
    - implementación 3eros. CU, pruebas
- Caso de estudio de aplicación de modelado de servicios en SoaML con el plug-in Eclipse desarrollado, que podrá ser de laboratorio (ejemplo del estándar de SoaML) o real en alguna organización afín.
- Informe del proyecto de grado durante todo el proyecto por capítulos, final, correcciones y defensa.

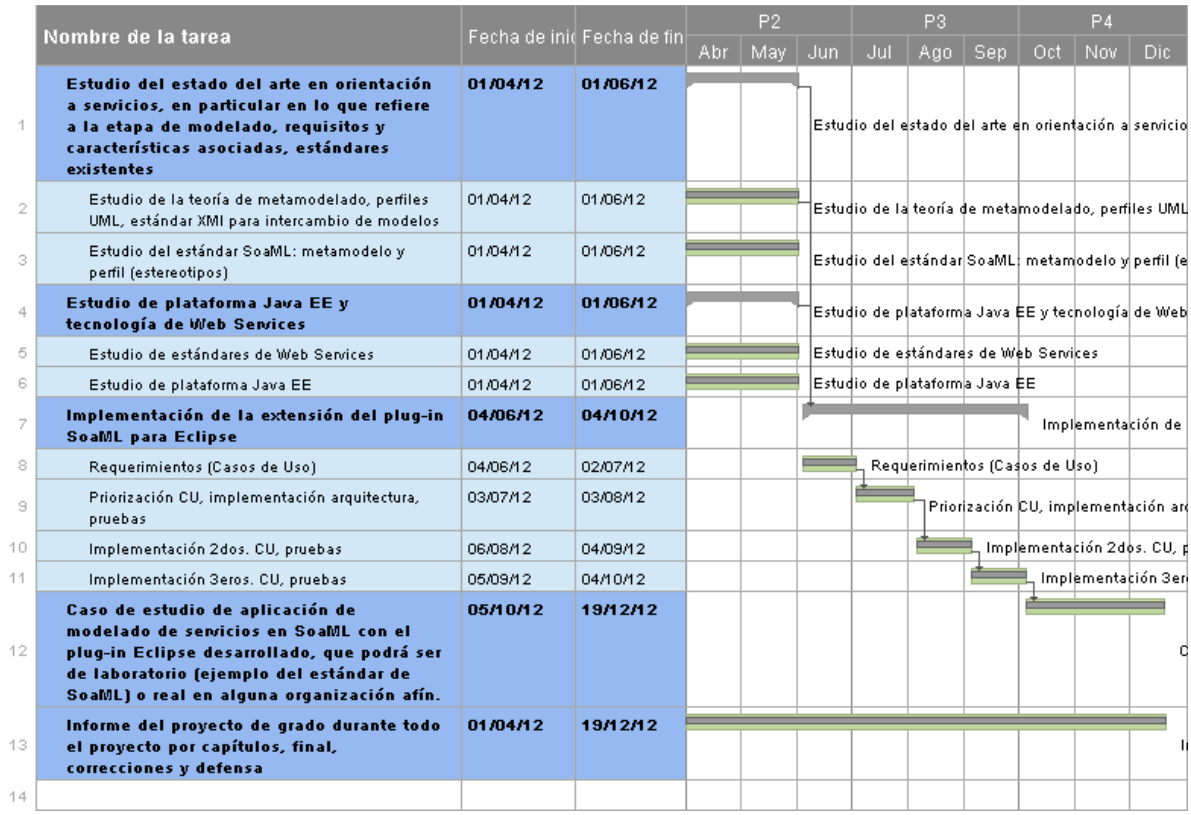


Figura1.2. Diagrama de Gantt del cronograma previsto

Durante el desarrollo del proyecto se produjeron varias desviaciones del cronograma establecido inicialmente. En particular se extendieron las etapas de implementación y documentación, debiéndose acortar las etapas correspondientes a la definición de los casos de uso y al desarrollo del caso de estudio. De todas maneras no se logró cumplir el plazo estipulado inicialmente para la finalización del proyecto de grado en diciembre. En la Figura 1.3 se muestra el diagrama de Gantt con el tiempo real de desarrollo del proyecto, correspondiente a los semestres durante los que se realizó el proyecto.

Las demoras experimentadas fueron en ambos casos por dificultades técnicas. La primera, se debió a que se extendió la investigación de las alternativas de desarrollo para la implementación del plug-in, donde se investigaron algunos de los frameworks que se encargan de generar código a partir de un WSDL. A su vez, el desarrollo de prototipos sobre cada uno de los frameworks nos insumió mayor tiempo del estipulado pero nos permitió minimizar los riesgos existentes de no poder cumplir alguno de los requerimientos principales planteados para el producto. Entre los frameworks considerados, se encontraban JBossWS, Apache Axis, Apache CXF los cuales se detallan más adelante en el Capítulo 4 correspondiente a la evaluación de alternativas de desarrollo. Otra de las causas que incidió en la desviación de tiempo, fue que no se encontró una herramienta que generara código con anotaciones EJB [2] a partir de un WSDL. Esto insumió mucho tiempo de investigación, optándose por utilizar los generadores provistos por los frameworks mencionados anteriormente. Dada esta elección hubo que investigar además, herramientas para manipular las clases generadas, lo que requirió más tiempo del previsto.

[2] Los EJB (Enterprise JavaBeans, EJB) son una de las APIs que forman parte del estándar de construcción de aplicaciones empresariales. El objetivo de estos es evitarle al programador problemas generales de una aplicación empresarial (concurrencia, persistencia, seguridad, etc.). http://java.sun.com/products/ejb/

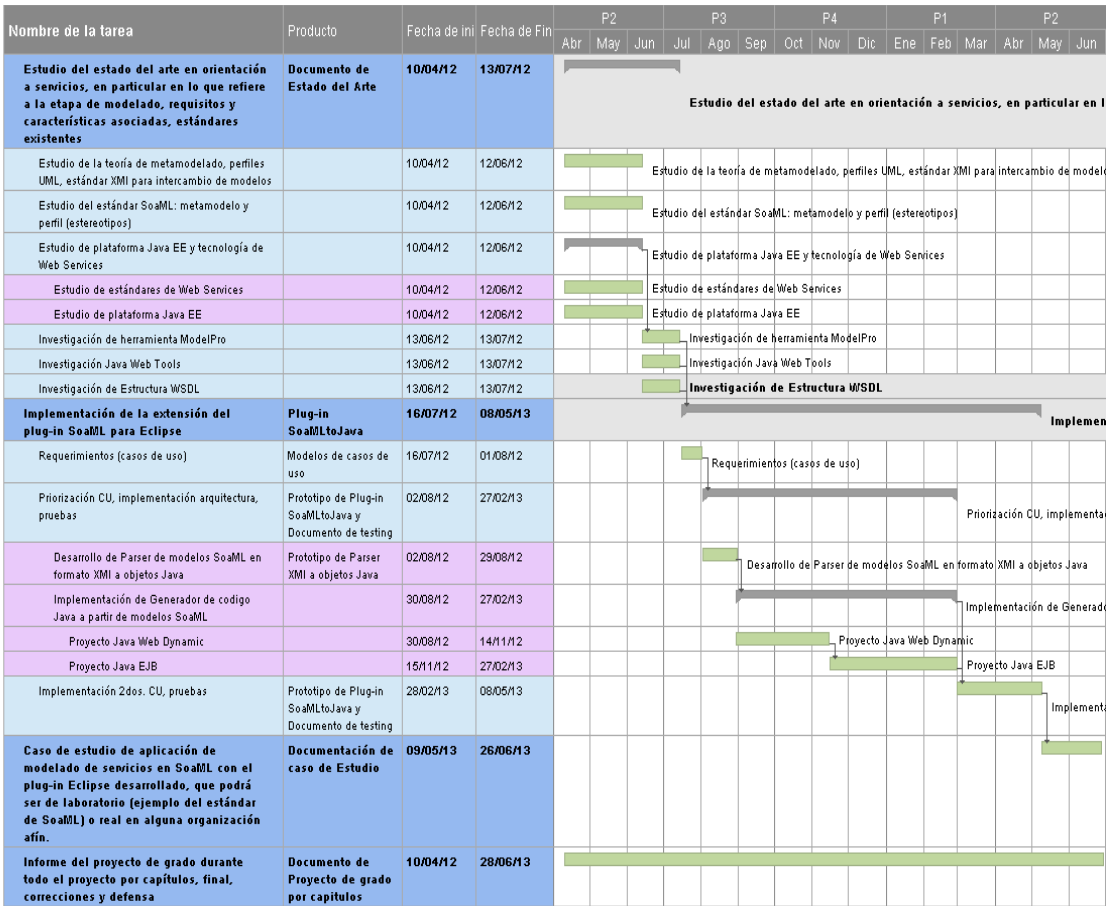


Figura 1.3. Diagrama de Gantt del cronograma real

A continuación se muestran los gráficos correspondientes a las horas de dedicación al proyecto discriminando por meses, por etapas y por actividades desarrolladas. El gráfico de la Figura 1.4 corresponde al consolidado de horas dedicadas -entre los tres integrantes del grupo- en el transcurso de los meses en los que se extendió el presente proyecto de grado.

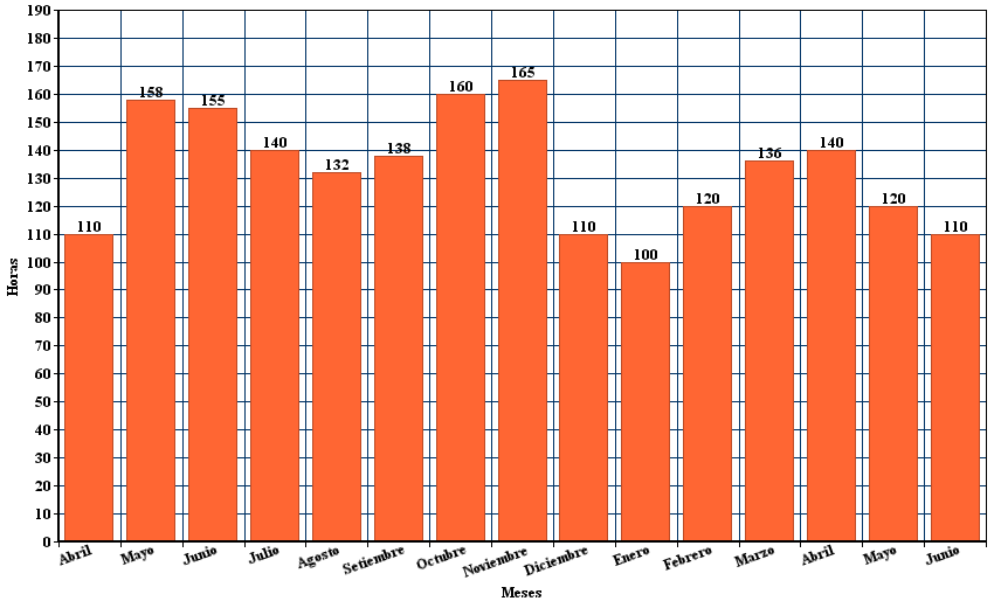


Figura 1.4. Gráfica de horas totales dedicadas al proyecto por mes

La dedicación esperada para el proyecto de grado es de 15 horas semanales, por lo que siendo tres integrantes, la dedicación por mes esperada sería de 180 horas. Como se puede observar en la gráfica, la dedicación en todos los meses estuvo en el entorno de las horas esperada, estando levemente por debajo de dicha cifra en la mayoría de los meses. Además se puede observar que la dedicación de horas al proyecto fue un poco superior en la primera etapa de investigación de las herramientas y frameworks, así como en las primeras iteraciones de implementación. Esto es consistente con la amplitud con la que se enfrentó la investigación al elegir las herramientas y frameworks a utilizar, además de las dificultades presentadas en la etapa de implementación. En la gráfica de la Figura 1.5, se muestra con un mayor nivel de detalle las horas de dedicación discriminadas por fase -Estado del arte, investigación de tecnologías y alternativas de desarrollo para el generador de código, desarrollo iterativo incremental del generador de código e informe final-.

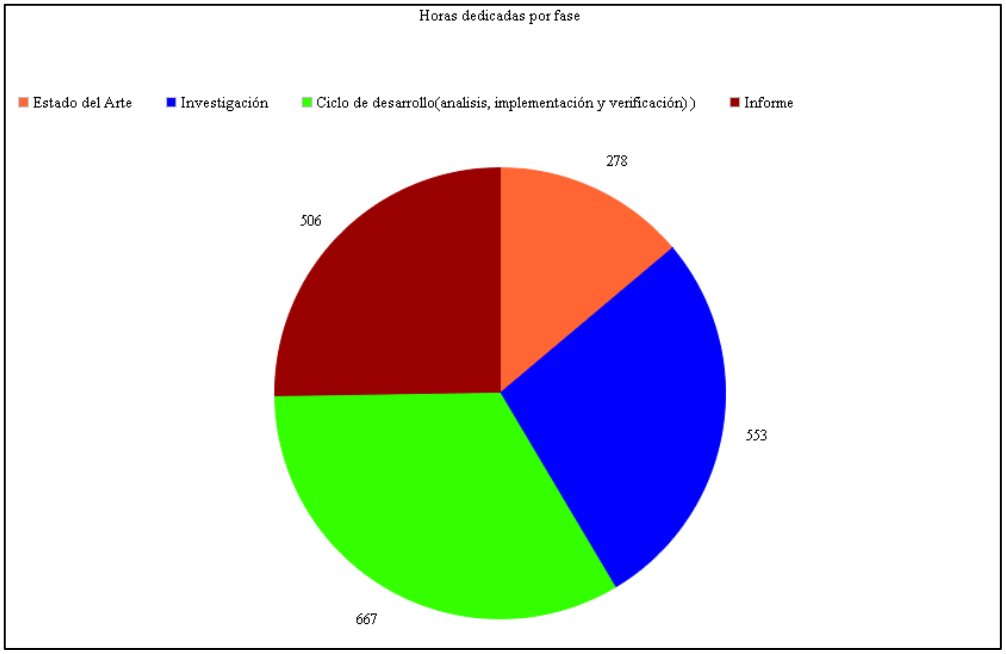


Figura 1.5. Gráfica de horas de dedicación por fase

En la gráfica anterior -de horas por fase-, se puede ver claramente que las etapas que implicaron mayor dedicación fueron las de investigación -de herramientas y frameworks existentes para la generación de código a partir de modelos SoaML- e implementación, que en su conjunto implicaron más del 60% de las horas invertidas en el proyecto. La etapa de desarrollo a su vez abarca distintas sub etapas: análisis, investigación para el "parser" de archivos en formato XMI a objetos Java y para la generación de código a partir de un WSDL, implementación del generador de código y pruebas realizadas para comprobar el correcto funcionamiento del mismo. Tomando los datos en su conjunto, podemos concluir que del total de 2004 horas dedicadas al proyecto, el 27.6% del tiempo empleado se utilizó en investigar las distintas alternativas para desarrollar el generador de código, el 33.3% del tiempo al desarrollo en varios ciclos iterativos incrementales y la verificación del producto generado. El resto del tiempo se dedicó a construir la documentación presentada correspondiente a cada una de las etapas.

1.4. 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 plug-in de Eclipse que permite la generación de código a partir de modelos SoaML.

De esta forma, se aporta a la comunidad una herramienta para Eclipse que junto con el plug-in SoaML completa el ciclo de modelado y desarrollo bajo el estándar SoaML, siguiendo una arquitectura dirigida por modelos (MDA). Cabe destacar que con esta herramienta no hay necesidad de utilizar ninguna otra fuera del IDE de Eclipse. Este producto permite exponer 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.

En cuanto a los objetivos específicos, se lograron estudiar los elementos indicados en el OE1 y a partir de dicho estudio se elaboró el documento de estado del arte que se adjunta como anexo.

Con respecto al OE2, se realizó un estudio exhaustivo en todas las áreas relacionadas a la programación y uso del entorno Eclipse para integración de funcionalidades en forma de plug-in y se investigaron cuáles eran las herramientas existentes para el desarrollo de generadores de código a partir de modelos SoaML, así como también se investigaron y analizaron distintos frameworks para la generación de código que permiten exponer servicios.

Para cumplir con el OE3, se implementó un parser y un generador de código que a partir del resultado obtenido por el primero, nos brinda un software de aplicaciones en lenguaje Java que representa la realidad expuesta por el modelo y puede ser ejecutado en un servidor de aplicaciones.

Por último, para cumplir el OE4, se logró realizar la verificación de la herramienta construida, a partir del caso de estudio seleccionado [8], obteniendo como resultado, la generación del código esperado que representa el modelo estudiado.

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 y herramientas existentes para la generación de código. En el Capítulo 4, se presenta la investigación y el estudio de las alternativas de desarrollo evaluadas. En el Capítulo 5 se describe la solución propuesta en conjunto con su arquitectura e implementación realizada. En el Capítulo 6, se expone el plan de verificación y los resultados obtenidos. En el Capítulo 7, se describe el caso de estudio y su realización utilizando el plug-in SoaML2Code. Por último, en el Capítulo 8, 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 Verificación Iteración I e Iteración II.
- Documento Técnico.
- Guía de Instalación
- Manual de Usuario

2. Estado del arte

En esta sección se abordan los temas estudiados en la primera etapa del proyecto con el objetivo de establecer un marco conceptual que sirva para el entendimiento del resto del documento.

El capítulo consta de ocho secciones descriptas a continuación. En la sección 2.1 se presenta el concepto de SOA (Arquitectura Orientada a Servicios, SOA) y conceptos relacionados como el paradigma SOC (Service Oriented Computing, SOC). En la sección 2.2 se presenta el lenguaje de modelado SoaML, que se utiliza para la especificación de una arquitectura orientada a servicios. En la sección 2.3 se aborda el concepto de MDA (Model Driven Architecture, MDA), el cual es un nuevo paradigma que surge a partir de las tecnologías que están siendo estandarizadas por la OMG. En la sección 2.4 se describe el estándar XML Metadata Interchange (XMI) de la OMG, que permite el intercambio de modelos. Por último, en las cuatro secciones finales se da un contexto sobre las tecnologías que componen la solución del proyecto a desarrollar, las cuales son la tecnología Web Services, la Plataforma Java EE, el IDE de desarrollo Eclipse y el plug-in SoaML desarrollado para el entorno de desarrollo de Eclipse.

Para un análisis más profundo sobre estos temas, si el lector así lo desea puede dirigirse a las referencias citadas o al documento del Estado del Arte que se incluye como anexo.

2.1. SOA: Service-Oriented Architecture

SOA es un estilo de arquitectura de software que define la utilización de servicios como construcciones básicas para el desarrollo de aplicaciones. Se basa en la independencia de plataformas de hardware, de sistemas operativos y de lenguajes de programación. SOA fortalece la reutilización de los sistemas actuales que se construyeron y se utilizaron durante años y crea un ambiente en el que los procesos de negocio y la tecnología de la información pueden interactuar entre sí. En una aplicación que utiliza este estilo, sus funcionalidades se definen como servicios independientes, con interfaces que pueden ser llamadas en secuencias dadas para formar procesos de negocios [9].

SOA se fundamenta en:

- Ejecutar rápido.
- Adaptarse al mercado.
- Reutilizar los componentes de los procesos de negocios.
- Garantizar resultados que sean repetibles y predecibles

Motivación

En muchos casos crear aplicaciones requiere de funcionalidades ya antes implementadas como parte de otros sistemas. En este punto los arquitectos de software se pueden enfrentar a dos opciones:

1. Reutilizar funcionalidad ya existente en otros sistemas. Una labor difícil, debido a que éstas no fueron pensadas para integrarse, encontrándose implementadas sobre plataformas y/o tecnologías no compatibles entre ellas.
2. Re-implementar la funcionalidad ("reinventar la rueda"). Aunque implica más tiempo de desarrollo, es en la mayoría de los casos la más fácil y segura.

La segunda opción trae como resultado que la misma funcionalidad se encuentre replicada en varias aplicaciones, dificultando de esta manera la migración de los sistemas internos por tener múltiples conexiones desde sistemas que dependen de éstos para su funcionamiento. Al no haber una estrategia de integración de aplicaciones, se generan múltiples puntos de falla en múltiples aplicaciones. El modelo generalmente es muy poco escalable.

Ventajas de la arquitectura SOA

Los servicios pueden tener distinta granularidad. Un servicio puede hacer uso de otros servicios sin importar su ubicación física. Así un sistema evoluciona con la adición de nuevos servicios y su mejoramiento, donde cada servicio evoluciona de una manera independiente. SOA define los servicios de los cuales estará compuesto el sistema, sus interacciones y con qué tecnologías serán implementados. Las interfaces que utiliza cada servicio para exponer su funcionalidad son manejadas por contratos, que definen claramente el conjunto de mensajes soportados, su contenido y las políticas aplicables.

Elementos SOA

Esta arquitectura presenta una forma de construir sistemas distribuidos que entreguen funcionalidad como servicios para aplicaciones de uso final u otros servicios. En la Figura 2.1 se muestra un esquema de una arquitectura orientada a servicios.

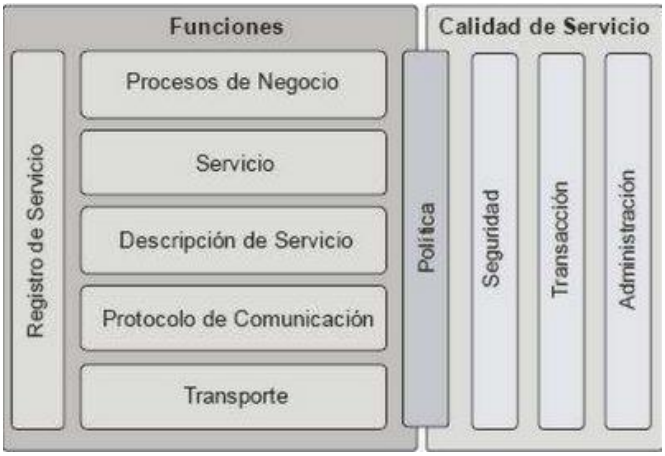


Figura 2.1. Elementos de la Arquitectura SOA [9]

A continuación se presenta una breve descripción de los componentes representados en la Figura 2.1.

Funciones:

- *Transporte*: Es el mecanismo utilizado para llevar los pedidos de servicio desde un consumidor de servicio hacia un proveedor de servicio y las respuestas desde el proveedor hacia el consumidor.
- *Protocolo de comunicación de servicios:* Es un mecanismo acordado a través del cual un proveedor de servicios y un consumidor de servicios comunican qué está siendo solicitado y qué está siendo respondido.
- *Descripción de servicio:* Es un esquema acordado para describir qué es el servicio y qué datos requiere el servicio para invocarse correctamente.
- *Servicios*: Describe los servicios que están actualmente para ser utilizados.
- *Proceso de Negocio*: Procesos que satisfacen un requerimiento de negocio y son realizados con invocaciones a servicios.
- *Registro de Servicios*: Es un repositorio de descripciones de servicios donde un proveedor puede publicar sus servicios, así como un consumidor puede hallar servicios disponibles.

Calidad de Servicio:

- *Administración:* Es el conjunto de atributos que pueden aplicarse para manejar los servicios manejados o consumidos.
- *Transacciones:* Conjunto de atributos que podrían aplicarse a un grupo de servicios para entregar resultados consistentes.

- *Seguridad:* Es el conjunto de reglas que se aplican para la identificación, autorización y control de acceso a consumidores de servicios.
- *Política:* Conjuntos de reglas que deben cumplir el proveedor para dejar disponible sus servicios a los consumidores.

Principios de la orientación a servicios

No existe una definición exacta de la Orientación a Servicios. Lo que se puede proporcionar es un conjunto de principios asociados a la orientación a Servicios. Estos principios son según *Thomas Erl* [9]:

- **Los servicios deben ser reusables**: todos los servicios deben ser diseñados y construidos pensando en su reutilización tanto dentro de la aplicación como para ser usado por otras.
- **Los servicios deben proporcionar un contrato formal**: Se debe proporcionar un contrato en el cual figure: nombre del servicio, forma de acceso, funcionalidades que ofrece, datos de entrada y salida de cada funcionalidad.
- **Los servicios deben tener bajo acoplamiento**: Los servicios deben ser independientes de otros. Para conseguir bajo acoplamiento, lo que se hará es acceder a los servicios a través de su contrato, logrando la independencia entre el servicio que se va a ejecutar y el que lo llama.
- **Los servicios deben permitir la composición**: Todo servicio debe ser construido de manera que puedan ser utilizados para construir servicios genéricos de más alto nivel.
- **Los servicios deben ser autónomos**: Los servicios deben tener su propio entorno de ejecución. De esta manera nos aseguramos que es reutilizable desde el punto de vista de la plataforma de ejecución.
- **Los servicios no deben tener estados:** Los servicios no deben guardar ningún tipo de información ya que una aplicación está constituida por un conjunto de servicios, lo que implica que si un servicio guarda algún tipo de información puede producir problemas de inconsistencia de datos. La solución a esto es que los servicios solo tengan lógica y que la información esté almacenada en algún sistema de información.
- **Los servicios deben poder ser descubiertos**: Todo servicio debe ser publicado para evitar la creación accidental de servicios que proporcionen la misma funcionalidad.

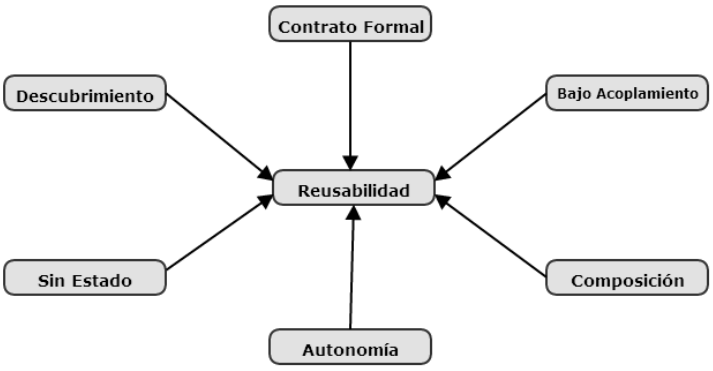


Figura 2.2. Principios de Orientación a Servicios [9]

2.2. SoaML

Es un lenguaje de modelado que se utiliza para la especificación de SOA. Este puede ser integrado en herramientas MDA para diseñar este estilo de arquitectura. Es una extensión UML, en la cual se permite la construcción de contratos, servicios, consumidores, proveedores,

capacidades, mensajes y demás conceptos de SOA. A continuación se especifican brevemente los elementos del metamodelo de SoaML según la especificación de la OMG [3].

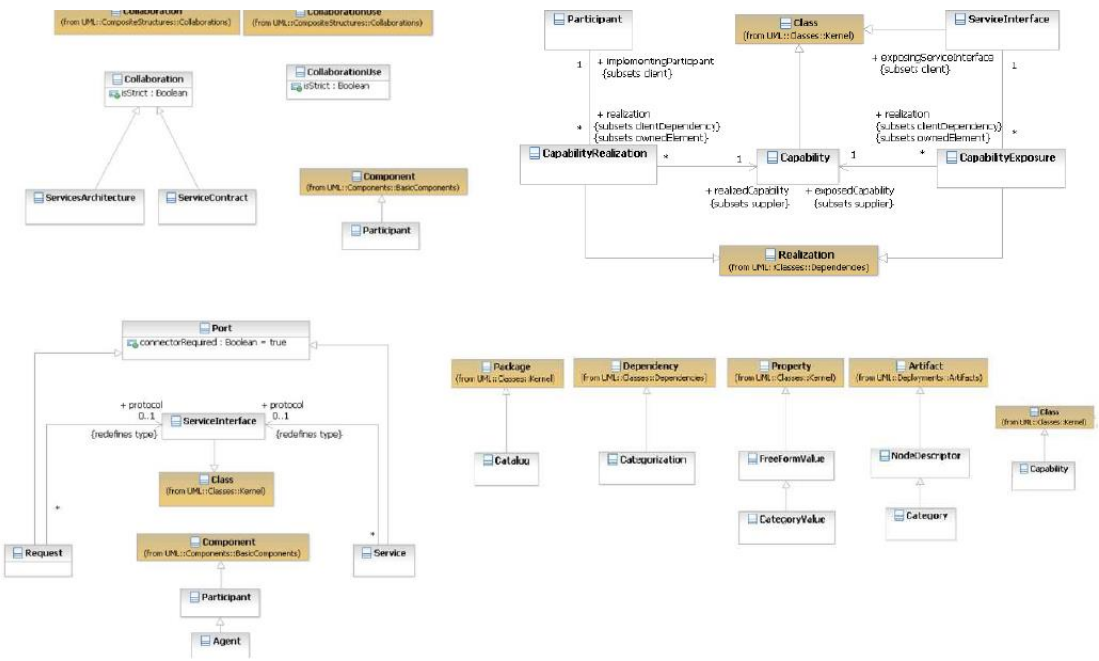


Figura 2.3. Perfil SoaML [3]

Services Architcture

Proporciona la vista de mayor nivel de abstracción de una arquitectura orientada a servicios, define un conjunto de participantes o capacidades que interactúan entre sí brindando y utilizando servicios que son mostrados en esta vista mediante los contratos de servicio, para lograr algún propósito. Cada participante que interviene en un servicio, lo hace bajo alguno de los roles que necesita el servicio particular para poder ser ejecutado. Un mismo participante de la arquitectura puede tomar distintos roles, en los distintos servicios en los cuales participa.

Service Contract

Especifica un acuerdo entre productores y consumidores del servicio que define. Dentro de los términos de este acuerdo se encuentran productos, información, obligaciones y valores que serán intercambiados entre ellos.

Participant

Un participante puede tanto consumir o proveer un servicio. Para brindar un servicio se deben especificar las operaciones que están involucradas en el mismo y opcionalmente se debe mostrar el protocolo de utilización del servicio. Esto último se logra agregando un componente de UML2 llamado “ownedBehavior” que muestra al participante (consumidor, proveedor) la forma de resolverla (implementación del método o forma de consumirla). Para ilustrar este comportamiento se puede utilizar cualquier diagrama de comportamiento de UML (actividad, estado, interacción, colaboración).

Agent

Son instancias de participantes, es decir son participantes activos en un sistema SOA. Los agentes representan componentes concretos que proveen y consumen servicios.

Capability
Extiende la metaclase Class de UML y se representa utilizando una clase o un componente con el estereotipo «capability». Especifican un conjunto de funciones o recursos que un servicio -provisto por uno o más participantes- ofrece.

Expose
Extiende la metaclase dependencia de UML y se denota con un estereotipo «Expose». Es una dependencia entre una Capacidad y una Interfaz de Servicio, usada para indicar que la Interfaz expone dicha Capacidad -provista o requerida por un Participante-. Una Interfaz expone una capacidad si provee operaciones e información consistente con dicha capacidad -no es necesario que la interfaz tenga las mismas operaciones y propiedades que la capacidad-. Las capacidades son expuestas por una interfaz de servicio, para que ésta pueda ser usada como tipo para los puertos mediante los cuales se accede a estas capacidades.

Servicio
Este concepto no tiene asociado un estereotipo, pero es un elemento central que ofrece sus capacidades a otras partes mediante un contrato. Debe existir un contrato para cada servicio donde se definen términos y condiciones para que los participantes interactúen al utilizar el servicio. El proveedor (debe incluir un puerto de tipo Service) es el participante que ofrece el servicio a otro que lo requiere que es el consumidor (debe incluir un puerto de tipo Request).

Service Interfaces
Las interfaces de servicio definen el tipo de un puerto (service o request). Éstas pueden especificar servicios bidireccionales -tanto el consumidor como el proveedor tienen responsabilidades de enviar y recibir mensajes y eventos-. La interfaz de servicio realiza las interfaces de los proveedores y utiliza la de los consumidores del servicio. Se usan para exponer las capacidades que queremos ofrecer en nuestro sistema.

Provider
Para el caso de un Contrato de servicio compuesto, extiende la metaclase Class. Para otro caso extiende la metaclase Interface. Se denota con el estereotipo «Provider». En el contexto de SOA, un consumidor pide un servicio a un proveedor que utiliza sus capacidades para satisfacer dicho pedido. El elemento Provider, modela la interfaz provista por el proveedor del servicio y es usada como tipo para el rol proveedor del Contrato del Servicio que gobierna la interacción antes mencionada -por lo tanto es usada también como tipo del puerto del participante que provee dicho servicio-. En particular, la interfaz del proveedor define el servicio desde la perspectiva del proveedor, representando las operaciones y señales que éste va a recibir durante la interacción del servicio. En el caso de una comunicación bidireccional el proveedor va a tener una dependencia de uso con la interfaz del consumidor.

Consumer
Para el caso de un Contrato de servicio compuesto, extiende la metaclase Class. Para otro caso extiende la metaclase Interface. Se denota con el estereotipo «Consumer». Análogamente al caso de Provider, Consumer modela la interfaz provista por el consumidor de un servicio y define al servicio desde la perspectiva del consumidor.

Message Type
Extiende las clases DataType , Class y Signal de UML2 y se denota con la palabra clave «MessageType». El atributo encoding específica la codificación del payload del mensaje. Representa la información intercambiada -datos pasados o retornados en la invocación de operaciones definidas en una interfaz de servicio- entre los participantes que requieren los servicios y los servicios mismos.

Port

Extiende del Port común UML, al cual se le agrega el tagged value de tipo boolean connectionRequired, que por defecto tiene el valor verdadero. Este atributo indica si para el puerto en cuestión es necesario que esté conectado o no a otro componente para ofrecer un servicio o para consumirlo. El tipo de un puerto puede ser una interfaz o una interfaz de servicio. Si es una interfaz de servicio, la interacción puede ser bidireccional, por lo que aparecen dos símbolos en el puerto si es necesario (círculo que implementa la interfaz, pinza que necesita una interfaz).

2.3. Model Driven Architecture (MDA)

En esta sección se describe MDA a partir de la especificación obtenida en [6]. MDA es una implementación del paradigma MDD, basado en las tecnologías estandarizadas por la OMG. Un modelo se puede definir como un conjunto de elementos ordenados en forma coherente con el fin de describir algo para su posterior análisis. MDA concibe la construcción de modelos de software, a distintos niveles de abstracción, como artefactos principales en el desarrollo de software. De esta manera, el diseño de los sistemas está orientado a modelos, lo cual permite una alta flexibilidad en la implementación, integración, mantenimiento, prueba y simulación de los sistemas.

Una de las ideas principales por la que surge MDA es separar la especificación de los sistemas de los detalles de su implementación en una determinada plataforma. Para ello provee un conjunto de herramientas para especificar un sistema independientemente de la plataforma de implementación, especificar dichas plataformas, elegir una determinada plataforma para el sistema, y transformar las especificaciones de los sistemas a la plataforma elegida. Por lo que podemos concluir que las metas fundamentales son: *portabilidad, interoperabilidad y reutilización*. En una visión a futuro, si se logra desarrollar lo suficiente esta visión, se evitaría la creación de código "desde cero" (from scratch), permitiendo una programación a alto nivel a partir de la especificación del problema a resolver, en forma de modelos. La iniciativa MDA cubre un amplio espectro de elementos que describen los distintos conceptos a lo largo del documento, a modo de ejemplo, dichas áreas serian: metamodelos basados en MOF, perfiles UML, transformaciones de modelos, definición de lenguajes de transformación (QVT), construcción de modelos PIM y PSM y transformaciones entre ellos, construcción de herramientas de soporte, aplicación en métodos de desarrollo y en dominios específicos, etc. Algunos de estos aspectos están bien fundamentados y se están empezando a aplicar con éxito, otros sin embargo están todavía en proceso de definición. En este contexto son necesarios esfuerzos que conviertan MDA y sus conceptos y técnicas relacionados en una aproximación realizable, basada en estándares abiertos, soportada por técnicas y herramientas maduras.

Ideas centrales en MDA

Se plantean dos ideas principales en MDA. La primera de ellas separar el diseño de la arquitectura y de las tecnologías de construcción, facilitando que el diseño y la arquitectura puedan ser alterados independientemente. Por otro lado, controlar la evolución desde modelos abstractos a implementaciones tendiendo a aumentar el grado de automatización.

Modelos MDA

La propuesta MDA especifica tres puntos de vista sobre un sistema, *independiente de la computación, independiente de la plataforma y el específico de la plataforma*, en base a los cuales se definen los tipos de modelos.

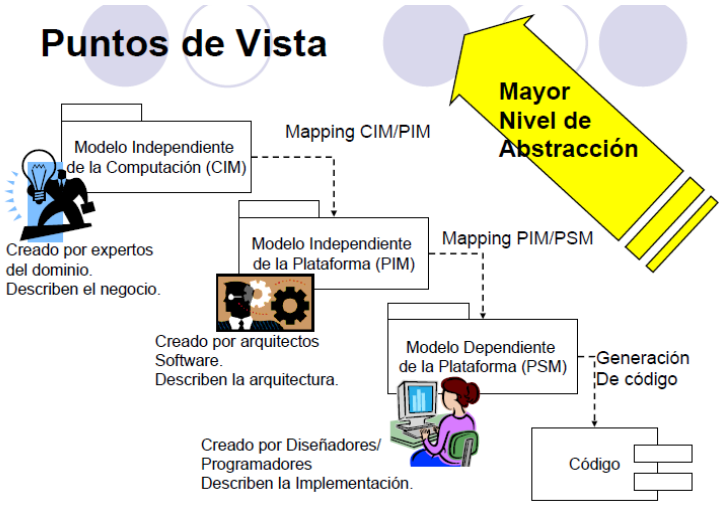


Figura 2.4. Puntos de vista MDA [6]

El punto de vista independiente de la computación se enfoca en el ambiente del sistema y los requerimientos del mismo, es decir, la lógica y requerimientos del negocio. Por otra parte, el independiente de la plataforma se enfoca en el funcionamiento del sistema mientras oculta los detalles específicos para cierta plataforma. Finalmente el punto de vista específico de la plataforma combina el punto de vista independiente de la plataforma con un enfoque específico detallando la utilización de una plataforma específica por un sistema.

Tipos de modelos

MDA describe distintos tipos de modelos para el diseño de software, en base a los puntos de vista planteados.

Un tipo de modelo es CIM (Computation Independent Model, CIM), el cual describe la lógica de negocio desde una perspectiva independiente de la computación. Estos modelos son realizados por expertos en el dominio funcionando como nexo entre los expertos en el dominio y los diseñadores. La finalidad del mismo es comprender el problema y proporcionar un vocabulario común para su uso en otros modelos. PIM (Platform Independent Model, PIM) es otro tipo de modelo donde se da una descripción de la funcionalidad del sistema, de forma independiente de las características de plataformas de implementación específicas. Esto permite que pueda ser utilizado en distintas plataformas de tipo similar. PSM (Platform Specific Model, PSM) es el tipo de modelo donde se describe el sistema en términos de una plataforma específica, determinando cómo el sistema utiliza los elementos definidos por ésta. Por último, el ISM (Implementation Specific Model, ISM) es una especificación del sistema a nivel de código, como pueden ser Java y C#, entre otros.

En Figura 2.5 se puede ver un ejemplo de un sistema de software desde el enfoque de los modelos MDA.

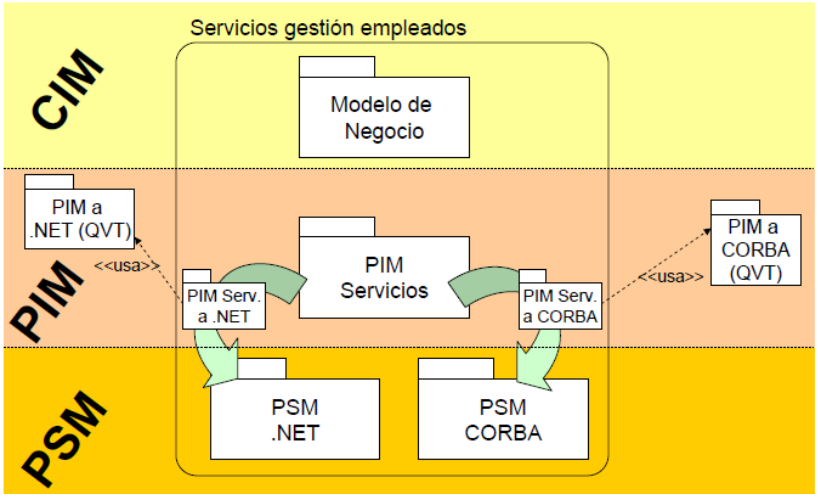


Figura 2.5. Ejemplo de tipos de modelo MDA [6]

Funcionamiento de MDA

Podemos observar que el ciclo de vida MDA no parece muy diferente del ciclo de vida tradicional. Si bien se podrían identificar las mismas fases, una de las diferencias principales reside en la naturaleza de los artefactos que se crean durante el proceso de desarrollo. Los artefactos son los modelos formales, es decir, modelos que puedan ser comprendidos por los computadores.

MDA define los modelos PIM, PSM y el código, así como la manera de relacionarse unos con otros. Se crean los modelos PIM, luego se transforman en uno o varios PSM (el paso más complejo en los desarrollos MDA) y finalmente se transforma en código. La novedad de MDA frente al desarrollo tradicional, es que las transformaciones se pueden hacer mediante herramientas que se ejecutan de forma automática. En concreto, el mayor aporte y beneficio de MDA, es la transformación de modelos PIM a modelos PSM. Las transformaciones, aunque es deseable que se realicen de forma automática mediante herramientas, no siempre se pueden realizar, pero la idea de las transformaciones automáticas va a permitir a los desarrolladores tener retroalimentación de forma rápida de un modelo PIM, ya que va a poder generar prototipos de forma inmediata. Con este planteamiento, el desarrollo se focaliza en la definición de los modelos PIM, ya que tanto los PSM, como el código se van a generar mediante transformaciones. Si bien la transformación requiere esfuerzo, la ventaja es que haciéndola una vez, se puede aplicar en muchos sistemas. Los desarrolladores definen mejor sus modelos porque se aíslan de los detalles técnicos, lo que permite centrarse exclusivamente en los detalles del negocio concreto, obteniendo resultados en menos tiempo. Por otro lado las trasformaciones, generarán tanto el PSM como el código, sin perder ningún detalle técnico, ya que todos ellos quedan definidos en las transformaciones.

Cualquier persona será capaz de leer y comprender los modelos que a su vez sirven como punto de partida para la generación automática del código. Así mismo, los modelos PIM no se abandonan al comenzar a codificar, todo lo contrario, cada vez que se modifican los requisitos del sistema, no se va a modificar el código, como ocurría en el enfoque tradicional, si no que se va a modificar directamente el PIM y se podrían regenerar desde éste el PSM y el código. Esta es la teoría, ya que los cambios en el PIM se suelen hacer sólo cuando existen cambios de requisitos de cierta relevancia, debido a que las transformaciones no son siempre automáticas.

En el caso de tener que migrar un sistema a otra tecnología, solo será necesario tomar el PIM y generar el PSM apropiado para la nueva plataforma, lo que da la característica de portabilidad a MDA.

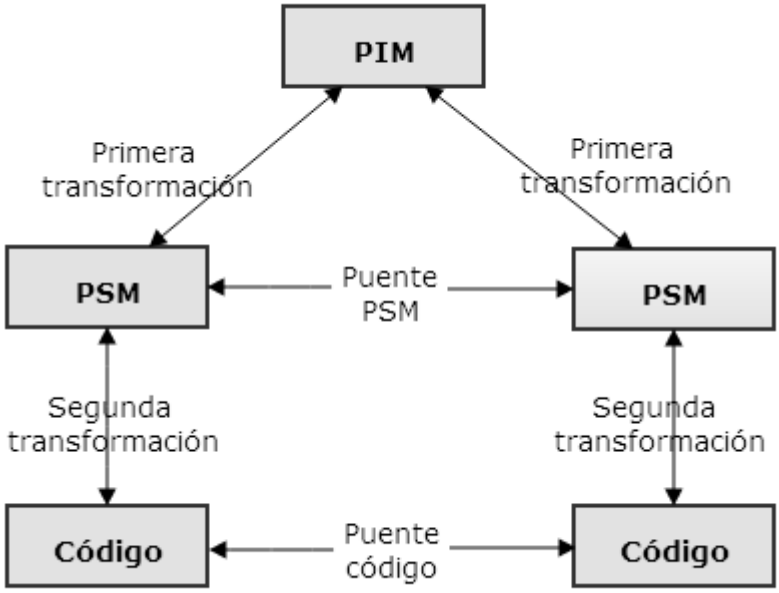


Figura 2.6. Interoperabilidad MDA mediante puentes [6]

El conjunto resultante de PSM, no estará directamente comunicados entre ellos. Para conseguir la interoperabilidad, hay que transformar conceptos de una plataforma en los de otra, construyendo lo que en terminología MDA se llaman puentes (bridges).

2.4. XMI: XML Metadata Interchange

En esta sección se describe XMI a partir de la especificación obtenida en [10]. Un metamodelo es un modelo que permite describir un lenguaje de modelado con un mayor nivel de abstracción al que presenta el lenguaje. El metamodelado permite definir modelos o lenguajes que se ajusten a dominios específicos. Para esto, se deben representar las principales características del lenguaje que según [11] son, la sintaxis concreta, la sintaxis abstracta y la semántica.

Una arquitectura de metamodelado, describe una jerarquía de clasificación en la que las entidades del modelo se ven como instancias de las clases definidas en un nivel superior. La OMG define una arquitectura [12] cuya jerarquía tiene cuatro niveles: M0 corresponde a las instancias, M1 corresponde al modelo del sistema en sí, M2 corresponde al metamodelo que describe el lenguaje y M3 es el meta metamodelo que describe al meta modelo. Los elementos de un nivel son instancias de un elemento del nivel superior. El lenguaje de metamodelado estándar propuesto por la OMG, es el MOF (Meta Object Facility, MOF) que es el utilizado para describir entre otros el metamodelo de UML. En la Figura 2.7 se muestra un diagrama de la arquitectura MOF.

XMI es otro estándar OMG que permite el intercambio de modelos definidos mediante MOF entre diferentes herramientas. Permite expresar en XML cualquier modelo o metamodelo que se haya definido en MOF. Se utiliza en herramientas de integración, repositorios, aplicaciones y almacenes de datos, ya que XMI proporciona reglas por las que permite expresar en XML cualquier modelo o metamodelo que se haya definido en MOF [OMG 2007c]. Mediante el estándar XMI, se define cómo se deben crear los documentos y esquemas XML. Cada documento XML que defina un modelo debe contener los elementos requeridos en la especificación XMI, los elementos que contienen los datos del modelo representado y, opcionalmente, elementos que contienen metadatos que representan extensiones del

modelo. Determinada información del modelo puede ser codificada en esquemas XML que permitan realizar validaciones de un documento XMI. Mediante los esquemas de XML se puede validar sobre todo la sintaxis de los elementos del modelo que se definen en el documento y algunos aspectos de la semántica. También se pueden validar extensiones del modelo ya que éstas se definen como elementos del modelo y por lo tanto pueden estar incluidas en el esquema como parte de los elementos a verificar.

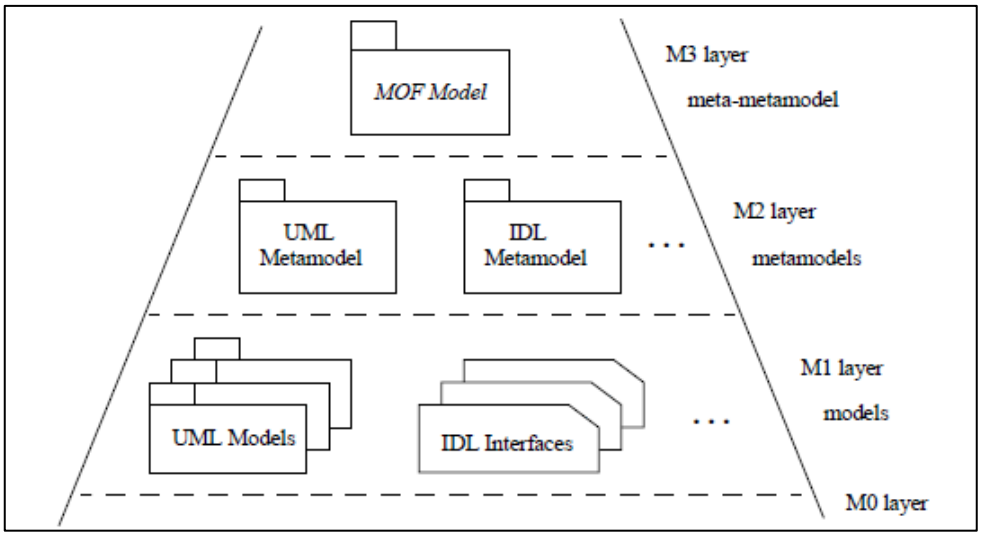


Figura 2.7. Arquitectura MOF [12]

XMI = UML + MOF + XML

XMI es un formato de intercambio de metadatos independiente de cualquier plataforma que proporciona interoperabilidad entre herramientas. Además permite la serialización de modelos y metamodelos MOF en XML.

Arquitectura de XMI

La arquitectura de XMI simplifica la comunicación entre diferentes aplicaciones y potencia la reutilización de objetos y componentes. Se especifica un DTD[3] (Document Type Definition, DTD) de XMI para UML que permite el intercambio de modelos UML de acuerdo con el metamodelo UML. Uno de los principales objetivos de este DTD es permitir la interoperabilidad entre herramientas de modelado OO (Orientado a Objetos, OO) consiguiéndolo con XMI. Cada DTD usado por XMI debe satisfacer como requisito que todos los elementos XML definidos por la especificación XMI deben ser declarados en el DTD. Además cada metamodelo construido (clase, atributo y asociación) debe tener una correspondiente declaración de elemento. Todos los elementos XML que representan extensiones hacia el metamodelo deberían ser declarados en el DTD interno o externo.

En la Figura 2.8 se muestra un ejemplo donde se quiere representar una realidad mediante el diseño de un documento XMI, en el cual se tiene un objeto “Auto” que es de determinado color y tiene cierta cantidad de puertas.

[3] DTD brinda una descripción de estructura y sintaxis de un documento, permite usar una estructura común y mantener la consistencia entre todos los documentos que utilicen la misma DTD.

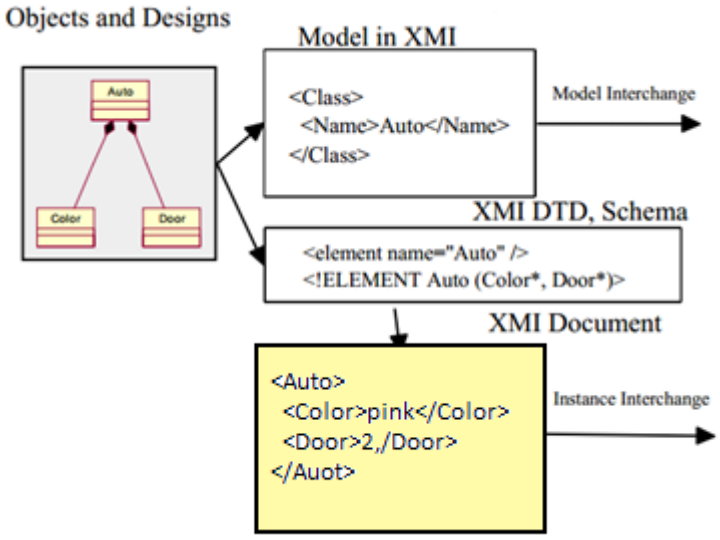


Figura 2.8. Ejemplo XMI para un modelo [10]

La definición de los documentos XMI posee un proceso de validación, mediante el cual se verifica la correctitud de dicho documento tanto en forma semántica, sintáctica y de su correcta formación. En la Figura 2.9 se muestra un esquema que representa el proceso de validación de XMI.

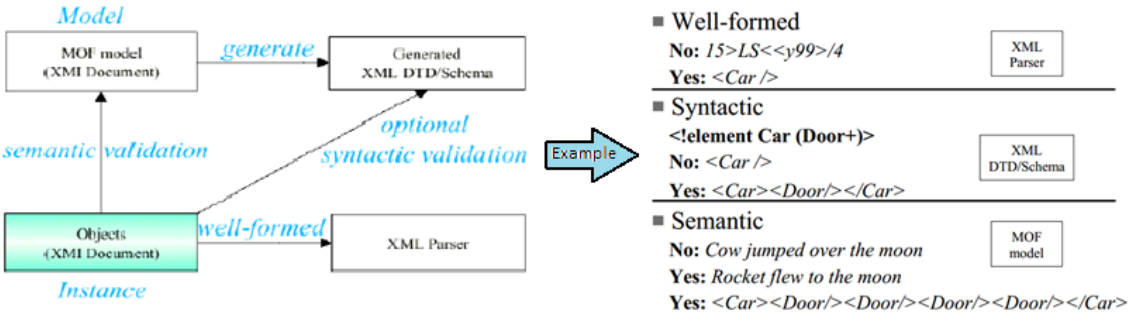


Figura 2.9. Proceso de validación de XMI [10]

2.5. Tecnología Web Services

Sin perjuicio a otras definiciones, vamos a utilizar la definición proporcionada por W3C: *[Definición: Un Web service o servicio Web es un sistema de software diseñado para apoyar la interoperabilidad entre diferentes aplicaciones de software, que se ejecuta en una variedad de plataformas y / o marcos. Tiene una interfaz descrita en un formato procesable por máquina (específicamente WSDL). Otros sistemas interactúan con el servicio Web en la forma prescrita por su descripción utilizando mensajes SOAP, típicamente transportados utilizando HTTP con una serialización XML en conjunto con otras relacionadas con los estándares Web.]* [13]

Conceptos de Web Services

En esta sección se describen los conceptos relacionados con Web Services explicando su interacción y funcionamiento.

Un Web Service es una noción abstracta que debe ser implementada por un agente concreto. El agente es la pieza concreta de hardware o software que envía y recibe mensajes, mientras que el servicio es el recurso que se caracteriza por el conjunto abstracto de la funcionalidad que se proporciona. Es posible implementar un Web Service en particular mediante distintos agentes manteniendo la misma funcionalidad [13].

El propósito de un Web Service es proporcionar alguna funcionalidad. La entidad proveedor es la persona u organización que proporciona un agente apropiado para implementar un servicio en particular. La entidad solicitante es una persona u organización que desee hacer uso del Web Service del proveedor. Se utilizará un agente solicitante para intercambiar mensajes con el agente proveedor. En la mayoría de los casos, el solicitante es el que inicia este intercambio de mensajes, aunque no siempre. Sin embargo, para mantener la coherencia, se le denomina con el término "agente solicitante" al agente que interactúa con el agente proveedor, incluso en los casos en que el proveedor inicia el intercambio. Para que este intercambio de mensajes pueda tener éxito, la entidad solicitante y la entidad prestadora debe primero ponerse de acuerdo sobre la semántica y la mecánica del intercambio de mensajes.

Para el intercambio de mensajes entre los agentes es necesaria una documentación que describa el Web Service. Esta documentación se conoce como WSD, la cual es una especificación de la interfaz del Web Service y debe ser interpretada por un software. El WSD es escrito en formato WSDL. El concepto de WSDL será desarrollado más adelante en esta misma sección.

Al momento del intercambio entre agentes debe estar especificada la semántica del Web Service. Dicha semántica define la expectativa compartida sobre el comportamiento del servicio, es el "contrato" entre la entidad solicitante y la entidad prestadora con respecto al propósito y las consecuencias de la interacción. De todos modos no es necesariamente por escrito o explícitamente negociado.

En la Figura 2.10 se muestra un esquema de los conceptos anteriormente mencionados y su interacción. Las entidades del solicitante y el proveedor se deben conocer entre sí, o al menos una saber de la otra (1). El solicitante y el proveedor están de acuerdo en la descripción del servicio y la semántica que gobiernan la interacción entre ellos (2). La descripción del servicio y la semántica son realizadas por el solicitante y el proveedor (3). El solicitante y los agentes proveedores intercambian mensajes (4).

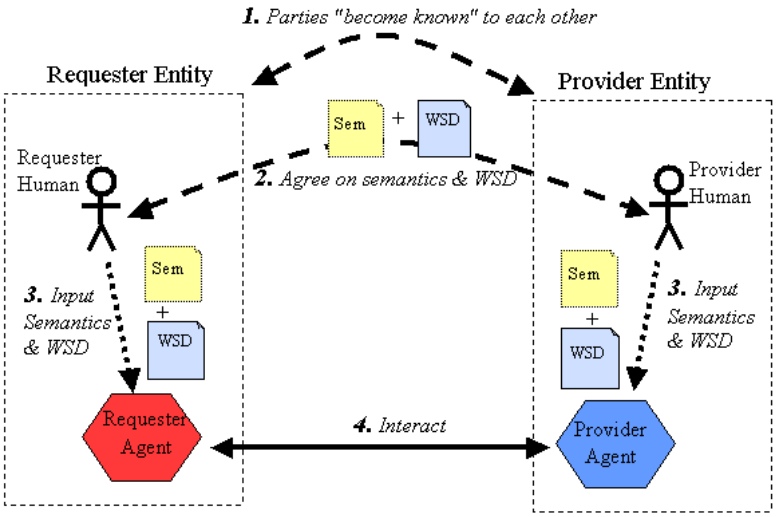


Figura 2.10. Proceso general de interacción de un Web Service [13]

Arquitectura de Web Services

La arquitectura de Web Services implica muchas tecnologías superpuestas e interrelacionadas. Hay muchas maneras de visualizar estas tecnologías, así como hay muchas maneras de construir y utilizar los Web Services. En la Figura 2.11 se proporciona un ejemplo de algunas de estas tecnologías.

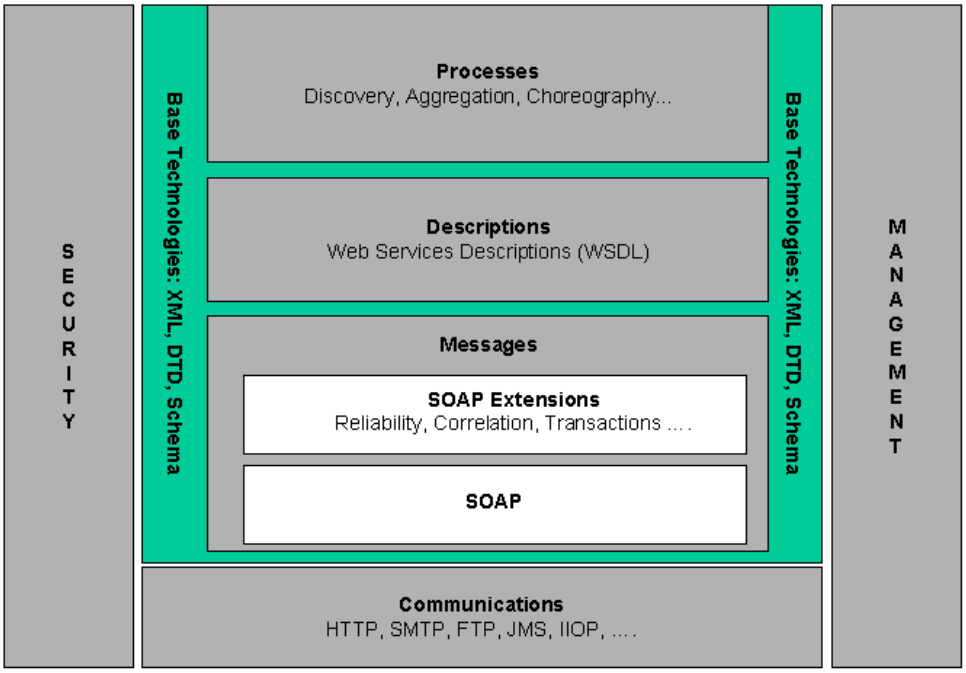


Figura 2.11. Arquitectura de Web Services [13]

A continuación describimos algunos de los componentes relevantes de la arquitectura de un Web Service.

Simple Object Access Protocol, SOAP

SOAP es un protocolo basado en XML que permite la interacción de múltiples objetos en diferentes procesos y tiene la capacidad de trasmitir información compleja. Los datos pueden ser trasmitidos a través de distintas tecnologías mostradas en la capa de Communications del ejemplo (HTTP[4], SMTP[5], etc.). SOAP específica el formato de los mensajes, los cuales están formados por un envelope o sobre cuya estructura a su vez está compuesta por un header o cabecera y un body o cuerpo. En la Figura 2.12 se muestra la estructura anteriormente mencionada [28].

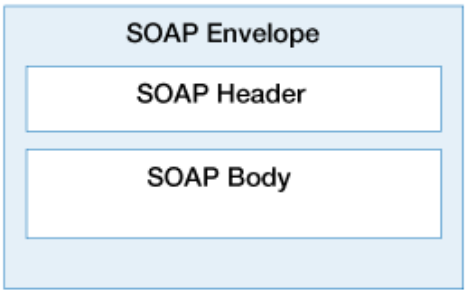


Figura 2.12. Estructura del mensaje SOAP [28]

Otro protocolo de mensajería que sería una alternativa a SOAP es REST.

Representational State Transfer, REST

REST es una técnica de arquitectura de software para sistemas distribuidos y es utilizado para describir cualquier interfaz web que utilice XML y HTTP. Una implementación concreta de un Web Services REST sigue cuatro principios de diseño fundamentales. Éstos son, (1) la utilización de los métodos HTTP de manera explícita siguiendo el protocolo definido por la RFC

[4] HyperText Transfer Protocol - http://www.w3.org/Protocols/
[5] Simple Mail Transfer Protocol - http://www.ietf.org/rfc/rfc2821.txt

2616[6], (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 URI[7]s con forma de directorio, las mismas 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[8] o ambos [29].

Las aplicaciones que siguen los principios mencionados anteriormente son llamadas RESTful.

Otro de los componentes importantes que forma parte de la arquitectura de Web Services y se encuentra en un nivel superior al de SOAP es la descripción del servicio mediante WSDL.

Web Services Description Language, WSDL

Es un protocolo basado en XML que describe los accesos a un Web Service. Se podría decir que funciona como un manual ya que indica las interfaces que provee el Web Service. En un WSDL se definen los formatos del mensaje, tipos de datos, protocolos de transporte y formatos de serialización de transporte que deben ser utilizados entre el agente solicitante y el agente proveedor. También especifica una o más ubicaciones de red en la que puede ser invocado un agente proveedor, y puede proporcionar alguna información sobre el patrón de intercambio de mensajes que se espera. En esencia, la descripción del servicio representa un acuerdo que regule los mecanismos de interacción con ese servicio [16].

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 2.13 se muestra un esquema de la estructura de los componentes abstractos.

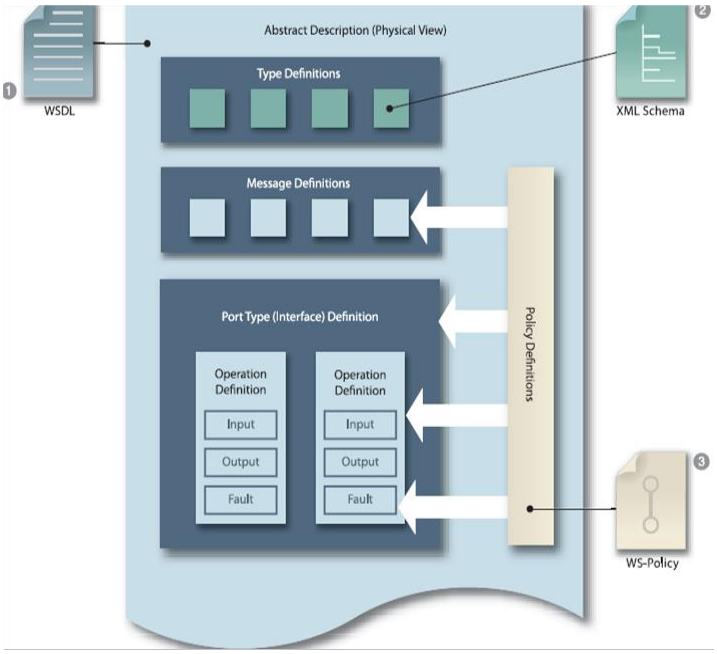


Figura 2.13. Descripción de los componentes abstractos en un WSDL [16]

[6] RFC 2616 - http://tools.ietf.org/html/rfc2616

[7] Uniform Resource Identifier, es una cadena de caracteres corta que identifica inequívocamente un recurso.

[8] JavaScript Object Notation - http://www.json.org/

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).

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 2.14.

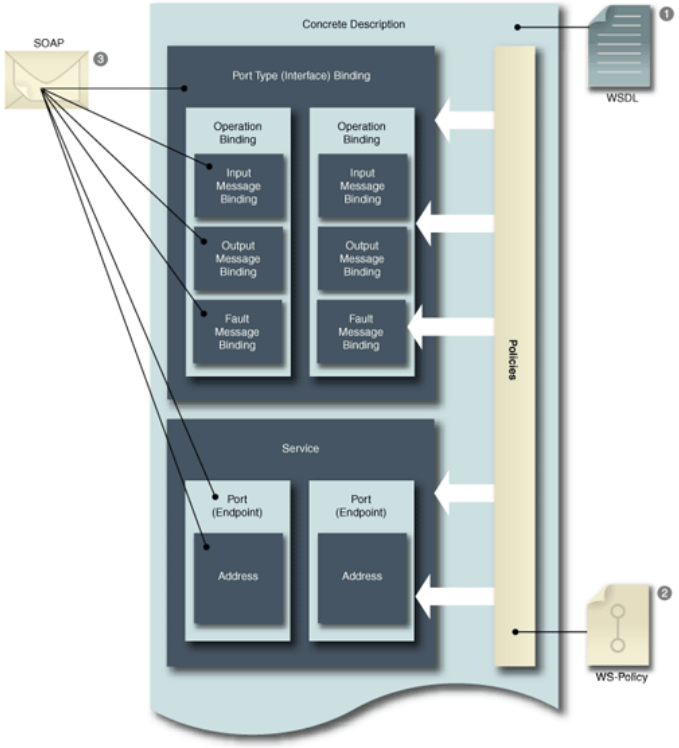


Figura 2.14. Descripción de los componentes concretos en un WSDL [16]

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.

Service

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.

Implementación de Web Services

Para la implementación de Web Services existen dos enfoques, Top Down y Bottom Up. Top Down es la estrategia de crear el Web Service a partir de un archivo WSDL, mientras que para el enfoque Bottom Up se comienza por el código y se genera a partir de éste el WSDL correspondiente [30].

Tabla 2.1. Ventajas y desventajas del enfoque Top Down.

Top Down	
Ventajas	**Desventajas**
Permite el desarrollo en paralelo del cliente y el servicio.	Requiere conocimientos de construcción de un documento WSDL.
Es simple al momento de realizar cambios.	

Tabla 2.2. Ventajas y desventajas del enfoque Bottom Up.

Bottom Up	
Ventajas	**Desventajas**
No requiere conocimientos sobre la construcción de un documento WSDL.	Los cambios en interfaces son difíciles de mantener.
Al tener el código escrito, se exponen los servicios rápidamente.	

2.6. Plataforma Java EE

A continuación se presenta la plataforma Java EE, haciendo hincapié en su arquitectura y componentes relevantes a este proyecto.

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. La definición de Sun Microsystems es*: [Java Platform, Enterprise Edition 5 (Java EE 5) define el estándar para el desarrollo de aplicaciones empresariales distribuidas, basadas en componentes, utilizando un modelo de múltiples capas.]*[14]

Java EE ofrece un conjunto de especificaciones y técnicas que proporcionan soluciones completas, seguras, estables y escalables para el desarrollo, despliegue y gestión de aplicaciones de múltiples niveles de funcionalidad basadas en servidores. Se reduce el costo y complejidad de desarrollo, lo cual resulta en servicios que se pueden desplegar y extender fácilmente. 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.

Existen actualmente varias versiones de Java EE, las cuales son:

- Java EE 5: simplificó el desarrollo de componentes de la capa de negocio.
- Java EE 6: aplica las mismas ideas para simplificar desarrollo de la capa web.
- Versiones anteriores: Java EE 1.4 y anteriores.

Arquitectura Java EE

La plataforma Java EE utiliza un modelo de aplicación distribuida de varios niveles, la lógica de la aplicación se divide en los componentes según su función. Los diversos componentes de

aplicaciones que componen una aplicación Java EE se instalan en diferentes máquinas. La Figura 2.15 muestra dos aplicaciones Java EE divididas en los niveles que se describen en la siguiente lista.

- Los componentes a nivel de cliente se ejecutan en la máquina cliente.
- Los componentes de nivel Web se ejecutan en el servidor Java EE.
- Los componentes de niveles de negocios, se ejecutan en el servidor Java EE.
- El sistema de información de la empresa (EIS) se ejecuta en el servidor EIS.

A pesar de que una aplicación Java EE puede consistir en uno de los tres o cuatro niveles mostrados en la Figura 2.15, generalmente constan 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.

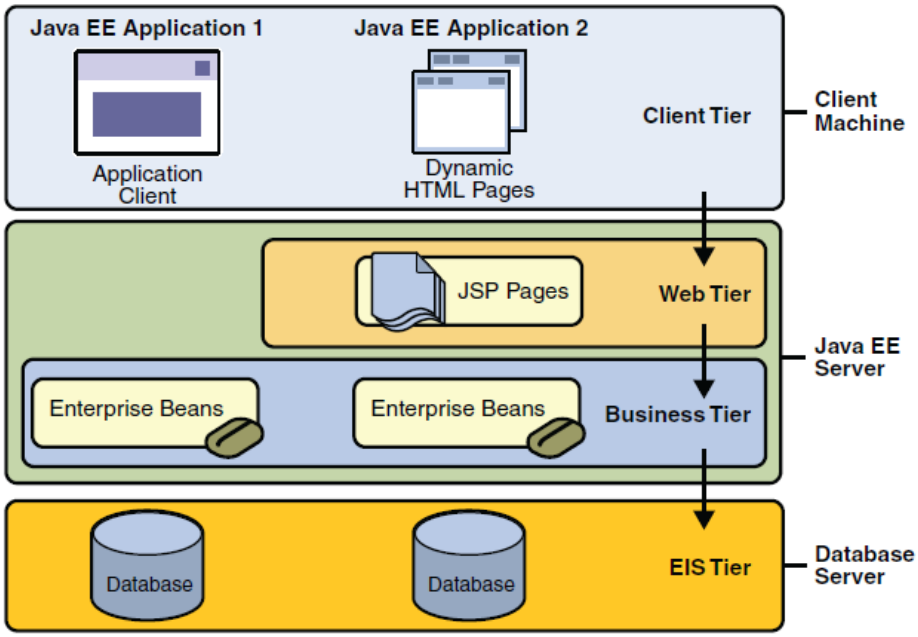


Figura 2.15. Aplicaciones multinivel [22]

Soporte a Web Services

Para el soporte de Web Services la plataforma Java EE proporciona JAX-WS (Java API for XML Web Services, JAX-WS), una API para construir Web Services y clientes de éstos, los cuales se comunican usando XML [14].

En JAX-WS, la invocación a una operación de un Web Service se realiza mediante protocolos basados en XML como SOAP. La comunicación en SOAP es mediante mensajes SOAP trasmitidos sobre HTTP, JAX-WS abstrae de la complejidad de estos mensajes al desarrollador de aplicaciones. Una característica que presenta JAX-WS es que permite la independencia de plataformas, es decir, el cliente y el Web Service pueden estar implementados en diferentes lenguajes de programación. JAX-WS delega el mapeo entre tipos del lenguaje Java y XML a JAXB (Java Architecture for XML Binding, JAXB).

JAXB tiene como características principales la capacidad de serializar y deserializar las referencias de objetos Java a XML y viceversa [14].

Framework Spring

Spring es un framework de código abierto para el desarrollo de aplicaciones en la plataforma Java. Es usado por más de un millón de desarrolladores, pudiéndose encontrar gran cantidad de documentación sobre el mismo, lo que proporciona un buen soporte para el usuario. Spring

presenta un diseño modular que permite la adopción gradual de cada módulo individual a medida que estos sean necesarios y además facilita el uso de las tecnologías para el acceso a base de datos y puede ser integrado mediante un plug-in al IDE de desarrollo Eclipse. Uno de los módulos que brinda dicho framework, es el proyecto Spring Web Services el cual es usado para el desarrollo de Web Services. El mismo, ofrece soporte para la exposición de Web Services SOAP bajo la API JAX-WS y de Web Services REST [31].

2.7. 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 [16]. Es una arquitectura basada en plug-ins, un plug-in es una unidad mínima de funcionalidad que permite escribir nuevas extensiones incrementando de este modo las funcionalidades provistas por el ambiente. Para esto, cada plug-in Eclipse puede ofrecer puntos de extensión que permiten la posibilidad de agregar nuevas funcionalidades. En la Figura 2.16 se presenta un esquema de la arquitectura descripta.

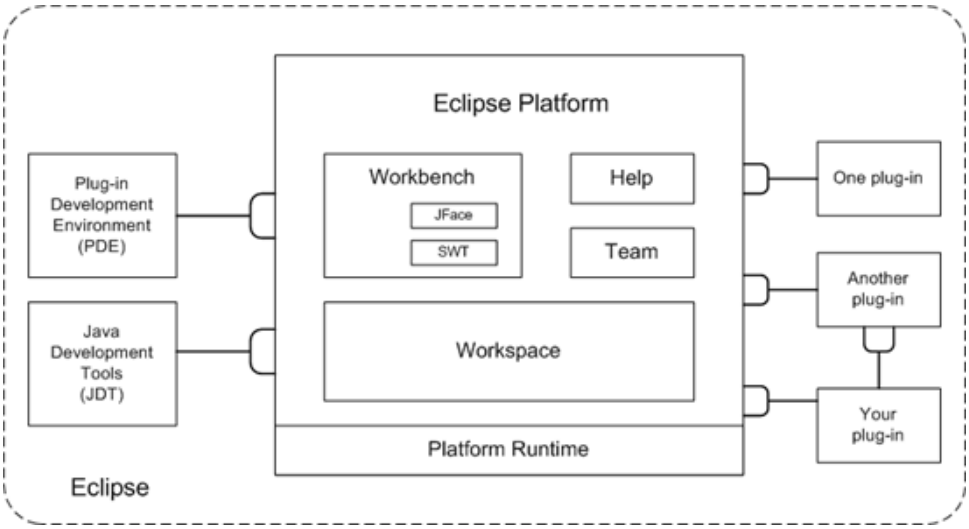


Figura 2.16. Arquitectura Eclipse [16]

Implementación de Web Services dentro de Eclipse

Dentro de la plataforma Eclipse existen dos tipos de proyectos que permiten la implementación de Web Services. Éstos son, Dynamic Web Project y EJB Project. El Dynamic Web Project permite exponer Web Services mediante una clase Java que se ejecuta en el contenedor Web, mientras que el EJB Project permite exponer Web Services mediante un EJB session stateless que se ejecuta en un contenedor EJB. Por otro lado el Dynamic Web Project puede ejecutarse sobre un servidor de aplicaciones o un servidor Web, mientras que el EJB Project solamente sobre un servidor de aplicaciones [16].

2.8. Plug-in SoaML para Eclipse

El plug-in de Eclipse SoaML, realizado en el marco de un proyecto de grado en el año 2010, es un plug-in 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, la mayoría de las cuales son comerciales por lo que este plug-in 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 plug-in extiende las funcionalidades del plug-in 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.

3. Requerimientos y herramientas existentes

En este capítulo se presentan los requerimientos definidos para la construcción del generador de código a partir de modelos SoaML y la investigación sobre herramientas existentes para cumplir con los mismos. En la sección 3.1 se describen dichos requerimientos y en la sección 3.2 se expone el relevamiento realizado sobre las herramientas.

3.1. Descripción resumida de los Requerimientos

El plug-in SoaML2Code debe ofrecer la posibilidad de generar código ejecutable encapsulado en un Proyecto EJB o un Proyecto Web, partiendo de un proyecto de modelado SoaML contemplando todos los distintos componentes que lo representan. Para esto se identificaron los casos de uso que se ilustran en el diagrama de la Figura 3.1.

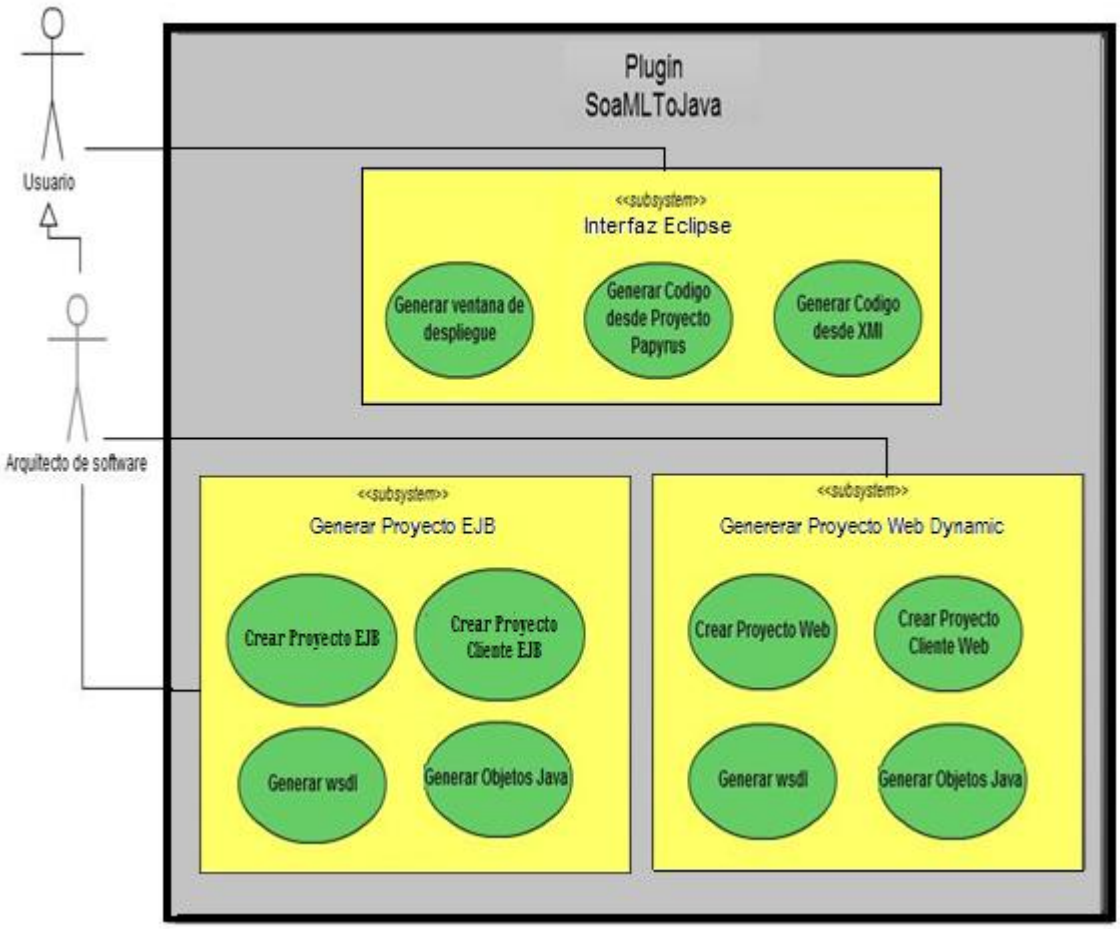


Figura 3.1. Diagrama de casos de uso

Para describir brevemente los requerimientos, se agruparán los casos de uso identificados en Generar Proyecto EJB, Generar Proyecto Web Dynamic, la generación de la ventana de despliegue y la interfaz sobre Eclipse para la generación de código. Todos ellos se describen a continuación.

Interfaz Eclipse

Eclipse brinda una interfaz de usuario donde se presentan todas las funcionalidades disponibles. Para cumplir con los requerimientos de nuestro proyecto fue necesario editar dicha interfaz para agregarle las funcionalidades necesarias. Bajo el nombre de Interfaz de Eclipse se agrupan los casos de uso que modifican esta interfaz.

Generar código desde proyecto Papyrus

Para generar el código en Java correspondiente a la implementación de un Web Service que represente un modelo SoaML, se debe permitir al arquitecto de software seleccionar desde la interfaz de Eclipse, el proyecto Papyrus que contenga dicho modelo.

Generar código desde XMI

Para generar el código en Java correspondiente a la implementación de un Web Service que represente un modelo SoaML, se debe permitir al arquitecto de software seleccionar desde la interfaz de Eclipse, un archivo XMI que represente dicho modelo.

Generar ventana de despliegue

Para generar el código en Java correspondiente a la implementación de un Web Service que represente un modelo SoaML, una vez escogido el modelo, se permite al arquitecto de software configurar desde una ventana de Eclipse el tipo de proyecto que desea crear para exponer el Web Service (Proyecto Web Dynamic o Proyecto EJB), el tipo de implementación del mismo (JAX-WS ri y JAX-WS + Spring) y si desea generar un cliente para dicho Web Service. Además puede indicar los elementos participantes del modelo que desee incluir en la generación, así como también la IP y el puerto donde estarán expuestos los servicios.

Generar Proyecto Web Dynamic

Dentro de los tipos de proyectos que se pueden generar para implementar un Web Service, se encuentra el proyecto Web Dynamic. Bajo el nombre de Generar Proyecto Web Dynamic, se encuentran todos los casos de uso que generan un Web Service con el tipo de proyecto mencionado.

Crear Proyecto Web

Permite al arquitecto de software crear un Proyecto Web Dynamic a partir del modelo SoaML previamente seleccionado, donde el proyecto generado contiene las clases Java que representan dicho modelo, exponiendo los servicios especificados en el mismo como Web Services.

Crear Proyecto Cliente

Permite al arquitecto de software crear un Proyecto Java que sirve como cliente Web Service para invocar los servicios que se exponen en el modelo SoaML previamente seleccionado.

Generar WSDL

Permite al arquitecto de software crear un Proyecto Web Dynamic a partir del modelo SoaML previamente seleccionado, donde el proyecto generado contiene un WSDL que expone los servicios indicados en el modelo SoaML escogido.

Generar Objetos Java

Permite al arquitecto de software crear un Proyecto Web Dynamic a partir del modelo SoaML previamente seleccionado, el sistema genera dentro del proyecto los objetos Java representados en el modelo SoaML a partir de los diagramas de Mensajes (elementos Message Type).

Generar Proyecto EJB

Dentro de los tipos de proyectos que se pueden generar para implementar un Web Service, se encuentra el proyecto EJB. Bajo el nombre de Generar Proyecto EJB, se encuentran todos los casos de uso que generan un Web Service con el tipo de proyecto mencionado.

Crear Proyecto EJB
Permite al arquitecto de software crear un Proyecto EJB a partir del modelo SoaML previamente seleccionado, donde el proyecto generado contiene las clases Java que representan dicho modelo, exponiendo los servicios especificados en el mismo como Web Services.

Crear Proyecto Cliente
Permite al arquitecto de software crear un Proyecto Java que sirve como cliente Web Service para invocar los servicios que se exponen en el modelo SoaML previamente seleccionado.

Generar WSDL
Permite al arquitecto de software crear un Proyecto EJB a partir del modelo SoaML previamente seleccionado, donde el proyecto generado contiene un WSDL que expone los servicios indicados en el modelo SoaML escogido.

Generar Objetos Java
Permite al arquitecto de software crear un Proyecto EJB a partir del modelo SoaML previamente seleccionado, el sistema genera dentro del proyecto los objetos Java representados en el modelo SoaML a partir de los diagramas de Mensajes (elementos Message Type).

3.2. Herramientas existentes de generación de código a partir de modelos SoaML

Se investigaron y estudiaron distintas herramientas disponibles, que a partir de modelos SoaML permiten generar código Java EE y Web Services comprobando que no hay muchas que cumplan con este requerimiento y la documentación sobre las mismas es escasa, dado que en la gran mayoría son herramientas comerciales. Las herramientas vistas son ModelPro e IBM Rational Application Developer. Sin embargo se investigó con mayor profundidad la herramienta ModelPro que es un proyecto de código abierto y sirvió como base para el desarrollo de este proyecto.

ModelPro

Es un proyecto de código abierto de ModelDriven.org, el cual sigue la visión MDA para proveer soluciones de software ejecutables sobre arquitecturas de negocio. El mismo parte de modelos SoaML y automatiza el diseño y desarrollo de arquitecturas orientadas a servicios, sobre las plataformas Java. ModelPro nos permite exponer servicios de forma rápida y eficiente en función de necesidades empresariales y procesos. Se basa en estándares de la industria y plataformas abiertas para que sus soluciones sean portables a través de tecnologías y nunca estén estrictamente acopladas a un proveedor o tecnología. El enfoque utilizado por MDA concibe la construcción de modelos de software, a distintos niveles de abstracción, por lo que el diseño de los sistemas está orientado a modelos, a partir de los cuales ModelPro nos permite utilizar los mismos como código fuente para múltiples tecnologías. Estos modelos MDA (y aplicaciones resultantes) se pueden modificar fácilmente, adaptarse y volver a generarse [15].

ModelPro se integra con la plataforma de desarrollo Eclipse para trabajar dentro de ésta. En la Figura 3.2 se muestra un diagrama de los componentes que interactúan en el proceso de generación de código del proyecto ModelPro.

El componente (1) UML Tool corresponde a los modelos SOA diseñados utilizando el estándar SoaML basado en UML (2). Los modelos provistos por (2) son utilizados en (3) por el SoaML Cartridge para automatizar la generación de código a partir de dichos modelos. El componente (4) está formado por las plataformas que utilizan el código generado para realizar una aplicación.

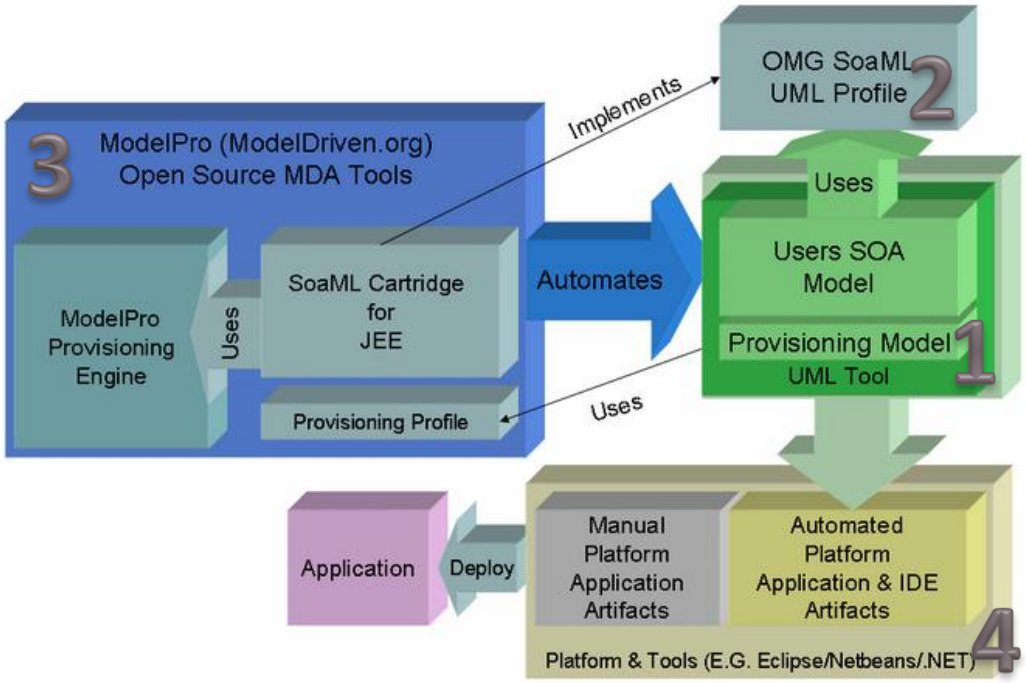


Figura 3.2. Interacción de componentes para la generación en ModelPro [15]

En la Tabla 3.1 se puede ver una lista de las plataformas tecnológicas utilizadas por ModelPro.

Tabla 3.1. Tecnología usada en ModelPro

Web Service Contract	WSDL 1.1
	XML Schema
Java EE	JDK 6.0
	EJB 3.0
	JAX-WS 2.1
Build Environment	Eclipse Web Tools
	Apache Ant

En la Figura 3.3 se muestran los estereotipos claves definidos en el perfil de aprovisionamiento JEE los cuales son explicados.

- **JEE Provisioning**. El motor ModelPro se acopla con el generador de código Java EE cuando se detecta un modelo de provisión específico en un entorno de ejecución del estereotipo Provisioning JEE. En este estereotipo se puede definir la etiqueta *Runtime* que especifica una definición de servidor en el espacio de trabajo de Eclipse.
- **JEE Web Service**. El generador provee un proyecto de Web Service para cada instancia participante con el estereotipo de JEE Web Service. Cada puerto definido por la instancia participante con el estereotipo ServicePoint expondrá un endpoint del Web Service. En el estereotipo de JEE Web Services se define la etiqueta *targetNamespace,* que especifica el espacio de nombres en XML de los artefactos de los Web Services provistos, tales como esquemas XML y definiciones WSDL. Los nombres de los paquetes generados en el código fuente de Java también se derivan del valor de esta etiqueta.
- **Provisioning Service Point.** Este estereotipo permite proporcionar endpoints adicionales al Web Service utilizando etiquetas. Cuando una interfaz de servicio implementa una interfaz UML y utiliza otra, la etiqueta *usedInterfaceAddress* permite al usuario especificar un endpoint estático al Web Service direccionando a la interfaz

utilizada. Esto puede ser sobrescrito en tiempo de ejecución. La etiqueta *usedInterfaceNamespace*, no se utiliza actualmente.

- **Entity Override.** El cartridge dispone de un marcador de posición para que el desarrollador Java pueda implementar lógica adicional para las clases de entidad con este estereotipo.
- **Entity Persistence.** No se utiliza actualmente.

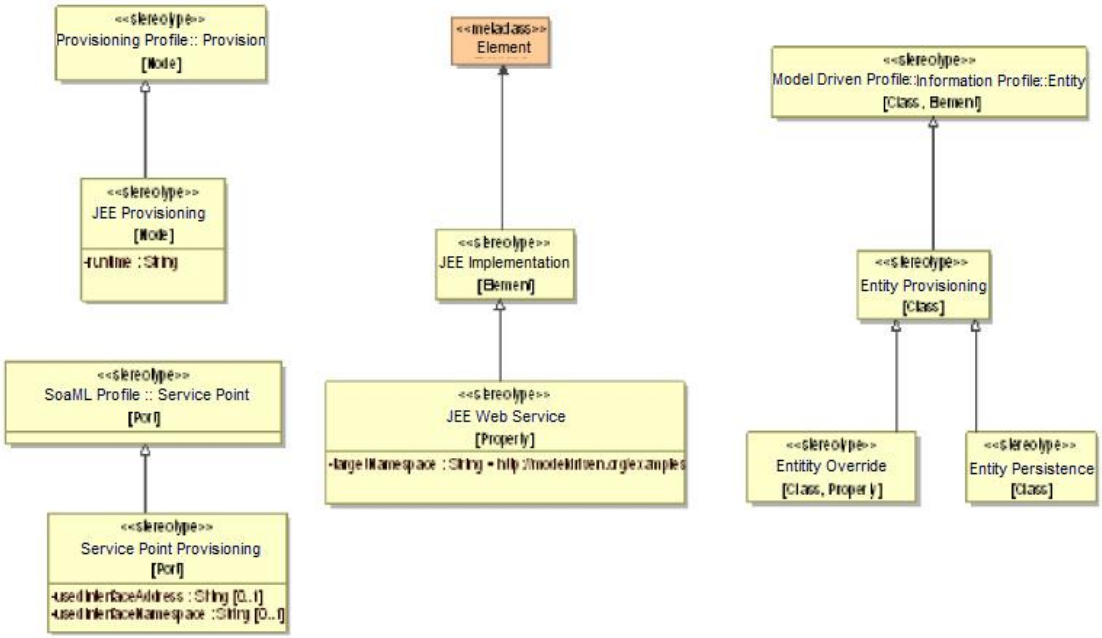


Figura 3.3. Perfil de aprovisionamiento de JEE [15]

En la Figura 3.4 se muestra cómo los diversos elementos que se especifican en el modelo SoaML que forman parte del PIM, son asignados a artefactos de la plataforma específica.

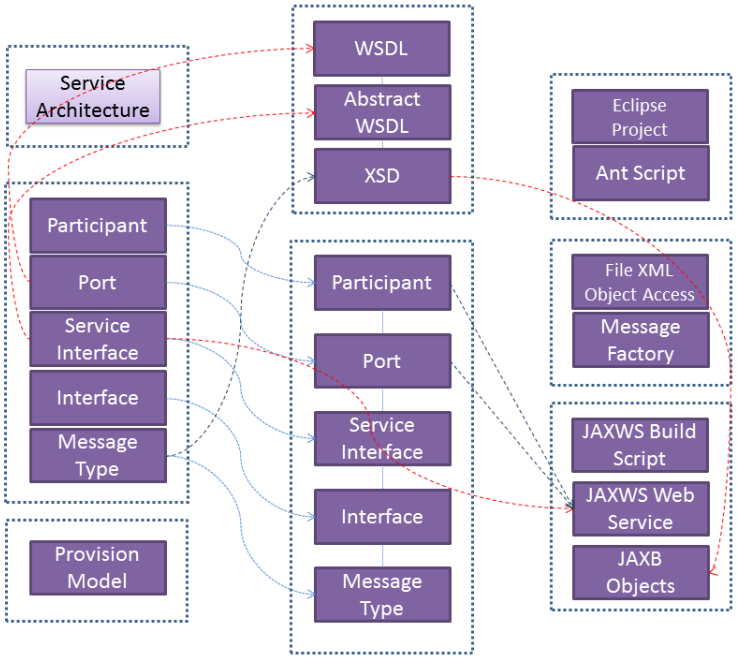


Figura 3.4 - Transformación de elementos SoaML a proyecto Java [15]

Los artefactos Java generados siguen un patrón de diseño que refleja los siguientes principios.

- Los Web Services se implementan utilizando la tecnología JAX-WS, incluyendo JAXB.
- No se imponen limitaciones específicas de tecnología en el modelo. Permitiéndose el uso de clases abstractas en las interfaces de servicio.
- Hay una clara separación de responsabilidades entre el código que se encarga de los contratos de Web Services y el código que implementa la lógica de negocio.
- La estructura de clases en la lógica de negocio (implementado en POJO[9]) refleja la estructura de clase del modelo.

El código Java generado se organiza en tres carpetas.

- **user.src** contiene un paquete donde el desarrollador implementa la lógica de negocio.
- **gen.src** contiene código generado que la lógica de negocio puede utilizar o referirse. Éste puede incluir implementación por defecto de las interfaces de servicio, las interfaces que representan los tipos de datos de referencia directa o indirecta de las interfaces de servicio, así como la aplicación por defecto de estas interfaces de datos.
- **cartridge.src** contiene código fuente utilizado en tiempo de ejecución.

El proyecto también tiene generadas las siguientes carpetas.

- **lib** contiene las librerías que son utilizadas en tiempo de ejecución.
- **wsdl** contiene los WSDLs y los esquemas XML.

IBM® Rational® Application Developer

Es una herramienta comercial que no requiere la instalación de un plug-in extra para la generación de código Java EE y Web Services a partir de modelos SoaML. Para la generación de los modelos se puede utilizar la herramienta IBM RSA (Rational Software Architect, RSA) y para aplicar la transformación a código se requiere el uso de una configuración de transformación, que por lo general consta de tres partes básicas: selección de los elementos de origen de la transformación, selección (o creación y posterior selección) de los elementos de destino y configuración de las propiedades de la transformación [21].

Luego de definida y ejecutada una transformación se colocan los elementos resultantes en una serie de proyectos Eclipse. Estos proyectos pueden ser proyectos de biblioteca o de módulo de WebSphere Integration Developer, como se describe a continuación.

- Los proyectos de biblioteca contienen los objetos de negocios, las interfaces y las exportaciones de módulos que se comparten con otros proyectos.
- Los proyectos de módulo contienen una implementación de módulos para cada participante del modelo de servicios UML.

Conclusiones sobre el relevamiento de herramientas existentes

A partir de la investigación realizada sobre las herramientas existentes para la generación de código ejecutable a partir de modelos SoaML, pudimos comprobar que no hay muchas que cumplan con este requerimiento. De las herramientas investigadas, la que nos brindó mayor cantidad de aportes fue la herramienta ModelPro, la cual nos ayudó a diseñar la transformación de los distintos elementos que forman parte del modelo SoaML hacia los elementos Java que representan dicho modelo. Otra de las características de la herramienta ModelPro que nos sirvió de base para el desarrollo de nuestro producto, fue el observar y analizar la arquitectura de los Proyectos Java generados, así como las implementaciones soportadas (EJB 3.0 y JAX-WS 2.1). Estas implementaciones fueron consideradas como requerimientos básicos para la creación de nuestro producto, aunque nos pareció que un aspecto a mejorar en nuestro producto en comparativa con la herramienta ModelPro, era brindarle al usuario una mayor cantidad de implementaciones de los Web Services.

[9] POJO, Plain Old Java Object. Término utilizado para expresar la utilización de clases Java simples.

4. Alternativas de Implementación

Este capítulo consta de cuatro secciones, en la sección 4.1 se realiza una breve introducción de los problemas que eran necesarios resolver para cumplir con los objetivos. La sección 4.2 explica la metodología utilizada para realizar la evaluación de las tecnologías que permita escoger las más convenientes. En la sección 4.3 se detallan las diferentes alternativas tecnológicas que se investigaron y por último en la sección 4.4 se presenta un resumen general del capítulo.

4.1. Introducción

Aquí haremos hincapié en la investigación realizada y forma de evaluación escogida para seleccionar las herramientas que a nuestro entender y de acuerdo a lo investigado serían las más adecuadas para el desarrollo de este plug-in Eclipse. Para esto es necesario describir brevemente las problemáticas principales a las cuales nos enfrentamos. En primera instancia se debió definir un mapeo entre los componentes SoaML y componentes Java, para esto debimos estudiar cómo interpretar documentos XMI. Una vez interpretados estos archivos que en definitiva son los modelos SoaML que servirán como insumo del plug-in a desarrollar, se necesita saber cómo trabajar con estos elementos interpretados. Luego de esta interpretación debimos investigar herramientas para generar el código representativo de la realidad expuesta en estos modelos. Por último, era necesario que el código sea ejecutable sin mayores inconvenientes por lo que se realizó la investigación necesaria para que el código una vez generado conste de todo lo necesario para ser ejecutado en un servidor web o de aplicaciones.

4.2. Criterios de evaluación

Para la elección de las diferentes tecnologías que a nuestro criterio serían mejores para la herramienta a desarrollar, se presentan una serie de puntos que permiten evaluar cada una de éstas. Dado que para cada una de las problemáticas contábamos con variedad de opciones para su solución, escogimos un método de selección que permita obtener un producto de calidad, respetando algunas reglas básicas en el desarrollo de software.

Los criterios utilizados para la elección de las distintas alternativas de implementación fueron los siguientes.

- Factibilidad para cumplir con los requerimientos exigidos.
- Costo de aprendizaje
- Tiempo de desarrollo
- Documentación disponible (en términos cualitativos y cuantitativos)

4.3. Tecnologías y frameworks involucrados

Una vez descriptos los criterios de evaluación, pasamos a describir las diferentes tecnologías investigadas relacionadas a las problemáticas que se tuvieron que resolver para cumplir con los requerimientos planteados. Estas tecnologías sirvieron de base para el desarrollo de los distintos componentes que forman la solución.

Componente Parser

La manipulación de documentos XMI fue una de las principales problemáticas que hubo que atacar a lo largo del desarrollo del proyecto, dado que los modelos SoaML que son el insumo necesario del plug-in a desarrollar, están representados mediante documentos XMI. Por consiguiente, se realizó una investigación sobre algunas de las herramientas existentes para la interpretación de este tipo de documentos. A continuación presentamos las tecnologías investigadas para realizar el parser necesario.

JDOM

Para profundizar más sobre los conceptos de JDOM manejados en esta sección se puede ver [17]. JDOM es una API Java "open-source" para el procesamiento de documentos XML. Esta herramienta provee una solución para la manipulación de XML y obtención de objetos Java a partir de éste.

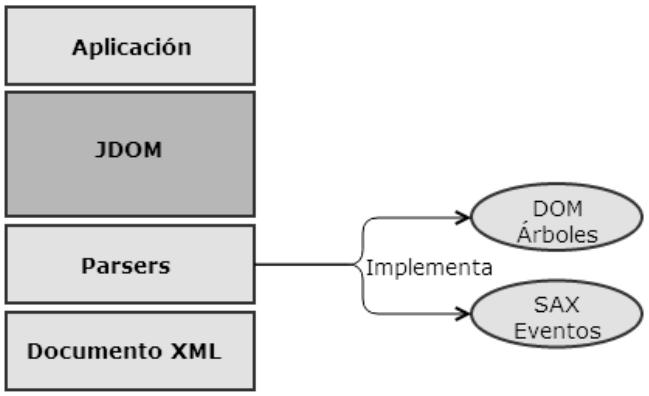


Figura 4.2. Composición de JDOM [17]

JDOM permite el modelado en Java de los objetos representados en el documento XML. JDOM inter-opera con facilidad tanto con SAX API[10] como con DOM[11] dado que utiliza JAXP[12] como analizador por defecto pero éste puede modificarse.

Dentro de las ventajas que presenta JDOM, se destaca que es una herramienta "open-source" y que al estar desarrollado en Java puede considerarse una extensión del lenguaje. Posee además gran simplicidad para su uso, ya que realiza un manejo de los elementos Java de forma muy natural. Uno de los puntos más importantes es la interoperabilidad que tiene con las normas existentes como SAX y DOM, lo cual lo convierte en una herramienta muy completa ya que permite manejar documentos XML de gran porte que en ciertas herramientas es un inconveniente. Además si estos documentos fuesen de pequeño o mediano porte también brinda la posibilidad de editarlos.

Una desventaja que presenta JDOM es que para utilizarlo es necesario agregar el componente al proyecto, a diferencia de otras APIs para parser, donde por ejemplo se encuentran en la JDK[13].

Xerces

Otra de las herramientas investigadas para resolver la problemática del parser, es Xerces. Ésta fue creada por Apache Software Foundation y es una colección de bibliotecas de parser que permite validar, serializar y manipular documentos XML. Xerces introduce XNI[14] (Xerces Native Interface, XNI) que ofrece una manera más sencilla de manipular y configurar analizadores XML. Está implementado en Java sobre la especificación de DOM, definida por World Wide Web Consortium. DOM genera un árbol jerárquico en memoria del documento XML, lo cual

[10] Simple API for XML es el estándar de facto para la utilización de XML en Java. Funciona mediante eventos, lo cual lo hace ideal para documentos XML de gran tamaño dado que no maneja un árbol lo que dificultaría su manejo. http://www.saxproject.org/

[11] Document Object Model realiza el procesamiento del documento XML mediante un árbol que puede ser modificado, presenta dificultad para trabajar con documentos grandes http://www.w3.org/DOM/

[12] Java API for XML Processing realiza el procesamiento de documentos XML utilizando analizadores de los mismos tanto SAX como DOM. http://docs.oracle.com/javase/tutorial/jaxp/intro/index.html

[13] Java Development Kit es un software que provee herramientas para el desarrollo de aplicaciones en Java http://www.oracle.com/technetwork/java/javase/

[14] Xerces NativeInterface-http://xerces.apache.org/xerces2-j/xni.html

permite que a través del parser (Xerces) pueda ser manipulada la información. Xerces ofrece también soporte para otros estándares como SAX y JAXP. Por otra parte, esta herramienta en su última versión tiene soporte para versiones 1.0 y 1.1 de XML.

Una posible desventaja con respecto a JDOM es el nivel de abstracción que posee, lo que no haría tan sencillo su manejo.

XML4J

Otra herramienta investigada, no con tal profundidad como las anteriores, fue XML4J. Ésta al igual que las ya descriptas permite la manipulación de documentos XML. XML4J desarrollado por IBM es como su nombre dice una herramienta especialmente realizada para el lenguaje Java. Tiene como base de su desarrollo a Apache Xerces. Al igual que Xerces maneja principalmente DOM, por lo que también maneja el documento XML como un árbol, en consecuencia puede ser modificado pero tiene dificultad para procesar documentos de gran porte. La última versión de XML4J tiene soporte para gran cantidad de estándares entre los más conocidos SAX, DOM, JAXP y XML en todas sus versiones.

Evaluación y alternativa escogida

Si bien los criterios descriptos en 4.2 fueron importantes al momento de decidir sobre cuál tecnología era la más adecuada para cumplir con los requerimientos, un factor importante que influyó en la elección de la tecnología para realizar el parser, fue la obtención del plug-in is4BPe (Insert Services for Business Process Execution, is4BPe) brindado por el tutor al comienzo del proyecto. Este plug-in es parte de otro proyecto en el cual, tomando un archivo XMI que contiene un modelo SoaML y un modelo de proceso de negocio con las actividades de servicio marcadas, inserta en el modelo de proceso las llamadas a las operaciones de los servicios definidos en el modelo SoaML, para su invocación en ejecución. Éste realiza un "parseo" del archivo XMI generado por el plug-in SoaML. El mismo no contempla completamente la realidad a la que hay que enfrentarse pero funcionó como punto de partida para afrontar el problema. Éste está implementado sobre JDOM, lo que ayudó en la elección de esta tecnología para la implementación.

Para comprobar la factibilidad y el costo de aprendizaje se realizó un prototipo para evaluar JDOM y de esta forma comprobar si los resultados eran los esperados para continuar con el desarrollo sobre el plug-in brindado por el tutor. Lo que se hizo fue extender y modificar el plug-in is4BPe, con el objetivo de abarcar todos los casos necesarios. Así comprobamos que JDOM es una herramienta de uso muy simple y que además contempla todas nuestras necesidades, con lo que se concluyó que no era necesario realizar un cambio de tecnología a la que ya traía el parser del cual partimos.

A continuación se muestra la Tabla 4.1 que resume la comparativa de las tecnologías analizadas de acuerdo a lo mencionado anteriormente.

Tabla 4.1. Comparativa de las tecnologías analizadas

	XML4J	Xerces	JDOM
Factibilidad	Si (según documentación, no se experimentó con prototipo)	Si (dado que JDOM y XML4j trabajan con esta herramienta)	Si (Comprobado empíricamente)
Costo de aprendizaje	Bajo	Bajo	Bajo
Tiempo de desarrollo	No se tiene el dato	--	Nivel de abstracción mayor por lo tanto más simple que trabajar directamente con Xerces
Documentación	--	--	--

Componente de generación de código

Dada la problemática de generar el código representativo de un modelo SoaML, es que se realizó el estudio que detallaremos a continuación. En principio se escogió el enfoque a seguir para la implementación de un Web Service. Dadas las alternativas planteadas en la Sección 2.5, se escogió el enfoque Top-Down, es decir, generar un WSDL y luego a partir de éste el código del Web Service. Se optó por este enfoque ya que al no contar con un contrato de servicio WSDL ni con las clases Java que implementan un Web Service, nos pareció más práctico y sencillo generar un documento WSDL a partir de los elementos interpretados del modelo SoaML. Para evaluar la dificultad se realizaron pruebas para generar clases Java y documentos WSDL. Por otro lado, analizando posibles ventajas y desventajas de seguir alguno de los enfoques en nuestro proyecto, se llegó a la conclusión de que ninguno de éstos tenía limitantes para cumplir con la generación de un Web Service, ni tampoco ninguno presenta una cualidad sobre el otro que impacte positivamente.

Posteriormente se realizó un relevamiento sobre las herramientas existentes que permiten generar código Java para la implementación de un Web Service. Entre los frameworks encontrados que ayudan a generar código Java, se destacan JBossWS, Apache CXF y Apache AXIS2. Estos frameworks brindan herramientas para la generación de código Java a través de las cuales se logra exponer y consumir servicios, soportando distintas implementaciones.

JBossWS (JBoss)

JBossWS es un framework de Web Services (WS) desarrollado como parte de JBoss Application Server. Este framework proporciona una integración entre el servidor de aplicaciones y las tecnologías WS relacionadas que se necesitan para lograr el cumplimiento de la Plataforma Java, Enterprise Edition 6 (Java EE 6).

Entre las herramientas que proporciona JBossWS, encontramos JAX-WS-Tools que puede ser utilizada de distintas maneras para obtener el desarrollo del lado del servidor, así como también del lado del cliente. Cuando desarrollamos un endpoint del WS (del lado del servidor), se tiene la opción de comenzar el desarrollo desde Java (bottom-up development), o a partir de un contrato WSDL que define un servicio (top-down development). Si se trata de un nuevo servicio (sin contrato WSDL), el enfoque botton-up es el camino más rápido, sólo debemos implementar las clases java y añadir a las mismas las anotaciones correspondientes para lograr exponer el servicio correctamente. Sin embargo, si se desea desarrollar un servicio con un contrato (WSDL) ya definido, será mucho más eficiente utilizar el enfoque top-down, ya que la herramienta va a generar el código para lograr exponer los servicios que se encuentren en el contrato especificado.

En la tabla [1] del Apéndice se puede ver una lista de herramientas de línea de comandos JAX-WS que se incluyen en JBossWS [18].

Para este proyecto se realizaron pruebas con el comando wsconsume, dado que el enfoque de desarrollo utilizado es el top-down. Este comando tiene una serie de parámetros que dan varias posibilidades sobre lo que genera la ejecución del mismo. El listado de parámetros se puede ver en la tabla [2] del Apéndice.

Apache CXF

"Apache CXF es un framework para servicios de código abierto. CXF ayuda a construir y desarrollar servicios usando APIs como JAX-WS y JAX-RS. Este servicio soporta gran cantidad

protocolos como SOAP, XML/HTTP, ResTful HTTP, CORBA y como transporte maneja HTTP y JMS[15]*"* [27].

CXF incluye un amplio conjunto de características, pero se centra principalmente en las siguientes áreas.

- Soporte a múltiples estándares de Web services.
- CXF tiene soporte para una variedad de lenguajes de programación en los clientes.
- Flexibilidad para el "deploy" (Tomcat, JBoss, Jetty , IBM WebSphere, etc)
- Soporta gran variedad de protocolos de transporte.

Entre las funcionalidades que presenta se encuentran herramientas para la generación de código, generación de WSDL, agregar endpoints, validar archivos, etc. A continuación mostramos la lista de herramientas completa que provee Apache CXF.

- Generación de codigo: WSDL to Java, WSDL to JavaScript, Java to JavaScript
- Generación de WSDL: Java to WSDL, XSD to WSDL, IDL to WSDL, WSDL to XML
- Agregar Endpoints: WSDL to SOAP, WSDL to CORBA, WSDL to service
- Generación de archivos de apoyo: WSDL to IDL
- Validación de archivos: WSDL Validation

Dado que el enfoque de desarrollo utilizado para este proyecto es top-down, la herramienta que es de nuestro interés es la de generación de código a partir de un contrato de servicio (WSDL), más precisamente la herramienta "WSDL to Java". Esta herramienta es brindada a partir del comando wsdl2Java. Este comando genera a partir del WSDL tanto la implementación del servicio (server) como también el/los cliente/s que consumen el servicio descripto por el WSDL.

El comando wsdl2Java al igual que wsconsume tiene una serie de parámetros que permiten configurar las variantes con las que se desea generar el código Java, entre ellas permite configurar si se desea generar un cliente, servidor o ambas. Algunos de los parámetros utilizados comúnmente se pueden ver en la tabla [3] del Apéndice.

Apache Axis2

El proyecto Apache Axis2 es una aplicación basada en Java, para el desarrollo de Web Services tanto del lado cliente como del lado servidor. Fue diseñado aprovechando las lecciones aprendidas de Apache Axis 1.0. Apache Axis2 proporciona un modelo de objetos completo y una arquitectura modular que hace que sea fácil agregar funcionalidad y soporte para las nuevas especificaciones y recomendaciones para los servicios relacionados con Internet.

Axis2 permite realizar fácilmente las siguientes tareas.

- Enviar mensajes SOAP
- Recibir y procesar mensajes SOAP
- Crear un servicio Web desde una clase Java simple
- Crear clases de implementación para el servidor y el cliente utilizando WSDL
- Recuperar fácilmente el WSDL para un servicio
- Enviar y recibir mensajes SOAP con archivos adjuntos
- Crear o utilizar un servicio web basado en REST
- Crear o utilizar los servicios que utilizan WS-Security, WS-ReliableMessaging, WS-Addressing, WS-Coordination.[16]

[15] Java Message Services es un estándar de mensajería que permite a las aplicaciones basadas en la plataforma Java, crear, recibir, enviar y leer mensajes. También hace posible la comunicación confiable de manera síncrona y asíncrona. http://docs.oracle.com/javaee/1.3/jms/tutorial/

[16]Para saber más acerca de WS-Security, WS-ReliableMessaging, WS-Addressing, WS-Coordination puede ir http://www.fing.edu.uy/inco/grupos/lins/documentos/soa/ws/LINS_Introduccion_a_WS.pdf

- Utilizar la estructura modular de Axis2 para agregar fácilmente soporte para nuevas recomendaciones que van surgiendo

Dado que el enfoque de desarrollo utilizado para este proyecto es top-down, se utilizará la herramienta de JAX-WS, wsimport, para procesar un archivo WSDL y generar artefactos portables de Java que se utilizan para crear un Web Service. Estos artefactos protables creados con la función wsimport son:

- Interfaz de Endpoint de servicio (SEI)
- Clase de servicio
- Clase de Exception que se asigna desde el wsdl: Clase de error (de haberlo)
- Arquitectura Java para vinculación XML (JAXB) genera valores de tipo que son clases Java mapeados de tipos de esquema XML

El comando wsimport utiliza una serie de parámetros, los cuales se pueden ver en el Apéndice en la tabla [4].

Evaluación y alternativa escogida

Siguiendo los principios descriptos en la Sección 4.2 para la evaluación de las tecnologías es que se realizaron tres prototipos, uno sobre la tecnológica brindada por JBossWS, específicamente mediante el comando WSConsume. Un segundo prototipo para evaluar la tecnología brindada por Apache CXF, más precisamente el comando WSDL2Java. Por último, se realizó un tercer prototipo para evaluar la tecnología brindada por Apache Axis2, más precisamente a partir del comando wsimport. En la tabla [6] del Apéndice, se puede ver unas tablas comparativas donde se observan algunas de las tecnologías y las características que soportan cada framework y la tabla [7] corresponde a los servicios web relacionados.

Considerando una posible extensión del plug-in SoaML2Code, para agregar por ejemplo una implementación de Web Services RESTful, vemos que el framework JBossWS es menos potente que los otros dos, dado que provee características más acotadas. En la Tabla 4.2 se muestran las características que fueron importantes para la elección del framework.

Tabla 4.2 – Comparativa de las tecnologías de generación de código

	Apache CXF - Wsdl2Java	**JBossWS – WSConsume**	**Apache Axis2 - WSimport**
Factibilidad	Si	Si	Si
Costo de aprendizaje	Bajo	Bajo	Medio
Tiempo de desarrollo	Bajo	Bajo	Bajo
Documentación	Muy Buena	Buena	Buena

El uso exitoso de estos frameworks, los cuales cumplen con gran parte de los requerimientos necesarios para la realización de este proyecto, junto con el simple uso de los mismos y la buena cantidad y calidad de documentación, nos llevó a inclinarnos por la opción de utilizar uno de ellos sin ni siquiera tomar en cuenta la idea de la implementación propia del generador de código dado los costos de tiempo y desarrollo que esto implicaba.

Luego de evaluar las alternativas planteadas, tanto el framework Apache CXF como el framework Apache Axis2 se muestran superiores al framework JBossWS. Estos dos proyectos proporcionan estándares para Web Services y presentan características similares. Las principales ventajas que presenta Apache CXF por las cuales elegimos este framework son su facilidad de uso, lo que comprobamos cuando realizamos pruebas en ambos frameworks, además encontramos un buen nivel de documentación y soporte mediante la gran comunidad que posee Apache CXF. Otra ventaja que presenta es que posee una mejor integración con

Spring, además de que el código generado es más entendible y por tal su mantenimiento es más sencillo.

Servidor de aplicaciones

El proyecto no presentaba ningún requerimiento en cuanto al servidor de aplicaciones en el que se debía "deployar" el código generado por el plug-in a desarrollar. Por esta razón se tomó la decisión de utilizar un servidor de aplicaciones y un servidor web que a nuestro entender son de uso muy común y masivo en el ambiente. Éstos fueron JBoss como servidor de aplicaciones y Apache Tomcat como servidor Web.

JBoss

Como se mencionó anteriormente es un servidor de aplicaciones que presenta como una de sus principales características que es de código abierto. Es una plataforma popular dentro de los desarrolladores de software, por lo que existe gran cantidad de información brindada en los distintos foros referentes a ella. Esto permite enfrentar con mayor facilidad algunas dificultades que pueden surgir a partir de su utilización. Por otro lado JBoss puede ser utilizado y distribuido sin restricciones de licencia [32].

Otras características que JBoss ofrece son, por ejemplo el soporte de Java EE y soporte de Cluster. En cuanto a usuarios, permite la creación y validación de éstos, permitiendo además la asignación de roles. Cabe destacar que desde el IDE Eclipse se pueden manejar las distintas versiones de JBoss permitiendo depurar y realizar otras tareas relacionadas con el desarrollo de aplicaciones.

La versión de JBoss utilizada para la publicación de los Web Services generados por el plug-in desarrollado es jboss-5.0.1.GA.

Apache Tomcat

Apache Tomcat es un servidor web multiplataforma que funciona como contenedor de servlets[17] bajo licencia de software libre.

La versión de Apache Tomcat utilizada para publicar los Web Services generados por el plug-in desarrollado es Apache Tomcat 7.0. Una de las características que presenta este servidor es que soporta la especificación Servlet 3.0, la cual permite configuración dinámica, soporte de anotaciones y soporte para la inclusión de contenidos externos directamente en una aplicación Web.

Entorno de desarrollo

Otra de las decisiones que debimos tomar fue la de qué versión de Eclipse íbamos a utilizar. Si bien en un comienzo se estaba restringido a la versión de Eclipse utilizada por el plug-in SoaML del cual partimos, decidimos que para la utilización del plug-in SoaML2Code sea posible utilizar cualquier versión de Eclipse, dado que para el plug-in desarrollado en este proyecto no se trabaja con ningún componente de Eclipse directamente, por lo tanto no quedó ligado a ninguna versión logrando una gran independencia en este aspecto. De todas maneras, el usuario que use ambos plug-ins deberá usarlos en Eclipse Helios Modeling SR2. En un comienzo se trabajó con Eclipse Helios Modeling SR2 pero luego se realizaron pruebas en otras versiones de Eclipse, como por ejemplo en Eclipse Juno.

4.4. Resumen

En esta sección del apartado realizamos un resumen general del Capítulo centrándonos en las tecnologías escogidas. De acuerdo a la investigación realizada, se lograron encontrar varias

[17] Servlets, son módulos que extienden los servidores Web, actúan como capa intermedia entre las aplicaciones del servidor y la petición del cliente.

herramientas/frameworks que nos simplificaron la resolución de las problemáticas planteadas para nuestro proyecto. A partir de la construcción de los prototipos y contando con alguna herramienta (parser brindado por el tutor) como para tener un punto de partida se escogieron las tecnologías más adecuadas para el desarrollo del plug-in.

A la hora de realizar el componente parser, las herramientas tienden mucho a parecerse y brindar las mismas funcionalidades para la manipulación de este tipo de documentos. Se escogió JDOM dado que era la herramienta que brindaba mayor facilidad a la hora del manejo de documentos XML. El otro motivo no menos importante fue tener un plug-in (is4BPe) que utiliza JDOM, el cual sirvió como base a la hora de procesar los modelos SoaML.

Por otro lado, para afrontar la problemática asociada a la generación de código Java se optó por el framework Apache CXF, dado que según la evaluación que se realizó, éste nos brindaba mejores alternativas asociadas a nuestro proyecto.

En cuanto a los servidores de aplicación, no se realizó una investigación exhaustiva. Lo que se hizo fue contemplar tanto JBoss como Apache Tomcat como servidores para ejecutar los Web Services generados por el plug-in desarrollado, dado que a nuestro entender ambos son servidores ampliamente usados. No se probó con ningún otro servidor de aplicaciones o web, pero seguramente no se encuentren inconvenientes al utilizar por ejemplo el servidor GlassFish.

5. Solución Propuesta (Plug-in SoaML2Code)

En este capítulo se presenta la forma en la que se llevó a cabo el desarrollo del generador de código Web Services y Java EE partiendo de modelos SoaML. En la sección 5.1 se describe la solución propuesta, en la sección 5.2 se explica la arquitectura propuesta y sus características. En la sección 5.3 se incluye una descripción breve de la implementación y por último en la sección 5.4 se mencionan las dificultades encontradas a lo largo del desarrollo de la solución.

5.1. Descripción de la Solución

La solución propuesta se basa en la creación del plug-in SoaML2Code para el ambiente de desarrollo de Eclipse, el cual sigue la visión Model Driven Architecture para proveer soluciones de software ejecutables sobre arquitecturas de negocio. Partiendo de modelos SoaML (basado en el estándar OMG) tanto en formato UML como en formato XMI (versión 2.0 y 2.1), nos permite exponer los servicios que componen el modelo antes dicho de forma rápida y eficiente en función de las necesidades empresariales y procesos sobre la plataforma Java. El plug-in SoaML2Code es una extensión del Plug-in SoaML el cual permite generar modelos SoaML para el ambiente de desarrollo de Eclipse, siendo estos modelos los que a posteriori serán utilizados como insumos por el plug-in SoaML2Code. Por más que el plug-in SoaML2Code sea una extensión, es totalmente independiente y no es necesario tener instalado el plug-in SoaML en el ambiente de desarrollo de Eclipse para poder utilizarlo correctamente.

En la Figura 5.1 se muestra un diagrama de los componentes que interactúan en el proceso de generación de código del plug-in SoaML2Code. En (1) se observa el plug-in SoaML que implementa el estándar SoaML de la OMG y provee los modelos que son utilizados por el componente (2) para la generación de código. En (2) se observa el plug-in SoaML2Code que utiliza como base de su desarrollo el plug-in is4BPe para el componente encargado de realizar el "parseo" de los modelos y el framework Apache CXF para el componente encargado de realizar la generación de código. El código generado por el componente (2) es ejecutado en los servidores de aplicación del componente (3).

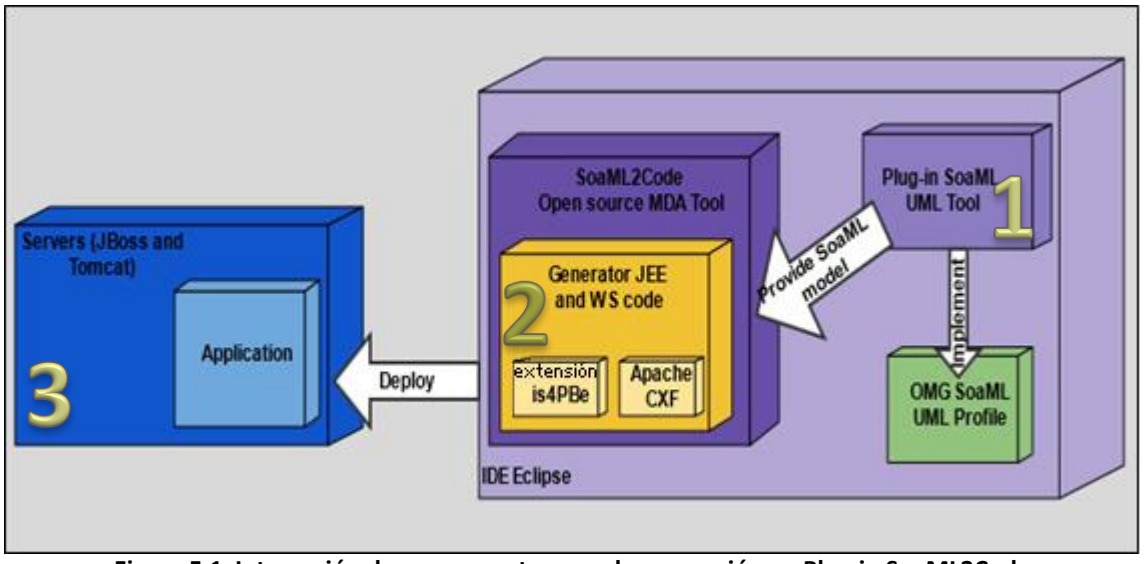


Figura 5.1. Interacción de componentes para la generación en Plug-in SoaML2Code

La siguiente es una lista de las plataformas tecnológicas de destino que el plug-in SoaML2Code provee.

Tabla 5.1. Tecnologías destino del plug-in SoaML2Code

Web Service Contract	WSDL 1.1 (SOAP 1.1)
	XML Schema 1.0
Java EE	JDK 6.0 o superior
	EJB 3.0
	JAX-WS ri 2.0
	JAX-WS + Spring 2.0
Build Environment	Eclipse Web Tools

5.2. Arquitectura de la solución

La arquitectura fundamental de Eclipse está basada en plug-ins, esto permite integrar elementos (nuevas funcionalidades) al sistema enriqueciéndolo con nuevas características. Cada plug-in debe extender ciertos puntos de extensión básicos del core Eclipse (relacionados a las vistas, perspectivas, etc.) y a su vez puede ofrecer nuevos puntos de extensión para que otros plug-ins puedan reutilizar o extender ciertas funcionalidades. El nuevo plug-in estará implementado en un módulo (jar), el cual se debe agregar en la carpeta plug-ins de Eclipse (carpeta en el directorio raíz de Eclipse que contiene los plug-ins que brindan distintas funcionalidades).

Durante el desarrollo de la solución, se identificaron tres componentes bien marcados los cuales se muestran a continuación mediante el diagrama de la Figura 5.2 que presenta la interacción entre éstos en la solución propuesta para el desarrollo del plug-in SoaML2Code.

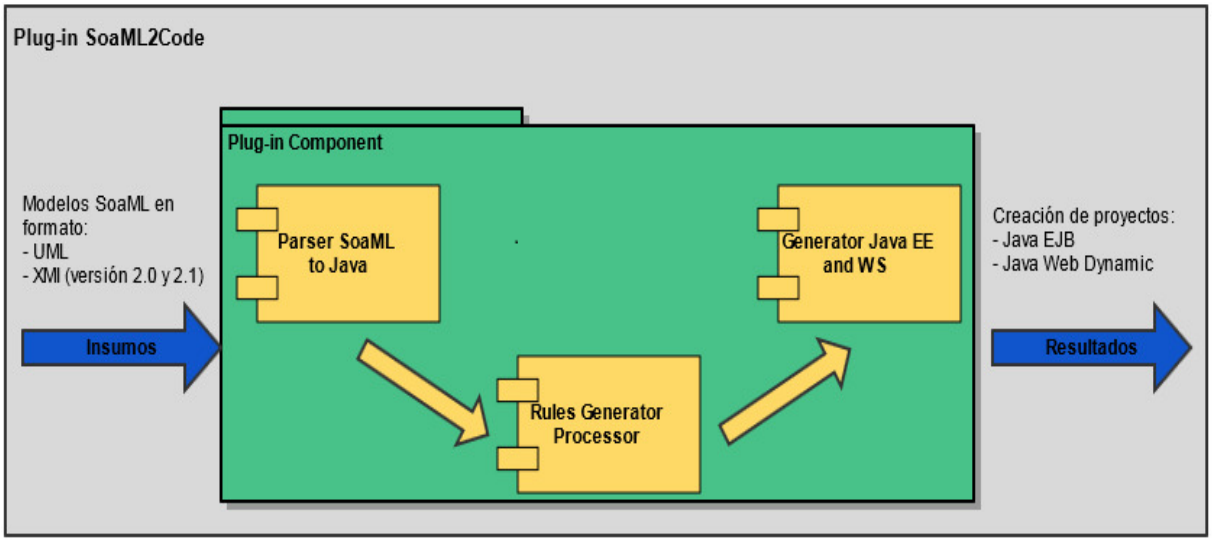


Figura 5.2. Diagrama de componentes del plug-in SoaML2Code

A continuación se describe cada componente que forma parte de la solución.

Parser SoaML to Java

Este componente de la solución es el encargado de realizar el "parseo" sobre un modelo SoaML (archivo UML o XMI) para la posterior creación de objetos Java que lo representan.

Rules Generator Processor

Este componente es el encargado de procesar los objetos Java generados a partir del modelo SoaML y aplicar un conjunto de reglas o condiciones para invocar a los generadores de código de Web Service y Java EE.

Generator Java EE and WS

Este componente es el encargado de generar el código de Web Service y Java EE que representa al modelo SoaML utilizado tomando en cuenta las reglas aplicadas para la generación.

5.3. Implementación

En esta sección se describe a grandes rasgos la forma en que se implementó cada componente del plug-in presentado en la sección anterior.

Implementación de Parser SoaML to Java

Para comenzar, se analizaron los elementos del dominio que componen el modelo SoaML y se estudió el formato del archivo de intercambio de modelos basado en el estándar XMI. El modelo de dominio SoaML del que partimos se muestra en la Figura 5.3.

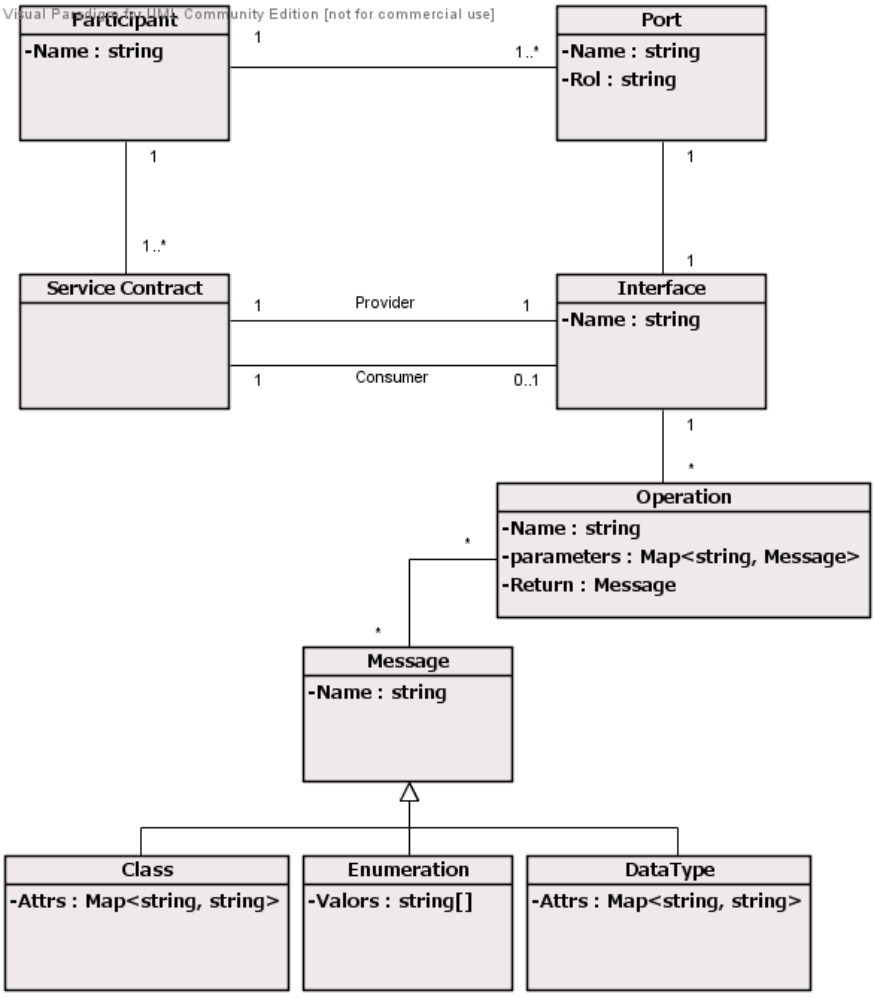


Figura 5.3. Modelo de dominio SoaML

Para implementar la solución de este componente se reutilizó la funcionalidad de exportar un modelo SoaML a un archivo XMI del plug-in SoaML. Para el manejo de archivos se utilizó JDOM, quien provee una completa solución basada en Java para acceder y manipular datos XML desde el código Java. A partir del mismo se realizó todo el "parseo" de los archivos XMI que correspondían a los modelos SoaML a transformar. Cabe destacar que el formato del archivo XMI a procesar debe coincidir con el formato de los archivos XMI exportados por el plug-in SoaML que se quiere extender. Este archivo debe cumplir con la estructura general presentada en la Figura 5.4.

```
<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="Opertation1">
            <ownedParameter name="parameter" />
          </ownedOperation>
        </packagedElement>
        ..
        <packagedElement>
          <ownedOperation name="OpertationN">
            <ownedParameter name="parameter" />
          </ownedOperation>
        </packagedElement>
      </packagedElement>
    </packagedElement>
    ..
</xmi:XMI>
```

Figura 5.4. Estructura general archivo XMI entrada del parser.

Para el desarrollo de la transformación nos basamos en el plug-in is4BPe el cual realiza un "parseo" de un modelo SoaML desde un archivo XMI y lo transforma en objetos Java que representan dicho modelo. Este plug-in debió ser modificado y extendido para considerar el modelo de dominio que utilizamos en este proyecto como se muestra en la Figura 5.3. Además, se debió agregar una capa para soportar distintas versiones de los archivos XMI que representan los modelos SoaML. Por el momento se realizaron dos implementaciones, es decir, se soportan las versiones 2.0 y 2.1 de archivos XMI donde para cada versión se realizó su implementación particular tomando en cuenta las variaciones de cada versión. Para facilitar la posibilidad de agregar otras implementaciones para próximas versiones de archivos XMI, se utilizó el patrón Strategy mostrado en la Figura 5.5.

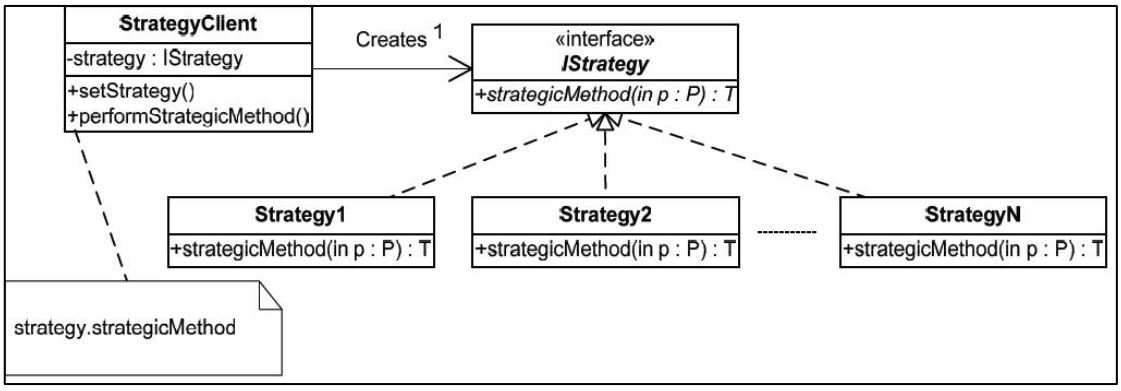


Figura 5.5. Implementación del patrón Strategy

La Interfaz es implementada según la versión que tenga el archivo XMI siendo el Cliente el responsable de crear y mantener una referencia a una estrategia concreta.

Luego de tener el archivo “parseado”, resta especificar el modelo Java de correspondencia con el modelo SoaML que fue definido, es decir la transformación de los objetos Java del modelo SoaML a sus correspondientes objetos Java que generan la implementación del modelo. Dicha transformación se muestra en la Figura 5.6.

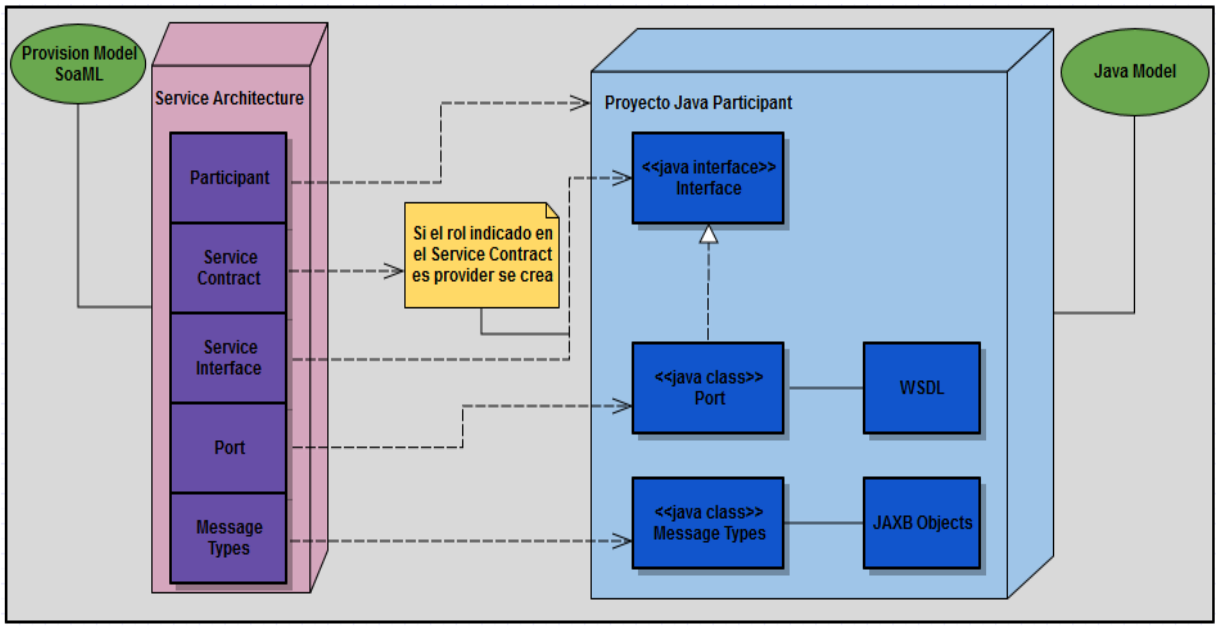


Figura 5.6. Implementación Java de modelo SoaML

Por cada participante (Participant), se decidió generar un proyecto Java dado un tema de facilidad de interpretación y desacoplamiento de los elementos del modelo. Cada participante tiene asociado un conjunto de puertos (Ports) los cuales tienen asociada una interfaz (Interface). La transformación de dichos elementos será representada por una interfaz (Java) por cada interfaz (Interface) del modelo SoaML, las cuales serán implementadas por una clase que lleva el nombre del puerto asociado a la interfaz. Dependiendo el rol de la interfaz (Interface provider o consumer), en caso de que el mismo sea provider, se genera un WSDL asociado al puerto para exponer un servicio. El objeto Operation del modelo SoaML se corresponde con la operación definida en la interfaz y los Message son clases Java o DataType.

De esta manera ya tenemos los objetos Java que representan el modelo SoaML que queremos implementar y ya definimos el modelo en el que lo queremos transformar. Sólo resta definir las reglas que deseamos aplicar para la generación y generar la implementación de dicho modelo.

Implementación del Rules Generator Procesor

Este componente es el encargado de procesar los parámetros de generación ingresados por el usuario desde la interfaz de Eclipse. Esta interfaz fue diseñada en base al diagrama de despliegue que provee Papyrus a partir de la versión 8.0., el cual no pudo ser utilizado dado que el plug-in SoaML del que partimos, funciona con una versión de Papyrus anterior a la mencionada, la cual no soporta diagramas de despliegue.

Luego de realizar el análisis de los elementos que se definen en el diagrama de despliegue, se decidió utilizar un conjunto de parámetros para la generación de código, los cuales indican el tipo de proyecto que se desea generar, entre las opciones que se despliegan se encuentran los proyectos Java EJB (jar) o los proyectos Java Dynamic Web (war), la implementación de los Web Services a publicar, la cual puede ser mediante el estándar JAX-WS RI o JAX-WS + Spring.

También se pueden seleccionar los participantes del modelo SoaML que deseamos incluir en la generación, pudiendo dejar parte del modelo sin transformar a código Java. Otra opción que es desplegada en la interfaz, es la de incluir la generación de un cliente Web Service para consumir los servicios que serán expuestos. Por último se puede editar la IP y puerto donde serán expuestos los servicios. En la Figura 5.7 se muestra la interfaz que es desplegada por Eclipse para la configuración de todos los parámetros de generación señalados anteriormente.

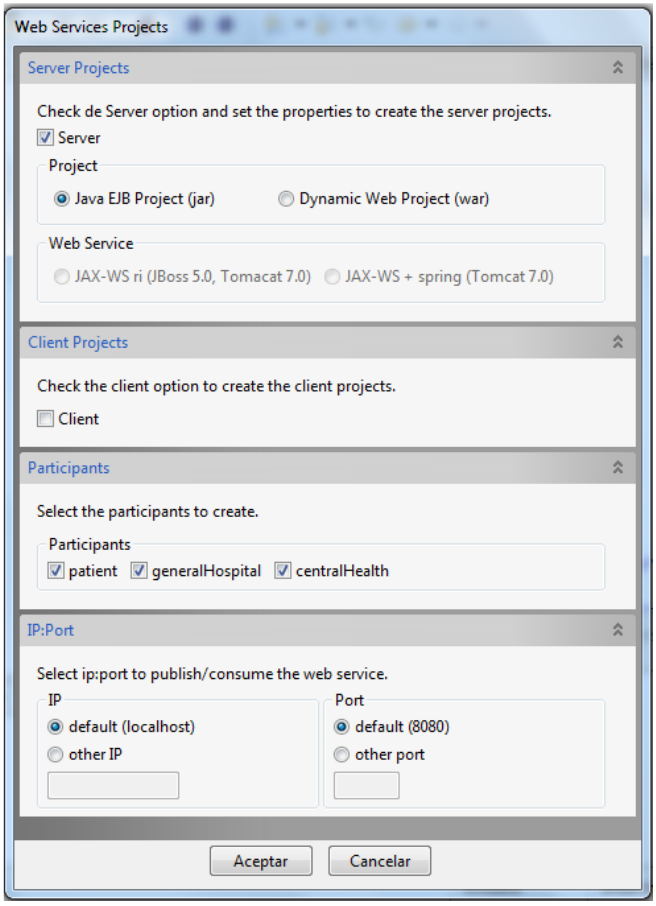


Figura 5.7. Interfaz de Eclipse

A partir de los parámetros seleccionados se procesan las reglas que deciden cómo será la generación de el/los proyecto/s que representan al modelo SoaML.

Implementación del Generator Java EE and WS

Para implementar el desarrollo de este componente, primeramente se debió realizar un estudio sobre las distintas técnicas de generación de Web Services. Se analizó la generación con enfoque descendente (top-down) o ascendente (bottom-up), para luego decidir cuál de los dos utilizar. En este caso, para la realización del proyecto se tomó la decisión de utilizar un enfoque descendente, es decir, crear el WSDL y a partir de éste el código. De esta forma el modelo de los servicios SOA no dependerá de las herramientas con las que se implemente, sino al revés y los clientes de los servicios pueden conectarse a ellos independientemente de si está implementado con Axis, con JWSDP, con .Net o con cualquier otra herramienta. Por consiguiente, se debió implementar una clase (Xmi2wsdl) que es la encargada de crear los WSDLs que se desean exponer. Por cada servicio que desea exponer un participante, se invoca a la función con el siguiente cabezal para crear dicho WSDL:

```
public static File generarWSDL(File dest, Participant participant, ServiceContract service, Port port,
        String ipNumber, String portNumber){
```

Para la creación de un WSDL, se realizó una investigación sobre los componentes que forman parte de éste, los cuales fueron descriptos en la Sección 2.5. Se analizaron las relaciones entre

dichos componentes y los elementos Java obtenidos a partir del parser SoaML to Java. A continuación se detalla la transformación resultante del análisis.

Para generar el componente types, a partir del ServiceContract recibido como parámetro en la operación *generarWSDL*, se busca la interfaz con rol provider asociada al mismo, obteniéndose a partir de ésta las operaciones. Por cada operación se toman los parámetros tanto de entrada como de salida los cuales son del tipo Message, definiendo así los tipos de datos relevantes que son utilizados en el intercambio de mensajes. En la Figura 5.8 se muestra gráficamente la transformación entre los elementos mencionados.

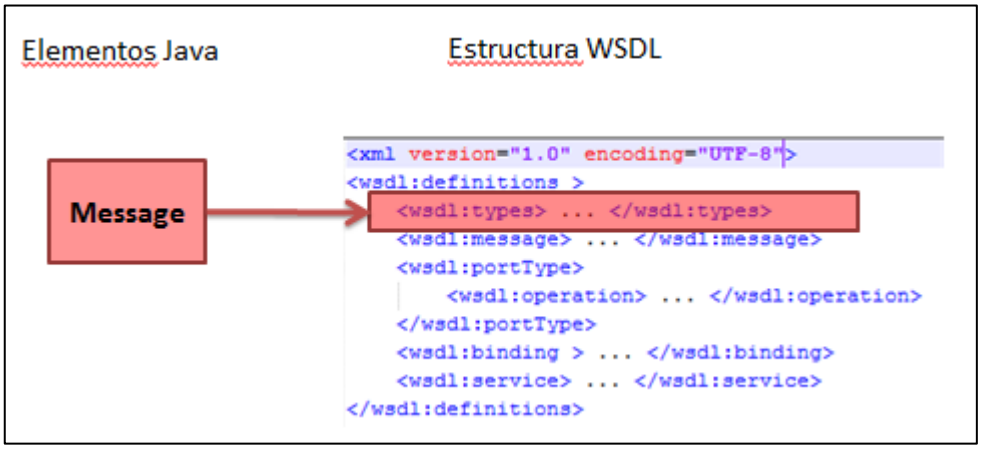


Figura 5.8. Mapeo de elemento Java a componente types de la estructura WSDL.

Una vez definido el componente types, pasamos a definir el componente messages. Éste representa un mensaje abstracto el cual es utilizado por el servicio para el intercambio con el consumidor. Por cada operación definida en la interfaz asociada al ServiceContract, se define un message que contiene los parámetros de entrada de dicha operación, el cual es denominado con el nombre del elemento Message más la extensión "Request". Además se define otro message que contiene el parámetro de salida de la operación, el cual es denominado con el nombre del Message mas la extensión "Response". Cada parámetro definido dentro de un message, debe ser de un tipo definido en el componente types. En la Figura 5.9 se presenta gráficamente la transformación mencionada anteriormente.

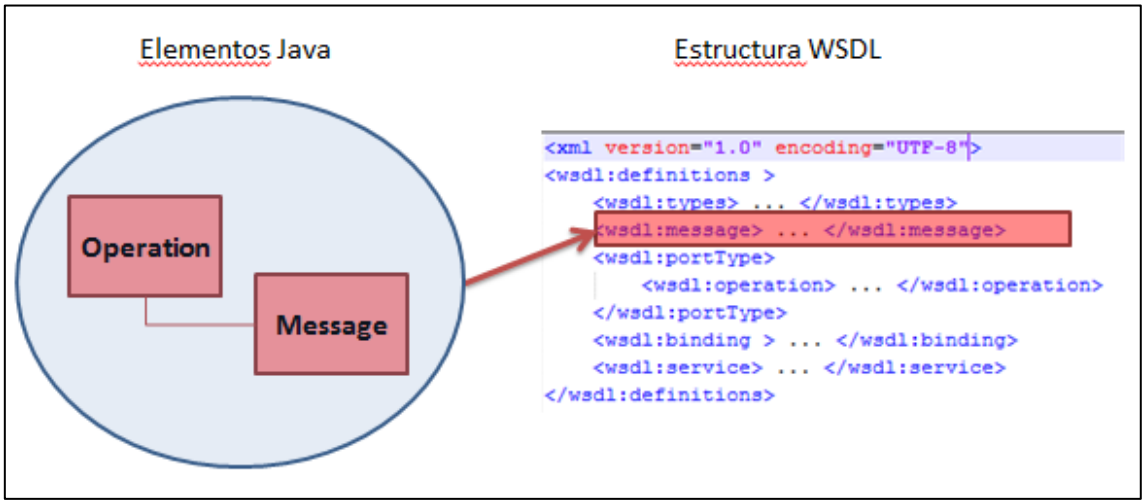


Figura 5.9. Mapeo de elemento Java a componente messages de la estructura WSDL.

Una vez definidos los componentes types y message, pasamos a definir el componete portType. Este componente funciona como una interfaz donde se define el conjunto de operaciones que van a ser provistas por el Web Services asociado a este WSDL. A partir del elemento Port recibido como parámetro en la operación *generarWSDL* se crea el componente

portType denominado con el nombre del Port más la extensión “PortType”. Dentro de este componente se definen las operaciones provistas por la interfaz asociada al Port. En la Figura 5.10 se muestras los componentes involucrados en la transformación descripta.

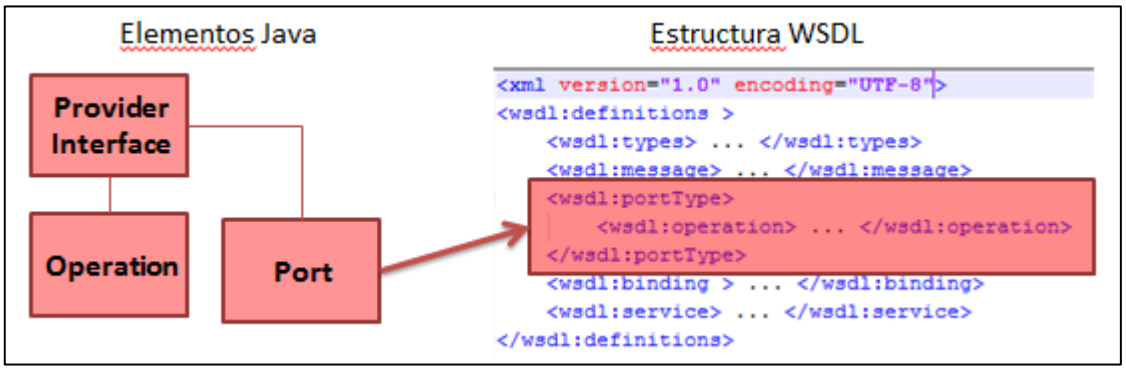


Figura 5.10. Mapeo de elemento Java a componente portType de la estructura WSDL.

El componente binding de un WSDL especifica el protocolo de transporte (SOAP en nuestro caso) utilizado para el intercambio de mensajes. Este componente se crea a partir del ServiceContract recibido como parámetro en la operación *generarWSDL* y es denominado con el nombre de dicho ServiceContract agregando la extensión “Binding”. Además se define como tipo de este componente el portType declarado anteriormente. En la Figura 5.11 se observa el mapeo definido para este componente.

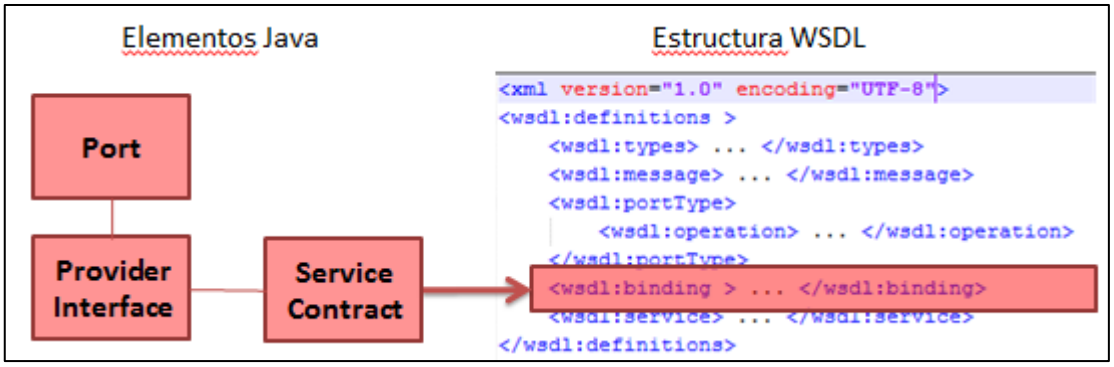


Figura 5.11. Mapeo de elemento Java a componente binding de la estructura WSDL.

Finalmente se define el componente service donde se asigna una dirección física, la cual será usada por los consumidores para invocar al Web Service. El nombre de este componente es determinado por el nombre del ServiceContract. Dentro de este componente se define el elemento port asociado al componente binding ya definido. Además en conjunto con la IP y puerto recibidos como parámetros por la operación *generarWSDL* se define la dirección física donde será publicado el Web Service. En la Figura 5.12 se muestra dicha transformación.

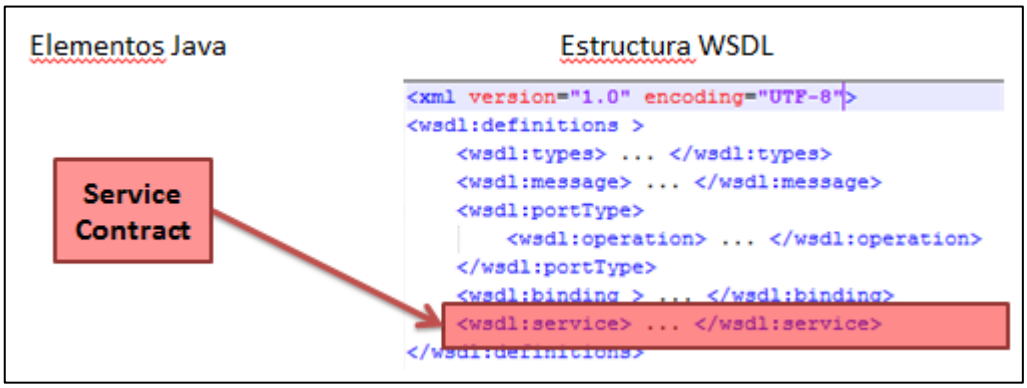


Figura 5.12. Mapeo de elemento Java a componente service de la estructura WSDL.

A partir de las reglas de creación declaradas anteriormente se implementó la operación *generarWSDL*, donde se añaden los elementos correspondientes en el archivo WSDL, a partir de los datos obtenidos en el "parseo".

Luego de tener generados los wsdls que exponen los servicios representados en el modelo, se debía generar las clases Java que lo implementen. Para ello se investigó las herramientas existentes para la generación de código Java a partir de un WSDL y sus distintas implementaciones. En el mercado existen muchos Frameworks para la generación de Web Services y Proxys (Clientes WS) tales como: Apache CXF, AXIS, AXIS2, JBossWS, Glassfish Metro entre otros. Para este desarrollo, se decidió utilizar el Framework Apache CXF, dado que soporta todos los estándares líderes de servicios web y proporciona un modelo de programación simplificado para el desarrollo de los mismos, junto con varios otros protocolos de aplicación. Además, otros motivos por los cuales nos decidimos a utilizar dicho framework es que presenta a nuestro entender una mejor generación de objetos, además de ser un framework de servicios de Software Libre. CXF nos ayuda a construir y desarrollar servicios utilizando JAX-WS como API de programación. Estos servicios pueden hablar una gran variedad de protocolos como SOAP, XML/HTTP, HTTP RESTful, o CORBA, y pueden trabajar sobre transportes como HTTP, JMS o JBI.

Analizando las necesidades que se nos plantean en el proyecto y las características que presenta CXF pensamos que es una muy buena opción a aplicar en el desarrollo para la generación de código. CXF trae una herramienta para crear esqueletos de servicios a partir de un WSDL, tanto para los clientes como para los servicios, llamada WSDL2Java. Cabe destacar que el desarrollo que es usado para la utilización del framework Apache CXF se encuentra modularizado para que fácilmente se pueda agregar otro modulo que realice otra implementación a partir de otro framework (AXIS, AXIS2, etc.).

Los artefactos Java generadas siguen un patrón de diseño que refleja los siguientes principios:

- Los Web Services se implementan utilizando la tecnología JAX-WS, incluyendo JAXB.
- Toda interfaz que expone las operaciones de un servicio, tiene una clase que la implementa con las anotaciones correspondientes.

El código Java generado se organiza en la carpeta de origen src para los proyectos Web Dynamic y en la carpeta de origen ejbModule para los proyectos Java EJB. En dichas carpetas se crea por cada servicio publicado la interfaz que contiene las operaciones del servicio. Estas interfaces son identificadas porque el nombre de la clase termina con el sufijo "PortType". También se crean las clases que realizan la implementación por defecto de las interfaces de servicio, estas clases son identificadas porque el nombre de las mismas termina con el sufijo "PortTypeImpl". Además de estas clases, se crea una clase que hace de server para publicar el servicio, y en caso de haber definidos en los servicios elementos complejos que son enviados como parámetros o como respuesta en los servicios publicados, se crean también como clases para permitir el correcto funcionamiento de los servicios pudiendo también utilizarse en la implementación de la lógica de negocio.

Debemos mencionar que para la implementación de la lógica de negocio que se desee asociar a los servicios publicados, la forma más prolija de hacerlo para el usuario sería que cree una carpeta de código de usuario (src.usr), y a posteriori una clase que tenga definida como operaciones todas las operaciones que expone el servicio logrando de esta manera desacoplar fácilmente la lógica de negocios del código generado. El proyecto también tiene generadas las siguientes carpetas:

- **WebContent** contiene las carpetas META-INF y WEB-INF, donde dentro de la carpeta WEB-INF se encuentran los archivos de configuración (web.xml, sun-jaxws.xml y beans.xml) y la carpeta lib que contiene librerías que son utilizadas en tiempo de ejecución.

- **wsdl** contiene los WSDLs y los esquemas XML (para los proyectos Java EJB no es creada esta carpeta y los WSDLs se encuentran dentro de la carpeta META-INF).

Creación de Proyecto Web Dynamic

Luego de usar CXF para la creación de las clases Java, fue necesaria la creación de los archivos de configuración complementarios para que los proyectos generados, al ser "deployados" expongan los servicios correctamente. Para ello, en caso de que la API de implementación fuera JAX-WS RI se creó el archivo sun-jaxws.xml y el archivo web.xml. La estructura del proyecto generado se muestra a continuación en la Figura 5.13.

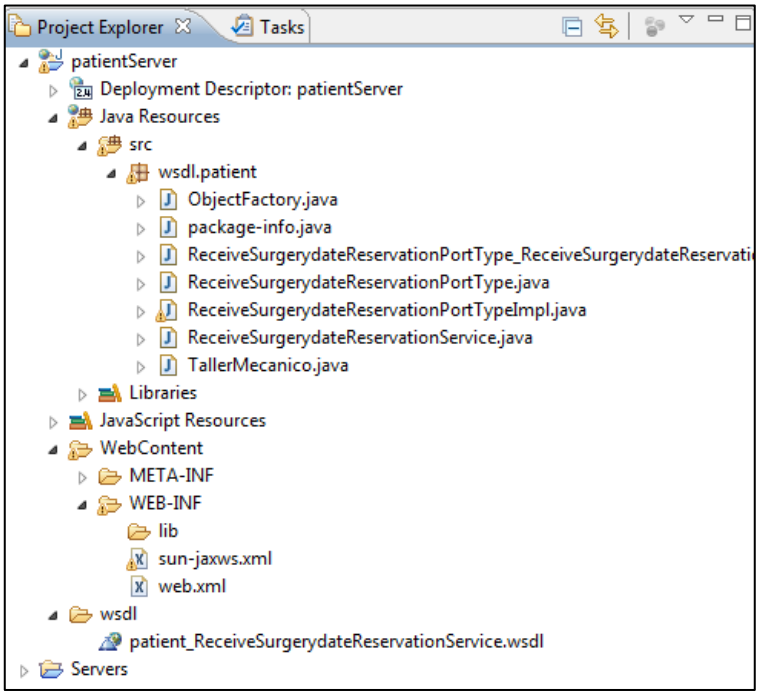


Figura 5.13. Proyecto Web Dynamic generado utilizando JAX-WS RI

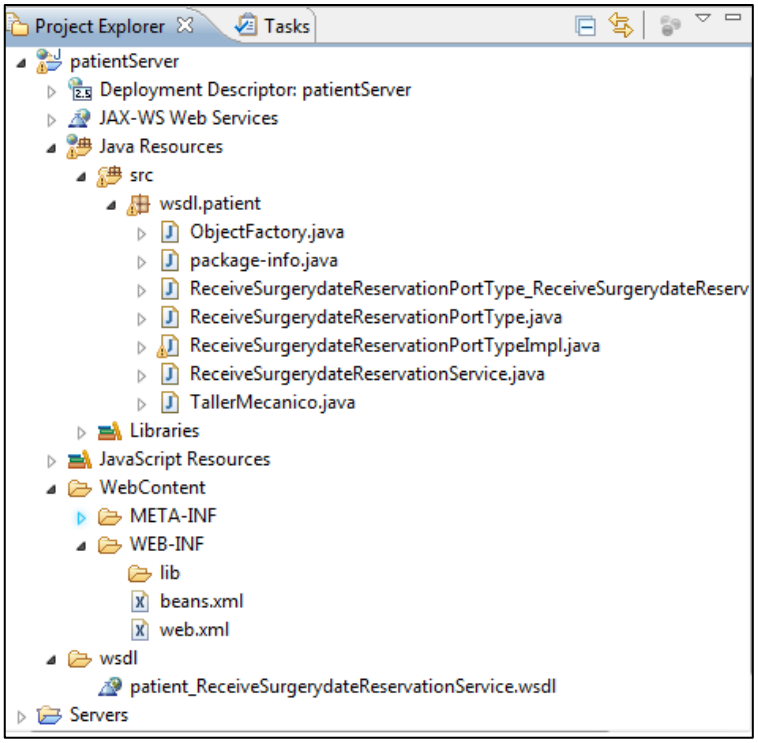


Figura 5.14. Proyecto Web Dynamic generado utilizando JAX-WS + Spring

Para el caso en que la API de implementación sea JAX-WS + Spring, se creó el archivo de configuración beans.xml y se modificó el archivo web.xml para utilizar como proveedor a Spring para la exposición de servicios. La estructura del proyecto generado se muestra en la Figura 5.14.

Creación de Proyecto Java EJB

Para la creación del proyecto Java EJB se encontraron dificultades extras, dado que no pudimos encontrar una herramienta que a partir de un WSDL generara las clases con las anotaciones EJB para exponer el servicio indicado por el WSDL. Por tal motivo, se decidió utilizar para la generación de las clases Apache CXF, aunque éstas no fueran generadas con anotación EJB. Luego se investigaron distintos plug-in para la manipulación y edición de clases Java como javassist [19], que es una biblioteca de clases para la edición de código de bytes de Java. Esta biblioteca permite a los programas Java definir una nueva clase en tiempo de ejecución y modificar un archivo de clase cuando se carga en la máquina virtual de Java (JVM). A diferencia de otros editores de bytecode similares, javassist proporciona dos niveles de API: nivel de origen y nivel de bytecode. Si los usuarios utilizan la API de nivel de código fuente, se puede editar un archivo de clase sin el conocimiento de las especificaciones del bytecode Java. Esta herramienta nos permitía modificar las clases generadas con CXF para agregarles la anotación necesaria para exponer los servicios deseados pero al momento de guardar los cambios, las clases Java editadas eran guardadas con formato .class, por lo que se debía agregar también un decompilador de código Java para pasar el archivo .class a .java. Por este motivo, se tomó la decisión de editar las clases generadas con CXF "manualmente", realizando la lectura de los archivos que se desean modificar y mediante un parser agregar el código necesario para exponer los servicios sobre un proyecto Java EJB.

En las clases que implementan los servicios, se agregó la anotación @Stateless con el nombre del servicio. Para lograr que el archivo Java compile, se debió agregar a los proyectos Java EJB las librerías de JBoss las cuales contienen la librería que implementa la clase javax.ejb.Stateless necesaria para utilizar la anotación. En caso de no tener agregado al Eclipse el Server JBoss para poder utilizar sus librerías, el generador crea las clases aunque éstas no estarán compilando y el usuario será el encargado de agregar manualmente las librerías para que el proyecto compile. La estructura del proyecto generado se muestra en la Figura 5.15.

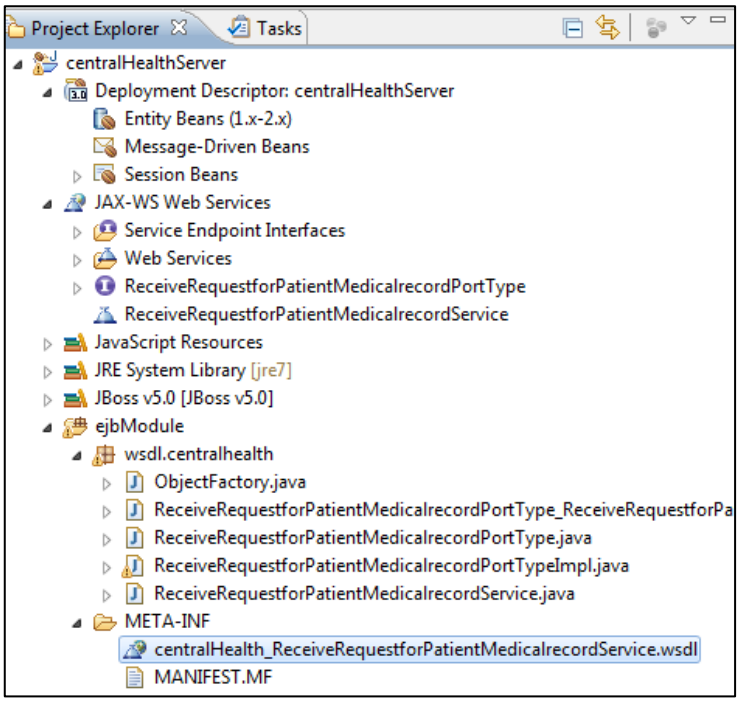


Figura 5.15. Proyecto Java EJB generado

5.4. 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 plug-ins entre otros.

La primera dificultad encontrada se dio al comenzar con la implementación del parser de modelos SoaML. Éste se realizó para archivos XMI en versión 2.0, pero al intentar asociar el parser con el plug-in de exportación a XMI realizado en el proyecto SoaML, nos encontramos con que el formato de los archivos XMI se generaban en versión 2.1, lo que cambiaba las anotaciones y tags del XML, por lo cual se debió agregar al parser el soporte para dicha versión XMI y se dejó desacoplado el mismo para que fácilmente se pueda agregar la implementación de una nueva versión XMI.

Otra dificultad fue no encontrar una herramienta de generación de código que generara anotación EJB a partir de un WSDL, esto ocasionó la perdida de gran tiempo en investigación y análisis para encontrar la mejor solución a dicho problema.

Una vez que comenzamos a realizar los “deploy” de los artefactos generados en el servidor de aplicaciones JBoss, surgió un inconveniente dado que tuvimos problemas con el manejo de librerías para publicar el proyecto Web con Spring. Al deployar en JBoss, se genera un conflicto entre las librerías de Spring y las librerías propias de JBoss. Se realizó la investigación necesaria para poder evadir dicho conflicto. Se probaron distintas alternativas de manera que a través de distintos archivos de configuración se pueda dar una jerarquía excluyendo las librerías que no eran de nuestro interés, pero no se logró una correcta solución. La solución más factible es trabajar con Maven, que permite fácilmente filtrar las librerías en cada proyecto. De todos modos esto implicaba demasiados cambios y más tiempo de investigación para la generación automática de código y escapaba a los intereses fundamentales del plug-in.

En la etapa de verificación nos encontramos como dificultad que la generación de modelos SoaML para poder realizar las pruebas con diferentes características, insumía mucho tiempo dado que era necesario generar todo el modelo con el plug-in de Eclipse SoaML, para luego recién poder realizar la verificación sobre nuestro proyecto.

La documentación encontrada para la generación de código de proyectos Java EJB en el entorno de Eclipse resultó escasa y poco fiable.

Por otra parte las constantes actualizaciones de Eclipse provocaron que el plug-in SoaML del cual partimos esté estabilizado en una versión determinada de Eclipse. Esta versión de Eclipse trabaja con una determinada versión de Papyrus la cual no provee diagramas de despliegue [20], los cuales eran necesarios para la creación del plug-in SoaML2Code por lo que se debió buscar la mejor alternativa para sobreponerse a esta limitante.

Finalmente nos encontramos con que la generación de proyectos Java se realiza de forma estándar, sin embargo, para la generación de proyecto Java Dynamic Web y proyectos Java EJB, se debe generar un proyecto Java y luego su correspondiente transformación. Esto genera un retardo en la implementación que a veces puede provocar que el proyecto generado quede desincronizado y al querer “deployarlos” aparezca un error. Esto se soluciona simplemente con un refresh o clean de los proyectos en cuestión.

6. Verificación del plug-in SoaML2Code

En la sección de verificación se resume los resultados obtenidos como consecuencias de las pruebas realizadas sobre las distintas versiones de generador de código Java EE y Web Services.

En la sección 6.1, se muestra y estudian los resultados obtenidos en lo referente a la ejecución de los diferentes casos de uso, con diferentes entradas (modelos SoaML). Por último en la sección 6.2 se realiza un resumen de la verificación realizada en el cual se cuantifican los casos de prueba realizados, corregidos y restantes.

6.1. Reporte de los casos de prueba ejecutados

Los casos de pruebas se dividen en dos partes, una que prueba casos de usos intermedios o complementarios que no tiene como salida código sino que determinan condiciones de generación y otra que son los casos de uso que tiene como salida código Java.

También se realizan dos tipos de prueba, unas que son de generación y otras de funcionamiento, o sea que se genere realmente lo que se quiere y luego que el código generado este apto para correr en servidores, pueda ser consumido el servicio expuesto o pueda consumir un servicio. En los casos de uso de generación es que estará probando el "parser" que es una parte fundamental.

Tenemos dos tipos de casos de usos, los que tiene como salida el código generado y los de uso intermedio que serían auxiliares de los primeros

Casos de uso que generar código:

- Crear Dinamic Web Project
- Crear cliente para servidor Dinamic Web Project
- Generar WSDL
- Crear Proyecto JavaEE o EJB
- Crear Objeto Java

Casos de usos "auxiliares" o "complementarios":

- Seleccionar proyecto papyrus para generar código
- Generar código desde archivo XMI
- Editar ventana de despliegue

Comenzamos a analizar la verificación en los casos de usos auxiliares o complementarios dado que un mal funcionamiento o error en alguno de estos podría derivar en la falla del caso de uso que lo invoca.

Seleccionar proyecto Papyrus para generar código

Esta funcionalidad permite seleccionar entre los distintos proyectos que se encuentran en el workspace como se muestra en la Figura 6.1.

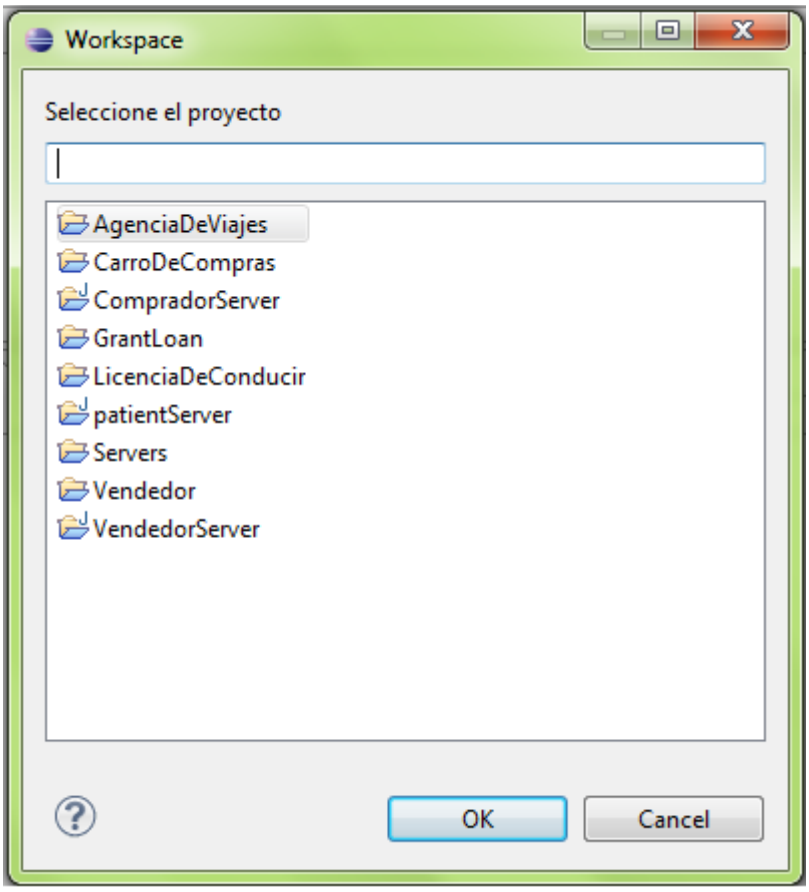


Figura 6.1. Lista de proyectos

Las pruebas realizadas sobre este componente fueron sencillas y tenemos 2 resultados posibles. Éstos son, uno de error y uno de éxito, el primero cuando el proyecto seleccionado no es un proyecto Papyrus, ante esta situación debería reportar un mensaje de error y no permitir continuar como se muestra en la Figura 6.2. El segundo tipo de resultado es el exitoso, el cual no se vería reflejado en un mensaje de éxito sino que por ser una funcionalidad intermedia, solo debería desplegar una ventana para editar las opciones de despliegue como se presenta en la Figura 6.3, y así continuar con la generación.

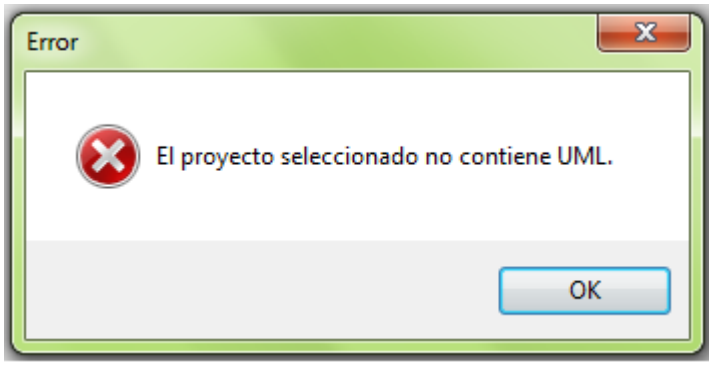


Figura 6.2. Error UML

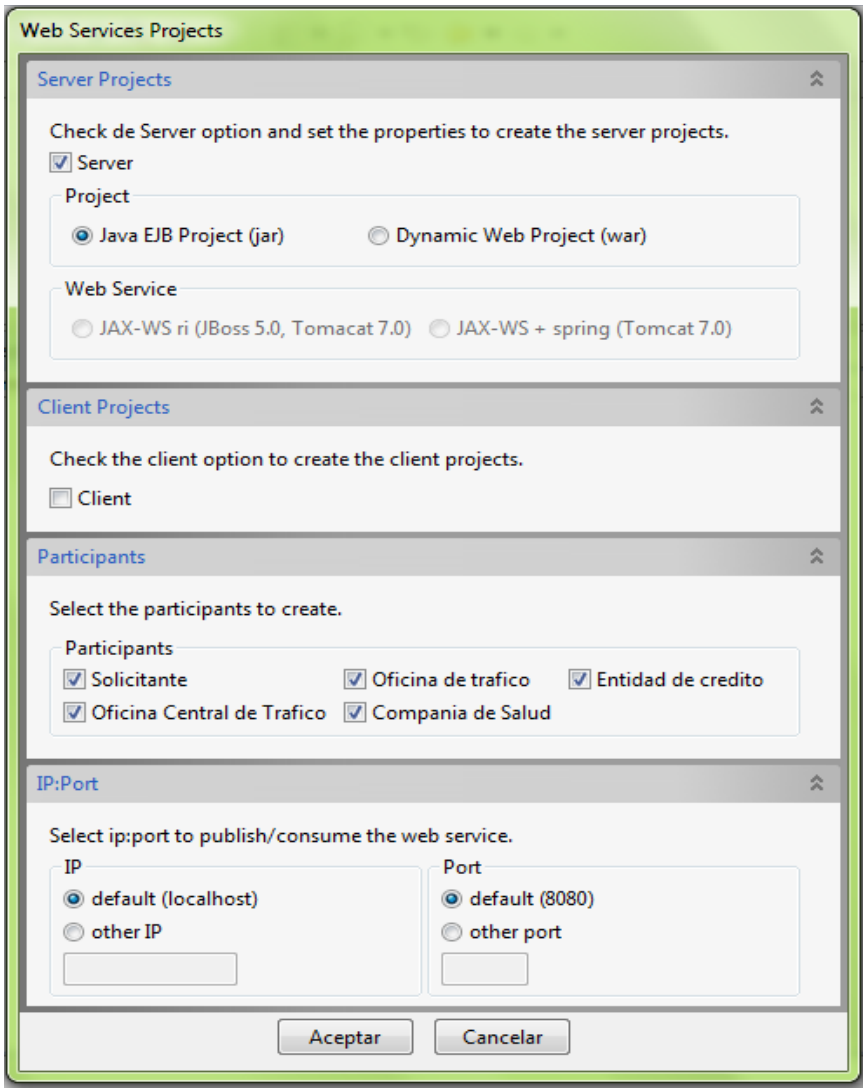


Figura 6.3. Ventana de edición de despliegue

En una primera versión del plug-in este componente no se encontraba por lo tanto no hay datos en el documento de verificación iteración I. En general como es una funcionalidad intermedia y muy básica los resultados fueron positivos.

Este componente es probado constantemente al probar las otras funcionalidades por ser una funcionalidad básica e intermedia. La única observación que puede hacerse es que permite seleccionar proyectos cerrados lo cual podría impedirse de ser necesario. A continuación se muestra gráficamente el resultado de la verificación.

Seleccionar formato del archivo de entrada (XMI o UML)

Este es al igual que el CU anterior, es un componente básico en el cual fue necesaria una modificación de la barra de herramientas de Eclipse como se ve en la Figura 6.4. Al igual que la anterior esta funcionalidad es usada continuamente y no genera mayores problemas. Sobre este componente en etapa de verificación se realizaron varias pruebas tanto en Iteración I de la verificación como en la iteración II.

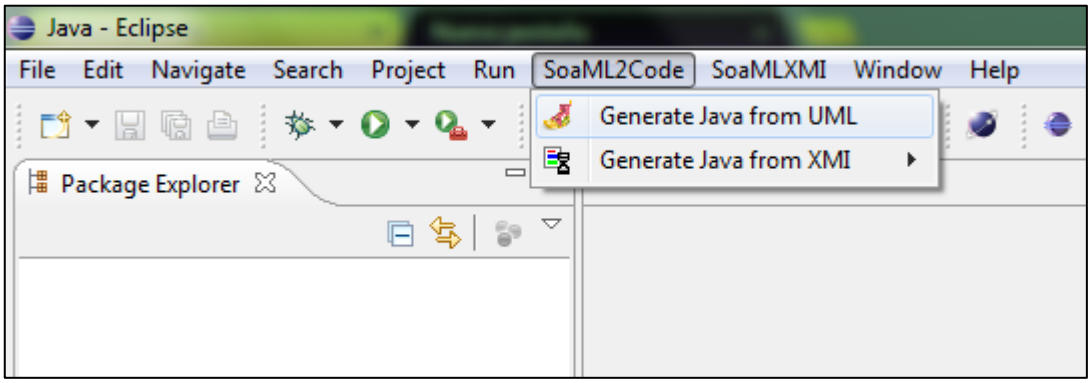


Figura 6.4. Menú Eclipse

Esta funcionalidad sufrió una modificación entre la iteración I y II de verificación dado que cambió el CU. Se realizaron tres tipos de pruebas para probar esta funcionalidad teniendo un resultado exitoso en todas las pruebas.

A continuación a modo ilustrativo se da una visión general de la verificación que se hizo exclusivamente sobre esta funcionalidad. Recordemos que esta funcionalidad es probada constantemente en la verificación de los demás componentes.

Notamos que los casos son todos exitoso pero recordemos que es un caso de uso muy básico.

Editar ventana de despliegue

La ventana de despliegue es la de la Imagen Ventana Edición de despliegue. Esta funcionalidad consiste en editar las opciones para la generación. Las pruebas que se realizan sobre esta funcionalidad son principalmente en el listado de los participantes, ya que esta ventana de despliegue permite escoger que participante generar y es el único componente que se carga dinámicamente ya que depende del archivo o proyecto Papyrus escogido en la pantalla anterior.

Como se detalla en el documento de Verificación Iteración II este caso de uso no tuvo un tratamiento particular a la hora de la verificación, ya que su principal función es redirigir al caso de uso indicado dependiendo de las condiciones elegidas dentro de las posibilidades brindadas por la ventana de edición para el despliegue.

La verificación se realizó de forma secundaria o en segundo plano al momento de verificar los casos de uso Crear Proyecto Web, Crear Proyecto JavaEE, Crear Proyecto Cliente.

Los participantes se listaron de manera adecuada en los diferentes casos, además de funcionar correctamente la redirección a los casos de uso indicados. Cabe recordar que esta funcionalidad no estaba en la Iteración I de la verificación por tratarse de una funcionalidad posterior a esta.

Crear Proyecto Web

A esta funcionalidad se llega cuando en el CU Editar ventana de despliegue se selecciona "Dynamic Web Project". A su vez también se decide qué tipo de "web services" se implementará, JAX-WS ri o JAX-WS + Spring (Apache CXF) y para que participantes se generará el código.

Los resultados en iteración I y II de verificación fueron diferentes debido que en la iteración I no se contaba con la posibilidad de implementar el "web services" en JAX-WS ri y esto no permitía "deployar" la aplicación en JBoss. No poder "deployar" en JBoss se marcó como una observación ya que el proyecto generado constaba con todo lo descripto en los archivos UML o XMI, además en Tomcat se levanta la aplicación correctamente y se puede consumir el servicio expuesto.

Por otro lado en iteración I, el plug-in no contaba con controles en cuanto a las herramientas utilizadas como por ejemplo, de no tener instalado Apache CXF el sistema no notificaba pero generaba un proyecto vacío o defectuoso. En tanto en la iteración II y final se realizaron las mismas pruebas que en iteración I las cuales anduvieron correctamente, manteniéndose la misma observación al momento de utilizar JBoss. En iteración II ya se cuenta con ventana de Edición de despliegue lo cual permite elegir otra implementación para el "web services", además de tener JAX-WS + Spring también se cuenta con JAX-WS ri.

En JAX-WS ri se probó "deployar" en JBoss y funciona correctamente, además se realizaron las mismas pruebas para JAX-WS ri que para JAX-WS + Spring teniendo resultados exitosos en todo los casos. Como vemos en los gráficos que continúan en iteración II se reportaron dos observaciones, una la ya conocida que es al momento de "deployar" el Web Service en JBoss con implementación JAX-WS + Spring se produce un error. La otra observación se produce al momento de querer consumir el servicio expuesto, cuando el servidor esta implementado con JAX-WS + Spring y se levanta en Tomcat pero si al momento de seleccionar la IP donde va a correr el servidor se selecciona localhost en lugar de poner la IP en formato numérico al momento de querer consumirlo desde otra IP (distinta a la que se encuentra el servidor) no se obtiene respuesta alguna. Se deja como observación porque si la IP se pone en formato XXX.XXX.XXX.XXX con X entre 0 y 9 se puede consumir correctamente desde cualquier otra IP.

A continuación se muestran los gráficos resultados de las pruebas puntuales que se hicieron a esta funcionalidad.

Resumiendo los resultados, los fracasos en iteración I se debieron a falta de control de restricciones. En iteración II se reportaron dos observaciones las cuales fueron detalladas anteriormente. También cabe mencionar que entre iteración I y II también se corrigieron algunos bugs no detectados en verificación I como por ejemplo la existencia de espacios en los nombres de las entidades que componían el archivo XMI o proyecto Papyrus por ejemplo en el nombre de participant, port, message types etc. lo cual fue solucionado con anterioridad a la iteración II.

Crear Proyecto JavaEE o EJB

A esta funcionalidad se llega cuando en el CU Editar ventana de despliegue se selecciona crear el server de tipo Java EJB Project.

En iteración I de la verificación este componente no estaba desarrollado por lo tanto no hay datos. En la iteración II se realizaron el mismo tipo de pruebas que los que se realizaron para los Dynamic Web Project salvo que corra en Tomcat por tratarse de un servidor web y no uno de aplicaciones como lo es JBoss.

Los casos probados fueron todos exitosos, la observación surge por no contemplarse que la ruta (path) donde está ubicado el workspce contenga un espacio en blanco. Genera aparentemente bien el proyecto pero a la hora de consumir el servicio da error.

A continuación se muestra como en el resto de los casos los resultados gráficamente.

Los resultados para las pruebas particulares que se realizaron a esta funcionalidad fueron satisfactorios, solo encontrándose la observación mencionada.

Crear Proyecto Cliente

Esta funcionalidad se ejecuta cuando en el CU de Edición de ventana de despliegue se elige generar el cliente para el Web Services. Se puede generar al mismo tiempo que el servidor o disjuntamente.

No se diferencia el cliente para cada tipo de servidor (Java EJB Project o Dynamic Web Project) ya que el proyecto cliente no sufre modificaciones y es igual en todos los casos.

En iteración I de la verificación el cliente se encontraba dentro de los proyectos servidor que se generaban, en esta iteración solo se contaba con servidores de tipo Dynamic Web Project y el cliente si se optaba por crearlo quedaba dentro del proyecto.

Para la iteración II se independizó el proyecto cliente del proyecto servidor, dando independencia tanto al momento de crearlo como al lugar donde se ejecuta (edición de IP y puerto).

En iteración I de verificación se realizan las mismas pruebas que al CU Crear Proyecto Web dado que era lo mismo salvo que debía contener las clases para consumir el servicio expuesto por este proyecto. Se verificó visualmente la existencia de las clases que consumiera el servicio

y además se probó que realmente lo consumiera. No se pudo probar consumirlo desde una IP diferente a la que corría el servidor ya que no había posibilidad de ingresar la misma.

Las pruebas realizadas en iteración II sobre la creación de proyecto cliente fueron pruebas un tanto básica en cuanto a probar la creación, se verifico que el proyecto Papyrus o archivo XMI seleccionado tuviera participantes, otro en que no los tuviera. En ambos casos el resultado fue exitoso.

Las pruebas principalmente fueron por el lado de probar si realmente consumía el proyecto servidor (una vez que este corriendo en alguno de los servidores) siendo este de cualquiera de los tipos posibles y probar consumir el servicio desde una IP distinta a la que se encuentra el proyecto servidor en ejecución. Los casos fueron todos satisfactorios salvo cuando el servidor corría con su IP en localhost y sobre el servidor web Tomcat, el cliente con el que se quiere consumir está en una IP distinta, este no obtiene respuesta alguna. Esta observación se realizó también en el CU Generar Proyecto Web.

En general los resultados fueron satisfactorios. No se muestran gráficos debido a que no revelarían nada significante dado la similitud con el CU Crear Proyecto Web.

Generar WSDL

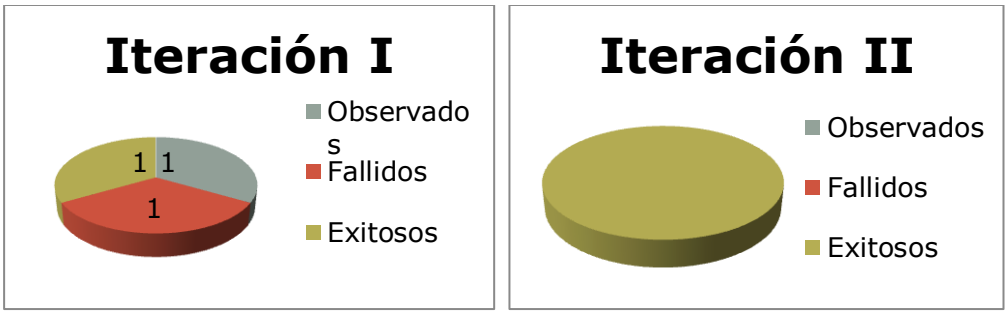
Este caso de uso se prueba de manera secundaria al probar las funcionalidades Crear Proyecto Web o Crear Proyecto EJB dado que no hay una opción para que el usuario directamente genere solo un WSDL. En todos los casos probados tanto en iteración I como en iteración II siempre se crea un WSDL dentro del proyecto. En el caso de los Dynamic Web Project dentro de una carpeta “wsdl”, en el caso de los proyecto JavaEE o EJB este se encuentra dentro de la carpeta “Meta-INF”.

Luego de verificar que el WSDL se creó correctamente se verifica también que todas las operaciones que tenía cada participante se encontraran en este y por último utilizado la herramienta de Eclipse Web Services Tools se comprobó que realmente con ese WSDL se podía consumir el servicio, lo cual fue satisfactorio en ambas iteraciones.

Generar Objeto Java

Esta funcionalidad se trató de manera similar a la de Generar WSDL dado que tampoco es posible generar un objeto Java en forma independiente. Se realizaron las mismas pruebas que se realizaron para Generar WSDL y el resultado en iteración I de verificación fue el mismo, para el caso que debía andar correctamente tuvimos la salida esperada. Para el caso de tener un archivo UML o XMI defectuoso o incompleto el caso no estaba contemplado, creando un proyecto defectuoso, tampoco se controlaba la existencia de las herramientas necesarias para que el proyecto “deployara” correctamente en los servidores.

En iteración II de la verificación el resultado es el esperado en todas las pruebas. Una visión gráfica de los resultados sería la siguiente.



6.2. Resumen de la verificación realizada

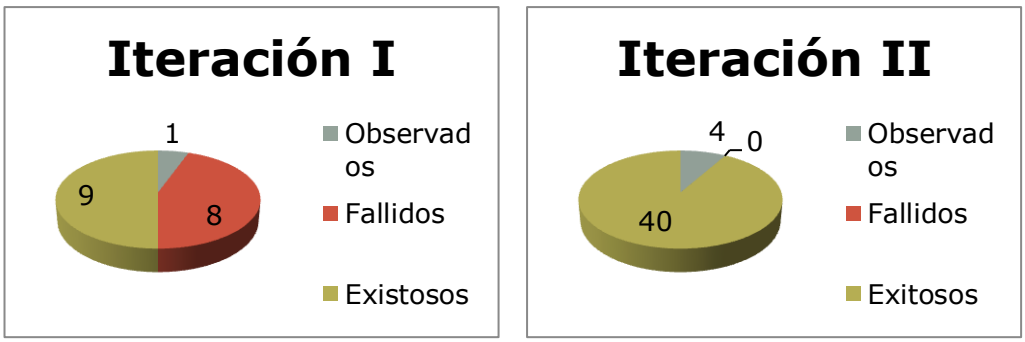
La verificación fue realizada en dos iteraciones, en las que se desarrollaron y ejecutaron casos de pruebas para los casos de uso incluidos en cada una de ellas.

En la primera iteración de la verificación se realizaron las pruebas para los casos de uso Crear Proyecto Web, Crear Proyecto Cliente, Generar WSDL, Generar Objeto Java, Seleccionar formato de archivo de entrada (XMI o UML). En la segunda iteración se agregaron nuevas funcionalidades que debían ser verificadas y se modificaron las anteriores en su totalidad por lo tanto se volvieron a verificar los casos de uso ya existentes. Por lo tanto la iteración II de la verificación estuvo compuesta por los casos de uso Crear Proyecto Web, Crear Proyecto JavaEE (o EJB), Generar WSDL, Generar Objeto Java, Crear Proyecto Cliente, Seleccionar formato de archivo de entrada (XMI o UML) y Seleccionar proyecto Papyrus para generar código.

A continuación se resume en números los casos de pruebas realizados exclusivamente a cada funcionalidad.

Tabla 6.1. Resumen casos de prueba ejecutados.

	Iteración I	Iteración II
Casos Ejecutados	**18**	**44**
Casos Observados	**1**	**4**
Casos Fallidos	**8**	**0**
Casos Exitosos	**9**	**40**



Con anterioridad se detalló en este documento y en los documentos de verificación en qué consisten cada uno de los casos exitosos, fallidos y observados de cada iteración.

Luego de las pruebas de generación también se realizaron pruebas de funcionamiento como se ha detallado. Estas pruebas consistieron en verificar que el código generado estaba listo para el despliegue (“deploy”) en los servidores mencionado en la ventana de edición de despliegue.

El resultado de estas pruebas se detalla en la siguiente tabla, donde el "tic" indica los casos exitosos y la cruz indica los casos fallidos.

Tabla 6.2 Resultados pruebas de funcionamiento

Proyecto	Proyecto EJB	Proyecto Web	
Servidor		JAX-WS ri	JAX-WS + Spring
JBoss	✓	✓	✗
Tomcat	No aplica	✓	✓

Con anterioridad en esta Sección del documento se explicó con más detalle en qué consisten estas pruebas de despliegue y por qué se obtuvo un resultado fallido en caso de haberlo.

Otro tipo de prueba de funcionamiento que se realizó en la verificación fue el de consumir los servicios expuestos por cada uno de los proyectos Server, también si el WSDL estaba bien construido y por ultimo si era posible consumir los servicios expuestos con el cliente generado. La tabla que continúa resume los resultados de este tipo de pruebas. Al igual que el resto de del resumen de la verificación los datos o pruebas en las que aquí se presentan en forma resumida están explicadas con detalle.

Tabla 6.3. Resultados pruebas consumir Web Service.

Proyecto	Proyecto EJB	Proyecto Web	
Cliente		**JAX-WS ri**	**JAX-WS + Spring**
Localhost (misma IP que el servidor)	✓	✓	✓
IP externa (IP distinta a la del servidor)	✓	✓	✗

En general el resultado de las pruebas específicas a cada funcionalidad es satisfactorio encontrándose algún problema con la incompatibilidad de tecnológica.

De las 44 pruebas realizadas 8 fueron pruebas que probaban en forma más integra todo los componentes del plug-in desarrollado. Los archivos para realizar estas pruebas fueron suministrados por el tutor.

Las pruebas incluyeron desde el listado de los proyectos para poder escoger uno hasta el despliegue del código generado y posterior consumo del servicio por este brindado. Cabe destacar que estas pruebas fueron exitosas lo que concluye un desarrollo bastante robusto. Dado que los archivos suministrados tenían una diferencia bastante considerable con respecto a los que se basó la implementación ya sea por un tema de variación en nombres de entidades SoaML como por ejemplo los nombres de participants, ports, operations, message, etc. que contenían espacios en blanco, como también por contener componentes que son de utilidad para este plug-ins pero que podían entorpecer la recolección de los datos necesarios.

Cabe mencionar que el tema de espacios en blanco que aquí se hace hincapié y parece un tema menor causó algunos problemas dado que para la generación de código se re-utilizan herramientas ya existentes y el tema de los espacios en blanco tanto en nombres como en la ruta donde estaba ubicado el workspace para el caso de los proyectos Papyrus como la ubicación del archivo XMI causó algún inconveniente.

7. Caso de estudio

Para probar y evaluar la herramienta desarrollada, se realiza un caso de estudio, el cual surge a partir del proceso de negocio (PN) “Admisión y Registro de Paciente para Cirugía Mayor Ambulatoria (CMA)” [8]. Este PN fue utilizado por la tutora de nuestro proyecto, en el marco de MINERVA [1] correspondiente a su doctorado y fue el utilizado por el grupo que realizó el plug-in SoaML. Se decidió utilizar este PN nuevamente para continuar y finalizar el ciclo del proceso. El material otorgado para la realización de nuestro caso de estudio, consta del plug-in SoaML para Eclipse para el cual realizamos la extensión, un archivo *XMI* y un archivo UML, ambos equivalentes en cuanto a la representación de la lógica del PN mencionado. Estos archivos sirven de entrada para la ejecución de la herramienta desarrollada. Sin embargo para una visualización completa del desarrollo del PN, la idea es utilizar el XMI como entrada del plug-in brindado. Así se podrá tener una representación gráfica del PN mediante el modelado, a partir del cual se genera un proyecto Papyrus, el cual será la entrada para probar y evaluar nuestra herramienta. Previamente a la utilización de nuestro plug-in, se debieron realizar ciertas modificaciones a los archivos. Las operaciones definidas en los mismos, si bien contaban con la especificación de parámetros, estos no tenían la definición de tipos, lo cual para comprobar el correcto funcionamiento de nuestra herramienta era fundamental.

7.1. 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 plug-in SoaML.

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.

La realidad del caso abarca más puntos, como por ejemplo, el chequeo de disponibilidad, pre condiciones para la cirugía, etc. Para nuestro estudio se simplifica, generando lo necesario para la interacción descripta anteriormente. En la Figura 7.1 se muestra el modelo BPMN (Business Process Modeling Notation, BPMN) donde se describe en forma completa la realidad del PN, de la cual se extrae el fragmento analizado.

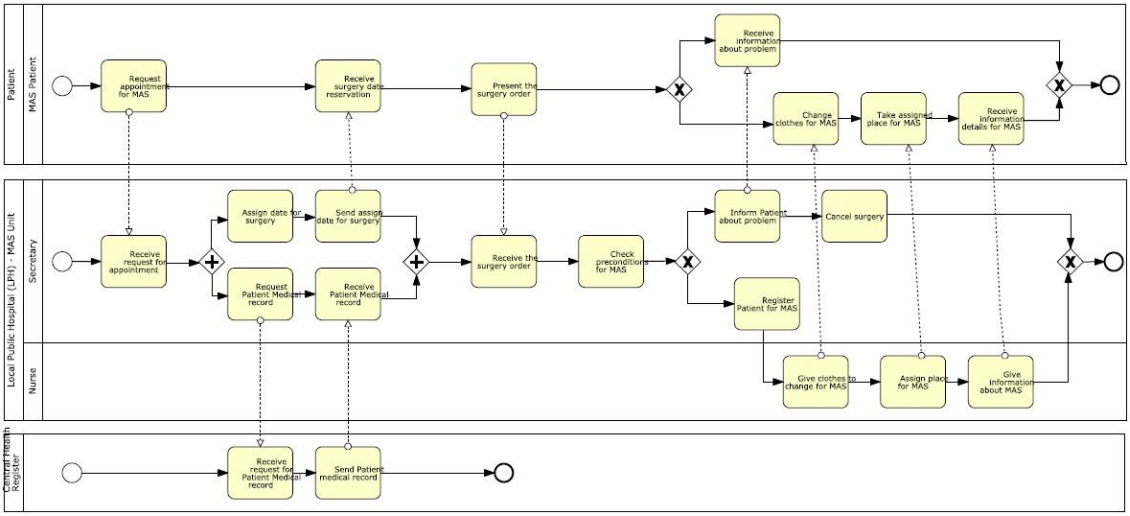


Figura 7.1. Modelo BPMN “Admisión y Registro de Paciente para Cirugía Mayor Ambulatoria” (CMA) [8]

7.2. Realización del caso de Estudio

Las siguientes imágenes muestran, a partir de la utilización del plug-in SoaML, una representación de lo que sería la realidad presentada en el punto anterior. Estas se corresponden con el proyecto Papyrus del cual partiremos. En la Figura 7.2 se ve una estructura de árbol donde se identifican los distintos componentes en su estructura correspondiente. En la Figura 7.3 se ven las relaciones en una forma más gráfica.

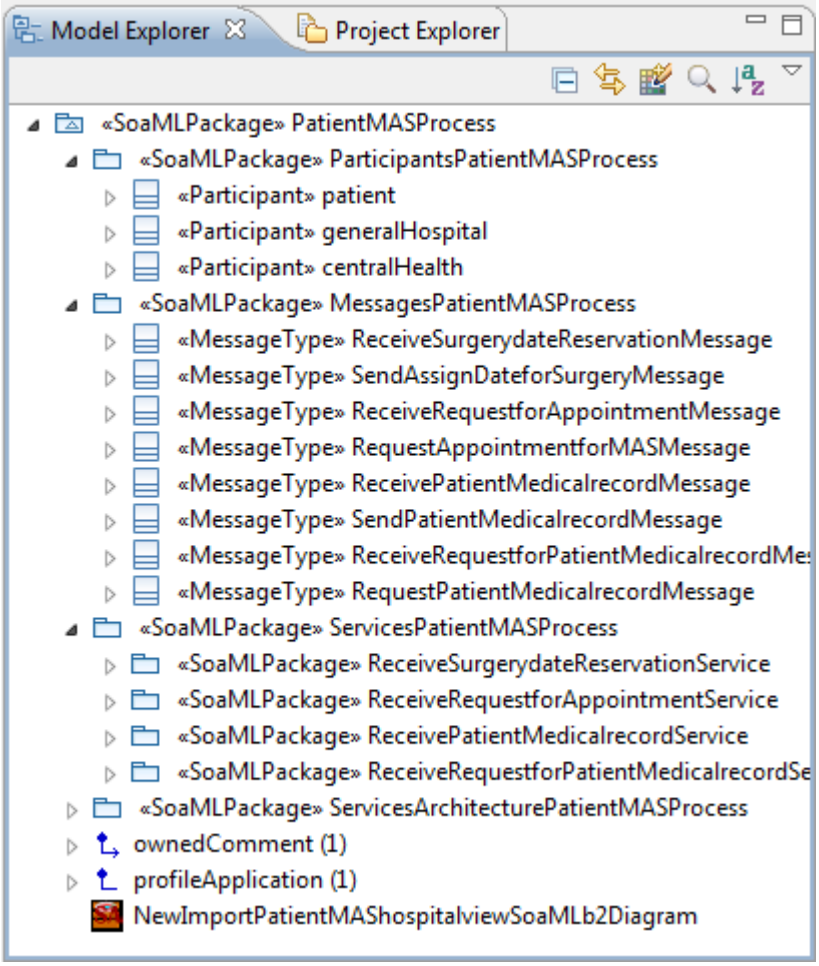


Figura 7.2. Componentes

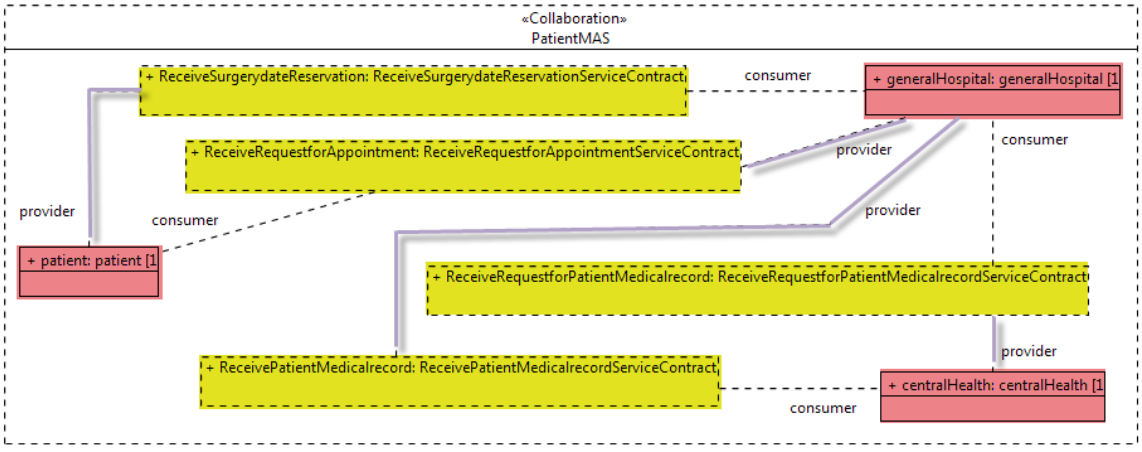
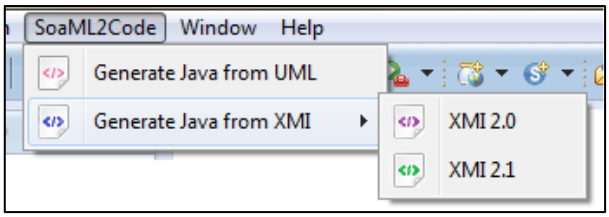


Figura 7.3. Relación de componentes

Para describir la utilización de la herramienta en la realización del caso de estudio, se utilizará un enfoque *top down*. Es decir, se describe la solución que se obtiene mediante el plug-in desarrollado y la relación de la misma con la Figura 7.3 que representa la realidad del PN. Luego se detallan los componentes obtenidos y los pasos necesarios y alternativas que permite la herramienta.



7.4. Menú de selección

En la Figura 7.4 se ve la interfaz de selección que se mencionó según si queremos partir de un uml o un xmi. Como se explicó anteriormente, la generación de los *consumer* y *provider* la realizaremos a partir del proyecto Papyrus presentado. Para ello debemos elegir en la barra de herramientas SoaML2Code-> Generate Java fromUML. A continuación aparece la lista de proyectos del workspace donde se está trabajando y se debe seleccionar el proyecto deseado.

Server

En la Figura 7.3 los recuadros rojos representan los participantes, aquellos que consumen y/o proveen operaciones. En la solución, cada uno es representado como un proyecto Java. Por lo tanto, se generan los siguientes proyectos: "patientServer", "centralHealthServer" y "generalHospitalServer". Los recuadros verdes, representan las operaciones, donde se puede ver, mediante las líneas de relación y las etiquetas en las mismas, quien las consume y quien las provee. En la solución obtenida, el código generado, implementa en cada participante, la o las operaciones que él provee. En cada proyecto se genera una clase, la cual su nombre termina en "*Impl",* donde se encuentra el código correspondiente a la implementación de cada operación, sin embargo la lógica que implementa cada una se debe ingresar manualmente, ya que esta no es representada en los diagramas.

La relación de "consumer" no se ve reflejada en la solución generada. Como se describió en punto anteriores, el código Java es generado a partir de un archivo wsdl. En cada proyecto generado se crea el archivo wsdl correspondiente, que provee la descripción de los elementos que formarán parte de cada uno de los servicios Web.

Solución.

En primer lugar debe estar seleccionada la opción de *Server* como se ve en la Figura 7.5. Hay tres tipos de generación de proyectos. El tipo que el usuario seleccione depende únicamente de las preferencias del mismo en cuanto a la estructura o configuración deseada.

Tipo 1

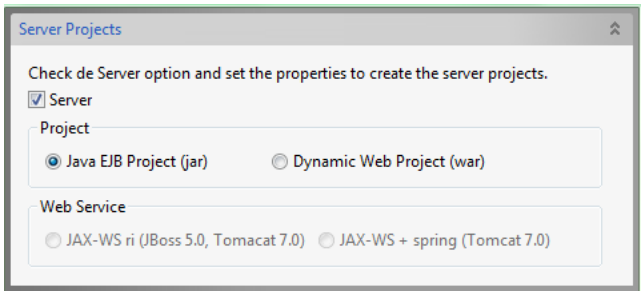


Figura 7.5. Generación proyecto Java EJB

En la Figura 7.5 se ve la selección que permite crear un proyecto Java EJB para cada participante. Para la correcta generación de este tipo de proyecto, es necesario tener instalado

el server JBoss en el Eclipse (ver Anexo Manual de Instalación). Este tipo de proyecto es deployado en JBoss como un jar.

Tipo 2

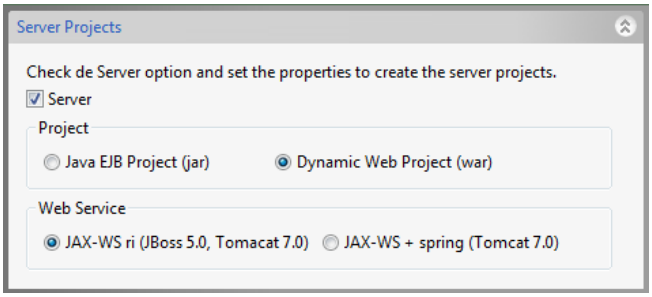


Figura 7.6. Generación proyecto Web JAX-WS ri

La Figura 7.6 muestra la selección para la generación de un proyecto Web con configuración de JAX-WS ri. Las clases Java generadas son las mismas que en el caso anterior. La estructura de carpetas es similar. El proyecto es deployado como war tanto en JBoss como en Tomcat. Para la correcta generación de este proyecto se debe tener agregado el plug-in de JAX-WS ri agregado en los plug-in de Eclipse (ver Anexo Manual de Instalación).

Tipo 3

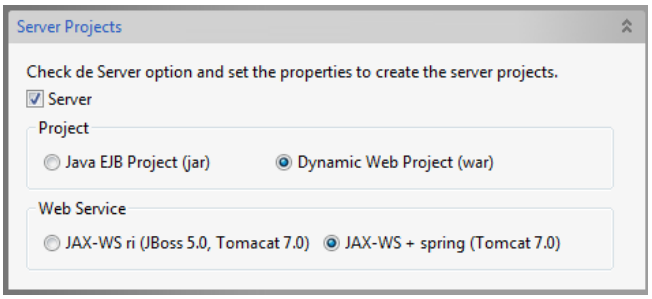


Figura 7.7. Generación proyecto Web JAX-WS spring

La Figura 7.7 muestra la selección para la generación de un proyecto Web con configuración spring. Las clases generadas son las mismas que en los casos anteriores. La estructura de carpetas es igual al caso anterior. La diferencia se da en los archivos de configuración web. El proyecto es deployado como war en Tomcat. Para la correcta generación de este proyecto se debe setear la configuración de CXF (ver Anexo Guía de Instalación).

Client

Las operaciones que provee cada participante, deben tener un cliente para poder consumirlas.

Solución

Cada cliente se representa con un proyecto Java, el cual tiene el nombre del participante que provee la operación seguido de "Client". Este proyecto tiene la lógica necesaria para poder invocar la operación en cuestión. Se genera una clase la cual su nombre finaliza con "_Client". En esta clase se implementa un *main* que permite la invocación de la operación. Nuevamente la lógica de negocio que permite manipular los parámetros y resultados se debe ingresar manualmente ya que esta no es representada en los diagramas.

En la interfaz para generar los proyectos Clientes simplemente se debe setear la opción "Client" como se muestra a continuación en la Figura 7.8.

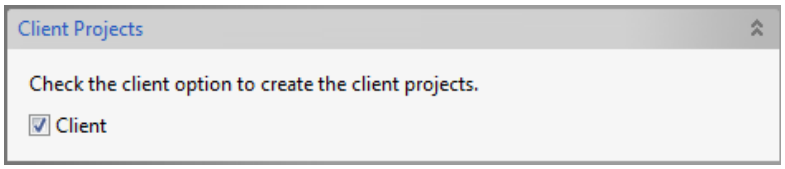


Figura 7.8. Generación Proyecto Cliente

Opciones

Selección de participantes.

No es necesario generar los proyectos de todos los participantes. Se listan como se ve en la Figura 7.9 los participantes en donde el usuario tiene la posibilidad de seleccionar únicamente los que desee generar.

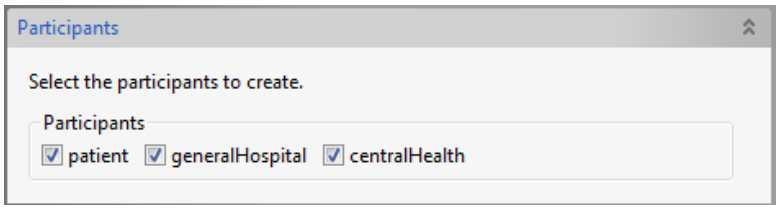


Figura 7.9. Selección de participantes.

Se genera un proyecto por cada participante seleccionado. Esta opción aplica tanto para la opción de *Client* como de *Server*.

Selección de IP y puerto.

Es posible que se desee generar los proyectos con un IP o con un puerto distinto al por defecto que es localhost:8080. Un ejemplo es el de generar un *Client* que consuma un servicio que esté corriendo en una IP distinta al localhost. En la Figura 7.10 se muestra como es el ingreso de una IP distinta. De igual forma se haría para un puerto distinto. Nótese que pueden cambiarse los dos valores o uno solo. Esta opción afecta tanto al *Client* como al Server. Si se quisiera generar cada uno de ellos con una IP y/o puerto distinto se deben generar por separado.

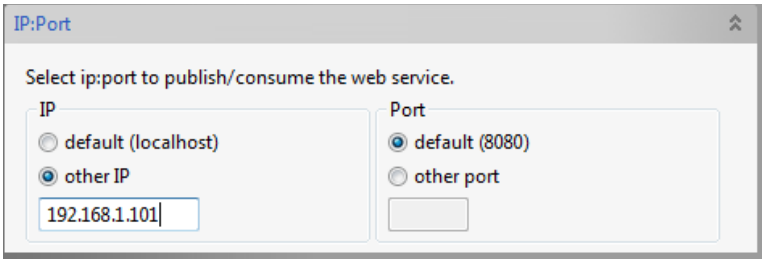


Figura 7.10. Selección de IP y Puerto.

Resultado generado

A partir de la descripción anterior de las posibles configuraciones para la generación del código, se toma una a modo de ejemplo para describir detalles del resultado obtenido. En la Figura 7.11 se muestra la configuración que se usará para generar un resultado el cual se describe a continuación.

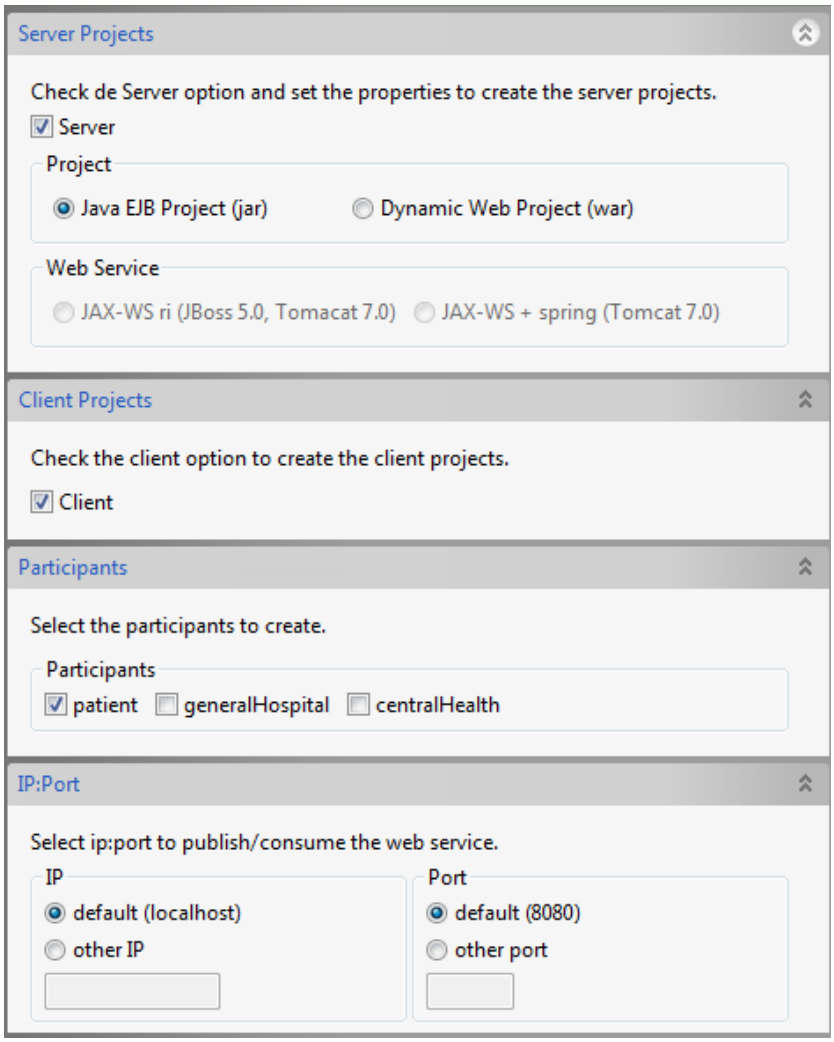


Figura 7.11. Configuración

En el proyecto *Server*, se encuentra la clase generada con el sufijo Impl donde se debe ingresar la lógica de negocios manualmente. A modo de ejemplo se presenta la Figura 7.12.

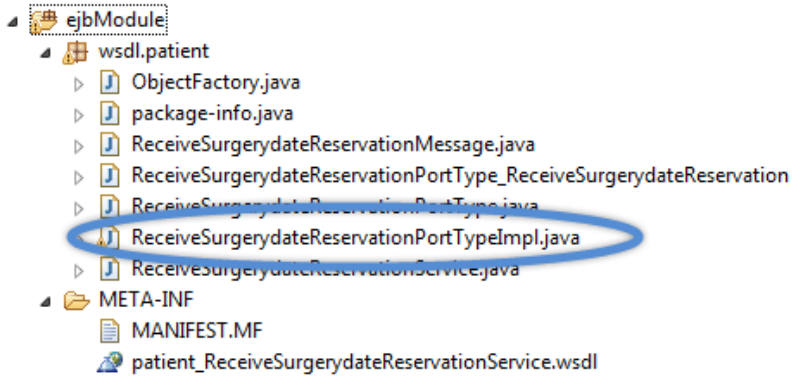


Figura 7.12. Código generado

En el segmento de código de la Figura 7.13 se puede ver la implementación de la operación que provee el paciente para esperar su fecha de cirugía. Una vez que esta operación es invocada por el *Client* el paciente tiene su fecha de cirugía, se agregó la lógica que imprime un texto con los datos obtenidos.

```
public wsdl.patient.ReceiveSurgerydateReservationMessage receiveSurgerydateReservationOperationReceive(
        ReceiveSurgerydateReservationMessage receiveSurgerydateReservationParameter1) {
    LOG.info("Executing operation receiveSurgerydateReservationOperationReceive");
    System.out.println(receiveSurgerydateReservationParameter1);
    try {

        wsdl.patient.ReceiveSurgerydateReservationMessage _return = receiveSurgerydateReservationParameter1;
        System.out.println("El paciente " + _return.getPatientName().toUpperCase() + " con número " +
        _return.getAffiliateNumber() + " tendrá la cirugía el día " + _return.getDate() + " a las " +
        _return.getHour() + " en el centro médico.");

        return _return;
    } catch (java.lang.Exception ex) {
        ex.printStackTrace();
        throw new RuntimeException(ex);
    }
}
```

Figura 7.13. Operación en el proyecto Cliente

En el proyecto *Client*, como se muestra en la Figura 7.14, se encuentra la clase generada con el sufijo *Client.* Allí se puede ver la invocación a la operación publicada. Se debe ingresar manualmente la lógica que permite utilizar el resultado obtenido y los parámetros con los cuales se invoca la operación.

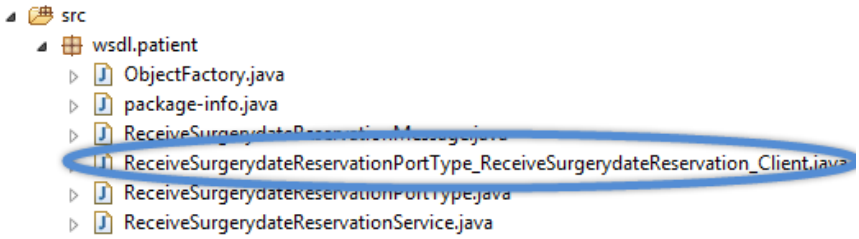


Figura 7.14. Código generado proyecto Cliente.

En la Figura 7.15 se muestra la lógica del *main* en donde se hace la invocación a la operación descrita, con los datos requeridos.

```
public static void main(String args[]) throws java.lang.Exception {
    URL wsdlURL = ReceiveSurgerydateReservationService.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();
        }
    }

    ReceiveSurgerydateReservationService ss = new ReceiveSurgerydateReservationService(wsdlURL, SERVICE_NAME);
    ReceiveSurgerydateReservationPortType port = ss.getReceiveSurgerydateReservation();

    {
    System.out.println("Invoking receiveSurgerydateReservationOperationReceive...");

    /* invocación a operación que busque fecha libre */

    wsdl.patient.ReceiveSurgerydateReservationMessage _receiveSurgerydateReservationOperationReceive_receiveSurgerydateReservationParameter1
                                                    = new ReceiveSurgerydateReservationMessage();
    _receiveSurgerydateReservationOperationReceive_receiveSurgerydateReservationParameter1.setPatientName("Jaime Rodriguez");
    _receiveSurgerydateReservationOperationReceive_receiveSurgerydateReservationParameter1.setAffiliateNumber(new BigInteger("187000"));
    _receiveSurgerydateReservationOperationReceive_receiveSurgerydateReservationParameter1.setDate("17/11/2013");
    _receiveSurgerydateReservationOperationReceive_receiveSurgerydateReservationParameter1.setHour("18:30");

    wsdl.patient.ReceiveSurgerydateReservationMessage _receiveSurgerydateReservationOperationReceive__return =
                port.receiveSurgerydateReservationOperationReceive(
                        _receiveSurgerydateReservationOperationReceive_receiveSurgerydateReservationParameter1);
```

Figura 7.15. Main en el proyecto Client.

En el archivo wsdl generado se puede ver el servicio definido en localhost.

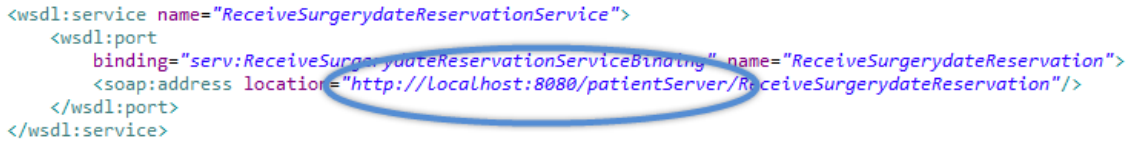


Figura 7.16. Archivo WSDL. Dirección de publicación.

7.3. Evaluación del caso de Estudio

La realización del caso de estudio se llevó a cabo sin problemas en la ejecución. El resultado que se obtuvo fue el correcto. Se pudo "deployar" en JBoss y consumir la operación con éxito.

La interfaz resulta simple e intuitiva. Brinda opciones que permite al usuario manejar con facilidad la herramienta, con una cantidad de pasos mínima que evita al usuario tener que manipular el código directamente para la configuración del producto generado.

8. Conclusiones y trabajo futuro

El capítulo se organiza en 4 secciones: en la 8.1 se exponen las evaluaciones del proceso de desarrollo, en la 8.2 se describen los aportes, en la 8.3 se comentan las posibles extensiones al producto en un futuro trabajo y por último en la sección 8.4 se exponen las conclusiones de todo el trabajo realizado.

8.1. Evaluación

Se logró llegar a concluir un producto de software que ofrece la mayoría de las características requeridas. Entre éstas destacamos que se logró desarrollar el generador de código tanto para exponer servicios mediante proyectos Java Web Dynamic y Java EE a partir de modelos SoaML, además se logró generar clientes que puedan consumir los servicios expuestos.

En el proceso de construcción de este producto, aparecieron variadas dificultades a la hora de decidir cuál iba a ser el mejor camino a seguir. Se investigaron y evaluaron varias alternativas distintas para la implementación del producto. Esta investigación implicó la realización de prototipos y el estudio de distintos frameworks (JDom, Apache CXF, Apache Axis, JBossWs, entre otros). Debido a la alta curva de aprendizaje tecnológico para la realización de algunos de estos prototipos, dicha etapa del proyecto se prolongó más de lo esperado. Se optó por pagar el costo de tiempo ya que se buscó obtener el mejor desarrollo para que el producto final lograra ofrecer las características deseadas y sea de fácil extensión y mantenimiento. Lo habitual para construir este tipo de plug-in en el entorno Eclipse, puede ser el desarrollo desde cero, pero luego de la evaluación de alternativas pudimos constatar que el tiempo requerido para cumplir con todos los requerimientos posiblemente sería mayor al esperado, lo que implicaría reducir el alcance afectando las características de usabilidad del producto final que se podrían ver empobrecidas.

Finalmente, después de realizar una gran cantidad de pruebas se llegó a la conclusión de que era conveniente utilizar el framework Apache CXF, ya que dadas las funcionalidades que presenta el producto final iba a ganar en potencial para futuras extensiones, obteniendo mejor desempeño y mayor facilidad para el mantenimiento del código generado.

Luego de tener desarrollado en gran parte el producto, se encontraron también dificultades para poder exponer los servicios correctamente en los servidores de aplicación elegidos en la etapa inicial, las mismas se resolvieron en base a la metodología de prueba y fallo a además de la investigación en la web de los distintos errores que se presentaban en el servidor de aplicación que se estaba usando en ese momento.

8.2. Aportes

Con el desarrollo de este producto, se aporta a la comunidad una herramienta de generación de código ejecutable a partir de modelos SoaML implementado para Eclipse. El mismo se enmarca en la extensión del plug-in SoaML de Eclipse que es una herramienta de modelado SoaML, logrando de esta forma con el plug-in SoaML2Code completar el ciclo de modelado y desarrollo bajo el estándar SoaML, permitiendo la generación de código desde un proceso de negocio, pasando por el diseño de los servicios a ser invocados desde el mismo, hasta el código para la implementación de los servicios, siguiendo una arquitectura dirigida por modelos (MDA), sin necesidad de utilizar ninguna otra herramienta externa a Eclipse. Este producto permite realizar una interpretación de modelos SoaML para posteriormente generar su representación en software, 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. Además permite también la generación de clientes Web Services para invocar los servicios expuestos y realizar pruebas sobre los mismos.

Otro aporte importante que cabe destacar, es que el producto desarrollado es independiente de la versión de Eclipse que se esté utilizando, lo cual aporta mucha practicidad a los usuarios que desean utilizar la herramienta. En base al caso de estudio realizado podemos concluir que la usabilidad de la herramienta permite generar software que representa el modelo SoaML desde el cual se esté partiendo casi en su totalidad, admitiendo cualquier realidad que involucre el manejo de servicios, faltando únicamente que el software generado mantenga las relaciones de los servicios que se desean utilizar desde un Participante, es decir, en el caso que un Participante utilice un servicio expuesto por otro Participante, se deberá generar un cliente que consuma el servicio expuesto y el usuario deberá agregar manualmente la dependencia al proyecto del Participante que utiliza dicho servicio.

Adicionalmente, otro aporte lo constituye la documentación realizada que deja sentada la investigación realizada sobre las herramientas existentes para la generación de código a partir de modelos, así como los frameworks que generan software y sus comparaciones, además de una guía de desarrollo (anexo documento Técnico) que constituye un manual con los pasos a seguir para construir rápidamente un WSDL. Consideramos que este aporte permite reducir los costos para este tipo de desarrollos, donde la curva de aprendizaje, las dificultades de implementación y el costo de investigación son altos.

Por último, se destaca como otro aporte la publicación en el CLEI 2013 por parte de las tutoras de la visión general de generación de código desde procesos de negocio incluyendo el Plug-in SoaML y el Plug-in SoaML2Code.

8.3. Posibles extensiones

En un ambiente tan amplio como lo es el desarrollo de software para exponer Web Services, donde los requerimientos y gustos de implementación son muy variados, observamos algunas posibles extensiones que darían una mayor variedad a la hora de implementar los servicios web y que por motivos de tiempo no forman parte del proyecto, las cuales comentaremos a continuación:

- Soportar la exposición de servicios web RESTful: la Transferencia de Estado Representacional (REST - Representational State Transfer) fue ganando amplia adopción en toda la web como una alternativa más simple a SOAP y a los servicios web basados en el Lenguage de Descripción de Servicios Web (Web Services Descripcion Language - WSDL). El framework Apache CXF utilizado para la solución, entre las características que presenta aparece el soporte de JAX-RS- que es la API Java para RESTful Web Services. Por lo tanto entendemos que no sería muy costoso agregar dicha extensión.
- Agregar algún otro framework para la generación de código, como por ejemplo Apache AXIS2 lo cual le daría al usuario más opciones a la hora de elegir como implementar la exposición de sus servicios web. La arquitectura del proyecto permite fácilmente agregar el uso de otro framework por lo que esta extensión se podría modularizar sin necesidad de alterar la implementación del framework Apache CXF.
- Otra extensión o mejora a realizarle al proyecto, sería el agregar el uso de Maven (es una herramienta de software para la gestión y construcción de proyectos Java) sobre los proyectos generados, dado que encontramos varias dificultades a la hora de gestionar las librerías que utilizan los proyectos generados al ser "deployeados" en los distintos servidores de aplicaciones en los que se probó.
- Soportar el estándar SoaML de forma completa, dado que del modelo de dominio SoaML del que partimos no se contemplaron algunos elementos que forman parte del mismo. Si bien los elementos más importantes son contemplados, esta extensión le daría mayor potencia al plug-in.

- Por último, otra extensión interesante sería soportar cualquier formato de archivo UML que represente un modelo SoaML. Las distintas herramientas existentes para el modelado SoaML, generan distintos archivos UML que representan un modelo, por lo cual, por el momento no es posible utilizar el generador de código a partir de cualquier herramienta de modelado, por ejemplo los archivos UML generados por la herramienta de modelado MagicDraw no son soportados ya que no tienen la misma composición que los que genera el plug-in SoaML que son en los que se basó la implementación del parser.

8.4. Conclusión

Se dedica esta última sección del documento para ofrecer una conclusión final sobre el proyecto realizado. A lo largo de todo el ciclo de vida del proyecto, se intentó buscar para cada requerimiento la solución más adecuada y práctica posible. En cada etapa del proyecto todos los recursos generados -documentación, fuentes, imágenes, guías- fueron respaldados en un repositorio, lo que permitió un correcto manejo del versionado que fue de gran ayuda a la hora de hacer modificaciones o recuperar previas versiones.

Se realizó el estudio de la especificación de SoaML, comprendiendo las características de cada elemento. Se estudió bibliografía y artículos involucrados con el modelado de arquitecturas orientadas a servicios para poder comprender el contexto de aplicación de SoaML, así como también el plug-in SoaML desde el cual partimos para realizar este proyecto. En cuanto al estudio inicial de tecnologías y frameworks, se recomendó por parte de las tutoras, el estudio de ModelPro y IBM Rational Application Developer , que son productos que generan código ejecutable a partir de modelos SoaML sirviendo estos de base para tomar ideas y características para conformar la solución de nuestro producto. Se dedicó gran parte del tiempo al estudio de los distintos frameworks involucrados en la generación de código para exponer servicios web, investigando cada una de las posibles alternativas de desarrollo. En todo este trayecto se documentaron las alternativas, con sus distintas características, esperando de esta forma colaborar con futuros emprendimientos semejantes. A la hora de tomar la decisión de cuál sería el camino a seguir, se tuvo que elegir principalmente entre construir un generador de código desde cero o reutilizar alguno de los frameworks estudiados para la generación de código, más específicamente la utilización de Apache CXF que nos pareció el framework más robusto entre los que se analizaron, aprovechando su conjunto de funcionalidades ya existentes. Se optó por ésta opción ya que está fue la alternativa que parecía hacer más factible el poder construir el total de los requerimientos establecidos.

Una vez elegido el camino a seguir, se intentó ser fiel a una metodología iterativa incremental que permitiera ir avanzando en la cantidad y calidad de las funcionalidades ofrecidas. Se realizaron en paralelo con la implementación, actividades tales como verificación y documentación de guías de desarrollo. Se utilizaron modelos SoaML ya construidos los cuales fueron provistos por las tutoras para generar casos de prueba para testear las funcionalidades desarrolladas y poder así verificar la calidad del prototipo entregado. Se incluyen como anexo los documentos de verificación donde se muestran las características y los resultados de las pruebas.

Se cumplió el objetivo principal del proyecto construyendo código ejecutable a partir de modelos SoaML que representa dichos modelos y expone los servicios de los elementos del modelo como Web Services. Dicha implementación es realizada en lenguaje Java EE para la plataforma Eclipse. Las alternativas de implementación que permite elegir el plug-in SoaML2Code son: generar proyectos Java Web Dynamic utilizando la API de implementación JAX-WS RI o JAX-WS + Spring y generar proyectos Java EJB. En cuanto a los distintos formatos de archivos que se soportan como entrada para interpretar los modelos SoaML, se logró aceptar archivos XMI en versión 2.0 y 2.1, y archivos UML generados por los proyectos de

modelado Papyrus. Se consiguió generar el código que representa el caso de estudio [8] propuesto de manera correcta utilizando la herramienta implementada. A partir de este caso de estudio se documentó cómo utilizar la herramienta y cómo realizar todas las funcionalidades.

Se nos presentaron varias dificultades a lo largo del proyecto, entre las que se destacan el no haber encontrado una herramienta que genere código Java con anotaciones EJB a partir de un WSDL, así como tampoco haber encontrado buena y suficiente documentación para las tecnologías de las alternativas evaluadas.

Consideramos que este proyecto fue concluido de forma satisfactoria por todo lo antes mencionado tanto por el proceso de desarrollo seguido, como por el producto logrado.
Esperamos que el producto generado a partir de este proyecto pueda constituir un aporte a la comunidad que utiliza arquitecturas dirigidas por modelos, permitiéndoles desarrollo de software a partir de modelos en Eclipse y que los pasos seguidos para construir el producto entregado dejen sentadas las bases para que hagan de guía para futuros trabajos.

9. Referencias

[1]Estado del Arte e Informe Final del Proyecto de Grado: Implementación de Estandar de modelado de Servicios SoaML como plug-in para Eclipse realizado en 2010.

[2] Edna D. LÓPEZ L., Moisés GONZÁLEZ G., Máximo LÓPEZ S., Erick L. IDUÑATE R. Proceso de Desarrollo de Software Mediante Herramientas MDA, http://www.iiisci.org/journal/CV$/risci/pdfs/C476AI.pdf.

[3] Soa Modeling Language (SoaML)- OMG, http://www.omg.org/spec/SoaML/1.0/Beta2/ [Último acceso 28/7/2012]

[4] OMG Unified Modeling LanguageTM (OMG UML), Infrastructure, versión 2.4

[5] OMG Unified Modeling LanguageTM (OMG UML), Superstructure, version 2.4

[6] http://www.omg.org/mda/ [Último acceso 28/7/2012]

[7] https://www.assembla.com/ [último acceso: 04/08/13]

[8] Andrea Delgado, Francisco Ruiz, Ignacio García-Rodríguez de Guzmán, Mario Piattini; "Business Process Service Oriented Methodology (BPSOM) with Service generation in SoaML". En: 23rd International Conference on Advanced Information Systems Engineering (CAiSE'11), Junio 2011, Londres, Reino Unido.

[9] Service-Oriented Architecture – Concepts, Technology and Design – Thomas Erl.

[10] http://www.omg.org/spec/XMI/2.1.1/, [Último acceso 28/9/2012]

[11] Clark Tony, Sammut Paul, Willans James. Applied Metamodelling A Foundation for Language Driven Development. Ceteva, 2nd edition, 2008.

[12] Meta Object Facility (MOF) Specification, http://www.omg.org/spec/MOF/1.4/PDF, [Último acceso 20/8/2013]

[13] http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[14] http://www.oracle.com/technetwork/java/javaee/overview/index.html

[15] http://portal.modeldriven.org/project/ModelPro (ModelPro™ Java EE Cartridge User's Guide v1.01) [Último acceso 10/2012]

[16] http://www.eclipse.org, [Último acceso 20/2/2013]

[17] http://www.jdom.org/ [Último acceso 11/2012]

[11]http://pic.dhe.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=%2Fcom.ibm.websphere.express.doc%2Finfo%2Fexp%2Fae%2Frins_config3.html

[18] https://community.jboss.org/wiki/ [Último acceso 20/05/2013]

[19] www.javassist.org/ [último acceso Febrero 2013]

[20] http://www.eclipse.org/forums/index.php/m/651787/ [Último acceso 03/2013]

[21] http://www.ibm.com/developerworks/ssa/rational/library/10/modelingwithsoaml-5 [Último acceso 09/2012]

[22] http://www.fing.edu.uy/inco/cursos/tsi/TSI2/2009/teorico/02-JavaEE.pdf [Último acceso 07/2013]

[23] Michael P. Papazoglou,Paolo Traverso, Schahram Dustdar, Frank Leymann. Service-Oriented Computing:State of the Art and Research Challenges, IEEE, 2007.

[24] DSDM: Desarrollo de Software Dirigido por Modelos. MDA y Aplicaciones, http://www.lcc.uma.es/~av/MDD-MDA/.

[25] T. Gardner, L. Yusuf. Explore model-driven development (MDD) and related approaches: A closer look at model-driven development and other industry initiatives, 2006. http://www.ibm.com/developerworks/library/ar-mdd3/

[26] Desarrollo de servicios con SoaML desde procesos de negocio en BPMN: metodología y automatización. Andrea Delgado, Ignacio García-Rodríguez de Guzmán, Francisco Ruiz, VII Jornadas de Ciencia e Ingeniería de Servicios (JCIS), JISBD, A Coruña, España, 2011

[27] Framework Apache CXF, http://cxf.apache.org/ [Último acceso 02/2013]

[28] SOAP, http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb [Último acceso 04/2013]

[29] REST, http://www.oracle.com/technetwork/articles/javase/index-137171.html [Último acceso 05/2013]

[30] Web Services, http://www.fing.edu.uy/inco/cursos/tsi/TSI2/2012/teorico/tsi2-07-web-services.pdf [Último acceso 04/2013]

[31] Spring Framework, http://www.springsource.org/documentation [Último acceso: 04/2013]

[32] JBoss, http://jboss.org [Último acceso: 04/2013]

Apéndices

[1] Tabla de comandos JAX-WS en JBossWs

Comando	Descripción
JBossWS - wsprovide	Genera artefactos portables JAX-WS y prove un contrato de servicio (WSDL). Es usada en un enfoque de desarrollo bottom-up.
JBossWS - wsconsume	Consume el contrato de servicio (WSDL y Schema files), y produce artefactos tanto como para servidor como para clientes. Es usada en un enfoque de desarrollo top-down y desarrollo de clientes.
JBossWS - wsrunclient	Ejecuta un cliente Java (que tiene un método main) usando el classpath de JBossWS.

[2] Lista de parámetros del comando wsconsume.

-h	Help	Show this help message
-b	Binding=<file>	One or more JAX-WS or JAXB binding files
-k	Keep	Keep/Generate Java source
-c	Catalog=<file>	Oasis XML Catalog file for entity resolution
-p	Package=<name>	The target package for generated source
-w	wsdlLocation=<loc>	Value to use for @WebServiceClient, wsLocation
-o	Output=<directory>	The directory to put generated artifacts
-s	Source=<directory>	The directory to put Java source
-t	Target=<2.0, 2.1,2.2>	The JAX-WS specification target
-v	Verbose	Show full exception stack traces
-l	Load-consumer	Load the consumer and exit (debug utility)
-e	Extension	Enable SOAP 1.2 binding extension

[3] Lista de parámetros del comando wsdl2Java.

-?,-h,-help	Displays the online help for this utility and exits.
-fe frontend-name	Specifies the frontend. Default is JAXWS. Currently supports only JAXWS frontend and a "jaxws21" frontend to generate JAX-WS 2.1 compliant code.
-db databinding-name	Specifies the databinding. Default is jaxb. Currently supports jaxb, xmlbeans, sdo (sdo-static and sdo-dynamic), and jibx.

-wv wsdl-version	Specifies the wsdl version .Default is WSDL1.1. Currently suppports only WSDL1.1 version.
-p [wsdl-namespace=] PackageName	Specifies zero, or more, package names to use for the generated code. Optionally specifies the WSDL namespace to package name mapping.
-sn service-name	The WSDL service name to use for the generated code.
-b binding-name	Specifies JAXWS or JAXB binding files or XMLBeans context files. Use multiple -b flags to specify multiple entries.
-d output-directory	Specifies the directory into which the generated code files are written.
-compile	Compiles generated Java files.
-client	Generates starting point code for a client mainline.
-server	Generates starting point code for a server mainline.
-impl	Generates starting point code for an implementation object.
-keep	Specifies that the code generator will not overwrite any preexisting files. You will be responsible for resolving any resulting compilation issues.
-wsdlLocation wsdlLocation	Specifies the value of the @WebServiceClient annotation's wsdlLocation property.
Wsdlurl	The path and name of the WSDL file to use in generating the code.

[4] Listado de parámetros del comando.

Option	Description
-d <directory>	Specify where to place generated output files
-b <path>	Specify external JAX-WS or JAXB binding files (Each <file> must have its own -b)
-B <jaxbOption>	Pass this option to JAXB schema compiler
-catalog	Specify catalog file to resolve external entity references, it supports TR9401, XCatalog, and OASIS XML Catalog format. Please read the documentation of catalog and see catalog sample.
-extension	Allow vendor extensions (functionality not specified by the specification). Use of extensions may result in applications that are not portable or may not interoperate with other implementations
-help	Display help
-httpproxy:<host>:<port>	Specify an HTTP proxy server (port defaults to 8080)
-keep	Keep generated files
-p	Specifying a target package via this command-line option, overrides any wsdl and schema binding customization for package name and the default package name algorithm defined in the

	specification
-s <directory>	Specify where to place generated source files
-verbose	Output messages about what the compiler is doing
-version	Print version information
-wsdllocation <location>	@WebServiceClient.wsdlLocation value
-target	Generate code as per the given JAX-WS specification version. version 2.0 will generate compliant code for JAX-WS 2.0 spec.
-quiet	Suppress wsimport output

[5] Tabla comparativa de las tecnologías analizadas.

	XML4J	**Xerces**	**JDOM**
Factibilidad	Si (según documentación, no se experimentó con prototipo)	Si (dado que JDOM y XML4j trabajan con esta herramienta)	Si (Comprobado empíricamente)
Costo de aprendizaje	Bajo	Bajo	Bajo
Tiempo de desarrollo	No se tiene el dato	--	Nivel de abstracción mayor por lo tanto más simple que trabajar directamente con Xerces
Documentación	--	--	--

[6] Tabla comparativa de características generales de generadores de código.

Caracteristicas	**Axis2**	**CXF**	**JBossWS**
Basic Profile 1.1 Compliant	X	X	X
Easily Create Services from POJOs	X	X	X
Open Source	X	X	X
RPC-Encoding			X
Spring Support	X	X	
REST Support	X	X	
Eclipse Plug-in	X	X	X
Hot Deployment	X	X	X
SOAP 1.1	X	X	X
SOAP 1.2	X	X	X
Streaming XML (StAX based)	X	X	
WSDL 1.1 ->Code (Client)	X	X	X

WSDL 1.1 ->Code (Server)	X	X	X
WSDL 2.0 ->Code (Client)	X	X	
WSDL2.0 ->Code (Server)	X	X	
Client-side Asynchrony	X	X	
Server-side Asynchrony	X	X	
Policy-driven code generation	X	X	

[7] Tabla comparativa de las tecnologías aplicadas a generadores de código relacionado a los servicios web.

Feature	Axis2	CXF	JBossWS
JAX-RPC			X
JAX-WS	X [1]	X	
JAX-RS		X	
JSR 181	X	X	X
JSR 181 on Java 1.4			X
SAAJ (1.2/1.3)	X	X	X
JSR 109			
JBI		X	

[1] Not JAX-WS TCK compliant due to lack of JAX-WS tooling

10. Anexos

· Documento de Estado del Arte

· Documento de Casos de Uso

· Documento de Arquitectura

· Documento Técnico

· Documentos de Verificación

· Guía de Instalación

· Manual de Usuario