Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Departamento de Ingeniería Eléctrica

Sección de Computación

Monitoreo de Procesos de Negocios con UML

Tesis que presenta

Lucio Daniel Castelán Vega

para obtener el grado de

Maestro en Ciencias

en la especialidad de

Ingeniería Eléctrica
opción Computación

Director de la tesis:

Dr. José Oscar Olmedo Aguirre

México, D.F. **Julio 2005**

Resumen

En la actualidad se ha reconocido que para mejorar la calidad de los sistemas de automatización de procesos de negocios se debe mejorar considerablemente la eficiencia del monitoreo de datos críticos en forma altamente utilizable lo cual permitirá tomar mejores decisiones en menor tiempo. Para ello es necesario registrar y presentar para el usuario no experto información crítica en forma comprensible, detallada y accesible.

En esta tesis se propone MS4WS, un sistema para el monitoreo de servicios Web utilizando diagramas UML de secuencia para mostrar el desarrollo de actividades de los procesos de negocio.

MS4WS permite que las interacciones entre los colaboradores de un proceso de negocio sean visualizadas usando lenguajes estándares y herramientas ampliamente disponibles en la Web. El nivel de abstracción alcanzado por usar diagramas UML facilita la comprensión del desarrollo de las actividades así como su validación sin que el usuario requiera tener conocimientos formales de computación o conocer los detalles relativos al sistema operativo o a la infraestructura de comunicaciones.

Entre las aportaciones que ofrece MS4WS se encuentra el monitoreo de las actividades comerciales utilizando una descripción de muy alto nivel que facilita la interpretación de datos importantes obtenidos de las peticiones de servicios y de sus respuestas. Además, MS4WS ofrece un marco en el cual se pueden integrar mecanismos para visualizar información estadística así como de controlar la ejecución de los procesos de negocio.

Abstract

With the aim of improving the quality of business process automation systems, it has recently been recognized the need of improving the efficiency of critical data monitoring in a highly usable manner to take better decisions in shorter time. To this end, it is necessary to collect and present to the non-expert user critical data in a clear, detailed and accessible manner.

In this thesis, we present MS4WS, a monitoring system for Web services that uses UML sequence diagrams to show the development of the activities of a business process. MS4WS uses widely available standard languages and tools in the Web to visualize the interaction among the activities of the business partners. The abstraction level reached by using UML diagrams facilitates the understanding of the development of the business activities and their validation, with no requirements for the users to possess formal knowledge in computing or to know the details of the operating system or the communication network used.

Among the contributions of MS4WS, the monitoring of commercial activities using high-level descriptions simplifies the interpretation of relevant data obtained from the Web service requests and their responses. Besides, MS4WS offers a framework in which mechanisms to visualize statistical information can be accommodated along with graphical controls to guide the execution of a business process.

Dedicatorias

Este trabajo es para cada uno de los integrantes de mi familia, quienes me apoyan y sirven de guías en mi vida...

Agradecimientos

- A mi mamá, Guillermina Vega Casteñeda, quien no se cansa de enseñar la manera de cómo afrontar situaciones de la vida y por ser un ejemplo de lucha, perseverancia y determinación en esta vida. Gracias mamá.
- A mi papá, Lucio Castelán Ibarra, quien siempre me brinda su apoyo incondicional y lo tengo como un ejemplo de persona en esta vida. Gracias papá.
- A mis hermanos, Reyna, Juan, Rosa y Dulce, quienes me han ayudado en todo y son ejemplos de personas en mi vida. Además de ser mis mejores amigos.
- A mis súper sobrinos, Carlos David y Carolina, con quienes me divierto y recuerdan lo especial y necesario que es jugar y sonreír. Además me recuerdan mi infancia junto a mis hermanos y amigos.
- A mi cuñada, Alicia, quien considero como una hermana más y por estar junto a mi hermano.
- A mis cuñados, David, Mateo y Marco, por estar junto a mis hermanas, cuidar de ellas y por tenerlos como nuevas amistades en las que puedo confiar.
- A mis amigos y compañeros, que son muchos, a quienes les agradezco todo el tiempo que pasamos juntos y las horas y horas de entretenimiento que pasamos jugando.
- A mi asesor de tesis, Dr. José Oscar Olmedo Aguirre, quien aportó excelentes ideas a esta tesis y por el apoyo otorgado en todo este tiempo que fue mi asesor.
- A Sofi, quien es secretaria de la sección de computación, por dedicarnos bastante tiempo para auxiliarnos de excelente manera en cuestiones administrativas. También por ser quien se preocupa y mantiene la relación de convivencia entre todos los integrantes de la sección.
- A mis revisores de tesis, Dr. Pedro Mejía Álvarez y Dr. Jorge Buenabad Chávez, quienes me ofrecieron su apoyo para revisar mi tesis y me señalaron errores que me ayudaron a mejorarla.

- Al equipo de trabajo, Giner, Ulises, Anahi y César, encabezados por el Dr. Olmedo, con quienes pasé buenos momentos.

- A los compañeros administrativos de la biblioteca del departamento de Ingeniería Eléctrica, quienes nos auxiliaron de buena manera cuando necesitamos encontrar trabajos de investigación.

- Al equipo de fútbol de Computación, somos como 30 integrantes, con quienes he vivido buenos y malos momentos pero sin duda alguna, el convivir con todos ellos me es de gran ayuda.

- Al Dr. Jorge Buenabad Chávez, quien ofrece, a todos los integrantes del equipo de fútbol, su apoyo tanto académico como moral. Además de pertenecer también al equipo y pasar junto a nosotros buenos y malos momentos que tenemos en el fútbol.

- Al Centro de Investigación y de Estudios Avanzados del IPN, CINVESTAV-IPN, junto con la Sección de Computación del departamento de Ingeniería Eléctrica, por ofrecerme las instalaciones y recursos necesarios para realizar este trabajo de investigación.

- Al Consejo Nacional de Ciencia y Tecnología, CONACyT, por el apoyo económico otorgado en el periodo Enero 2003 - Agosto 2004.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Monitoreo de procesos de negocios

Con el surgimiento del comercio electrónico que hace posible la conducción de negocios en Internet, ha surgido también la necesidad de que los medios de apoyo para la toma de decisiones estratégicas mejoren radicalmente. Mientras que las tecnologías de comercio electrónico han favorecido el desarrollo de las prácticas comerciales, la disponibilidad de recursos para realizarlas se ha limitado considerablemente. Dichas limitaciones se refieren principalmente al tiempo disponible para realizar operaciones comerciales, porque de su reducción depende el aumento en el número de operaciones.

Para mejorar la calidad de los sistemas actuales de automatización de procesos de negocio se está reconociendo el papel fundamental que juegan las decisiones que toman los administradores de servicios. Esta observación se contrapone a la tendencia hacia una automatización creciente en los negocios. Usar información reciente puede mejorar considerablemente la eficiencia de los procesos al revelar datos críticos en forma altamente utilizable para los administradores quienes podrán como consecuencia tomar mejores decisiones en menor tiempo. Para ello es necesario registrar y presentar para el usuario no experto información crítica en forma comprensible, detallada y accesible. Un administrador, por ejemplo, al observar el desarrollo de una operación comercial importante, puede identificar y tomar medidas ante situaciones excepcionales que pueden alterar el curso, continuidad y terminación de dicha operación. Entre las medidas que puede tomar, el ad-

ministrador puede enviar notificaciones de las condiciones observadas así como modificar o incluso detener el curso de la operación.

La recolección de datos y de su presentación en forma clara y concisa conlleva, sin embargo, varios problemas entre los que podemos destacar:

1. **Recopilación de la información.** De acuerdo a las entidades participantes en un proceso de negocio, los datos generalmente se agrupan por sus relaciones lógicas así como por su ordenamiento temporal. El retraso en la recopilación de los datos puede disminuir significativamente su valor para los usuarios. Por otra parte, aún cuando la información proveniente de alguna fuente esté disponible en forma casi inmediata, subsiste el problema de su publicación o distribución.

2. **Presentación de la información.** Suponiendo que los administradores pueden acceder a la información en forma casi inmediata, aún queda el problema de elegir una forma eficaz de presentación. En el comercio electrónico de negocio a negocio, las interacciones que mantienen entre sí los participantes de una operación comercial son en general muy complejas. Por tal motivo, la forma de presentar dichas interacciones, tales como el intercambio de mensajes entre las partes, debe utilizar alguna forma de diagramación que permita identificar tanto a los participantes como a sus relaciones. En comparación, las descripciones puramente textuales forzarían a leer de principio a fin la información presentada con el fin de extraer aquella que es relevante.

3. **Identificación de contexto.** Aún utilizando alguna forma de diagramación, la representación utilizada no debe incluir toda la información disponible ya que ello impediría reconocer la forma como están organizados tanto los participantes como la comunicación que sostienen. La representación utilizada debe mostrar la estructura subyacente del sistema de manera que permita formular consultas adicionales para recabar información específica. Por ejemplo, para el procesamiento de una cotización, los datos relativos a la orden de compra (como código de producto, cantidad, etc.) podrían mostrarse solamente si el usuario así lo expresa seleccionando el elemento gráfico que representa la orden. Para conseguir este nivel de prestaciones es

posible que se tengan que recabar datos de otras fuentes posiblemente localizadas en lugares remotos.

4. **Reacción inmediata.** El seguimiento y análisis de la información oportuna y convenientemente presentada hace posible una retroalimentación directa (o indirecta) con el proceso de negocio. Sin embargo, para ello es necesario considerar mecanismos adicionales que permitan tomar el control en el desarrollo de actividades comerciales. En general, los mecanismos de respuesta inmediata pueden ser bastante difíciles de implementar porque requieren, de la máquina de ejecución del proceso de negocio, facilidades para el manejo de contingencias como suspensiones, excepciones y compensaciones.

Para analizar estos problemas discutiremos brevemente las actuales tecnologías de la información relacionadas con ellos.

1.1.1. Recopilación de la información

La tendencia de integrar empresas en los mercados globales ha promovido su reorganización hacia una distribución más eficaz capaz de formar organizaciones virtuales con otras empresas. Dicha tendencia es posible gracias a las actuales tecnologías de Internet, entre las cuales, los servicios Web representan un nuevo paradigma en el desarrollo de sistemas distribuidos de información. Entre los beneficios que ofrecen los servicios Web para la integración de organizaciones destacan el aprovechamiento de la infraestructura de comunicación y de coordinación para la automatización de procesos de negocios. Mediante la automatización se ha conseguido reducir e inclusive eliminar los costos derivados de la contratación de personal, gestión manual de procedimientos administrativos, almacenamiento y distribución de documentos en papel, así como diversos problemas de mensajería y logística asociados con la distribución física de la empresa.

Aunque la automatización mediante servicios Web brinda disponibilidad oportuna de información, la diversidad, cantidad y contenido de la información conllevan problemas difíciles de abordar. Entre los problemas más difíciles se encuentra el de preservar las

relaciones lógicas y temporales que existen entre los datos así como de recuperar e inferir nuevas relaciones. En particular, la determinación de las relaciones tanto causales como temporales entre las actividades resulta del mayor interés porque dichas relaciones le permiten a una organización prestar un servicio siempre que realice correctamente el proceso de negocio correspondiente. La correlación entre los eventos que causan el inicio o la terminación de una o más actividades con un modelo de proceso de negocio es un problema fundamental. La correlación entre los eventos y el modelo permite dar coherencia a la información observada durante la ejecución del proceso. El modelo del proceso permite filtrar la información de acuerdo a sus relaciones lógicas y temporales de modo que asegurar la disponibilidad de la información sea un problema tratable.

1.1.2. Presentación de la información

Debido a la complejidad que revisten la mayoría de los procesos de negocio, el orden en la ejecución de sus actividades debe estar regido por un modelo que permita describir la evolución del comportamiento de sus participantes así como la información intercambiada entre ellos.

En este sentido se requiere de una metodología eficaz que permita especificar, visualizar y documentar el comportamiento de las entidades participantes usando el paradigma de orientación a objetos. En la actualidad, UML (*Unified Modeling Language*) [49] constituye la base de dicha metodología porque sus diagramas permiten capturar y comunicar la información fundamental de los procesos de negocios entre los miembros de una organización. La información refleja el conocimiento que se ha adquirido sobre la conducción de las actividades y de las relaciones entre los datos que intercambian. Los diagramas UML pueden usarse como referencia en todos los niveles de la organización en forma independiente del lenguaje de programación o de la plataforma operativa.

La expresividad y abstracción de los diagramas UML permite atribuir elementos gráficos a una variedad de conceptos fundamentales de sistemas distribuidos como: la creación y destrucción de objetos (posiblemente activos y distribuidos), la invocación de métodos locales, la realización de actividades en forma secuencial o paralela, el envío y recepción de mensajes síncronos y asíncronos y la verificación de condiciones lógicas definidas tanto

sobre el contenido de los mensajes como de los atributos de los objetos.

Aunque UML es un lenguaje concebido para la modelación de sistemas, por las características anteriores, parece también apropiado para identificar y mostrar la información relacionada con el desarrollo de los procesos de negocios. Desde esta perspectiva, los diagramas UML pueden usarse para correlacionar y visualizar la información generada por las instancias de las clases participantes, las actividades que desarrollan, las peticiones de servicio que requieren y las respuestas que obtienen. Esta forma de presentación de la información es sin duda un avance significativo en relación con las descripciones puramente textuales.

1.1.3. Identificación del contexto

Aunque la utilización de los diagramas UML representan una mejora considerable para organizar y presentar la información, se requiere del soporte tecnológico que haga posible recuperar información no mostrada explícitamente en los diagramas. Esta información se caracteriza por el contexto en el que se formula la consulta la cual puede variar en grados de sofisticación que van desde la selección de un enlace de hipermedia hasta la formulación de preguntas usando un lenguaje de consulta de base de datos. Puesto que la integración de múltiples tipos de medios (como texto, diagramas, imágenes, video y sonido) ha demostrado sobradamente su éxito en los sistemas de información basados en la Web, resulta natural utilizar las capacidades de navegación de los sistemas actuales de acceso a la Web como un método eficaz para recuperar información dependiente del contexto.

Aunque el lenguaje original de la Web conocido como HTML (*Hyper Text Markup Language*) [28] ofrece los mecanismos de navegación necesarios, no tiene la capacidad de describir la información estructurada que se usa en las aplicaciones. XML (*eXtensible Markup Language*) [48] es un lenguaje que ha revolucionado y cambiado la Web porque resuelve el problema de la descripción de la información. XML es un lenguaje de marcado al igual que HTML, pero la diferencia entre ambos es que XML permite definir etiquetas propias, lo cual permite al programador crear documentos perfectamente estructurados, facilitando la localización y la recuperación de la información marcada por estas etiquetas.

En comparación, HTML posee un número preestablecido de etiquetas, las cuales sirven solo para describir los detalles de presentación de la información contenida en el documento así como de su vinculación con otros documentos mediante enlaces de hipermedia.

El uso extensivo de XML para la creación de documentos cuyo contenido está orientado a dominios específicos ha derivado en la creación de dialectos de este mismo. Por ejemplo, XMI (*XML Metadata Interchange*) [50] se utiliza para representar diagramas UML utilizando marcado especial que se refiere a los elementos principales de los diagramas. Otro lenguaje derivado de XML es SVG (*Scalable Vector Graphics*) [42], el cual permite crear documentos estructurados con etiquetas que describen elementos gráficos, imágenes y animaciones. En los servicios Web, la adopción de los dialectos de XML, WSDL (*Web Service Description Language*) [41] y SOAP (*Simple Object Access Protocol*) [45], han estandarizado respectivamente la descripción de las interfaces de programación y del formato de los mensajes usados.

Los programas y las aplicaciones que se han desarrollado para XML y sus dialectos ofrecen la posibilidad no solamente de representar la información en forma de diagramas UML sino además ofrecen la posibilidad de recuperar información de contexto utilizando controles simples lo que facilita su presentación adecuada y recuperación mediante navegación.

1.1.4. Reacción inmediata

Las tecnologías de XML y de los servicios Web permiten en buena medida ofrecer información oportuna a los administradores. Al mismo tiempo ofrecen un marco de trabajo basado en interfaces gráficas de usuario en donde se pueden recibir de los administradores las decisiones que han tomado y aplicarlas en el desarrollo del proceso de negocio. En la ejecución de un proceso de negocio basado en servicios Web, las facilidades requeridas que garanticen el cierre del ciclo de control son desafortunadamente difíciles de conseguir. Las dificultades provienen principalmente de la falta de consenso en definir las políticas para el manejo de contingencias que se adoptan en las diferentes empresas. Mientras que para una gran mayoría de organizaciones, la suspensión del proceso y la negación del servicio son las opciones más claras, para muchas otras se deben también tener en cuenta

la compensación de los efectos secundarios que pudieran haber causado la ejecución de las actividades anteriores a la suspensión. Si bien es cierto que este enfoque es el más apropiado en cualquier caso, es cierto también que para su cabal realización se requieren de infraestructura computacional y de logística considerables las cuales no son siempre accesibles para la mayoría de las organizaciones.

1.2. Planteamiento del problema

De acuerdo a lo anterior, se ha puesto de manifiesto la necesidad de recopilar información crítica en forma clara y oportuna para mejorar la toma de decisiones en los procesos de negocios que se realizan mediante servicios Web. Para su solución, este problema se puede descomponer en los siguientes problemas específicos los cuales siguen la línea de la argumentación anterior:

1. **Recopilación de la información.** ¿Cómo se puede recopilar la información en un sistema de intermediación para el comercio electrónico de negocio a negocio?

2. **Presentación de la información.** ¿Cuál es el tipo más conveniente de diagrama UML que permita presentar en forma clara, concisa, detallada e intuitiva la información relativa al desarrollo de las actividades comerciales?

3. **Identificación del contexto.** ¿Cómo se puede presentar la información de contexto del proceso de negocio en ejecución?

4. **Reacción inmediata.** ¿Cómo se puede controlar el curso de la ejecución de un proceso de negocio?

5. **Contribuciones.** ¿Cuáles son los beneficios y las aportaciones al desarrollo de nuevas tecnologías para la conducción de negocios por Internet?

A continuación discutiremos brevemente sobre las soluciones que se proponen a estos problemas.

1.3. Propuesta de solución

La solución que se propone consiste en el diseño y construcción de MS4WS (*Monitoring System for Web Services*, sistema de monitoreo para servicios Web) [37], el cual busca resolver los problemas anteriores como se explica a continuación.

Recopilación de la información. Para recopilar la información de una aplicación de comercio electrónico de negocio a negocio se sugiere utilizar el sistema de intermediación BPIMS-WS (*Business Process Integration and Monitoring System for Web Services*) [33] (otra referencia para BPIMS-WS es [36]) que facilita la integración de empresas mediante el registro, descubrimiento y ejecución de los servicios Web que ofrecen. Por su estructura centralizada, el sistema de intermediación facilita la concentración y posterior distribución de los datos de las aplicaciones para su registro y monitoreo. Con esta base, el sistema de monitoreo se puede identificar como una de las capas en la organización de BPIMS-WS encargada de almacenar las peticiones de los servicios Web que recibe el servidor así como de las respuestas que emite. La publicación o distribución de los datos se hace usando el esquema de petición - respuesta en el que se basa el protocolo HTTP de la Web. Así, las diferentes aplicaciones cliente (como los navegadores usuales de Internet) que soliciten monitorear la información producida durante la ejecución de un proceso de negocio, solamente deben ingresar al sitio Web que brinda el servidor para recibir la información actualizada del estado del proceso y de los mensajes intercambiados.

Presentación de la información. Por sus características, los diagramas UML de secuencia ofrecen la descripción más conveniente (clara, concisa, detallada e intuitiva) en comparación de aquellas que ofrecen otros tipos de diagramas como los diagramas de colaboración, de actividad o de estados. Probablemente la principal razón se encuentra en que los diagramas de secuencia ordenan las actividades que se realizan en forma natural en una línea de tiempo de acuerdo al orden en que ocurren. Su poder descriptivo ha sido reconocido desde que fueron introducidos por Ivar Jacobson como diagramas de interacción de objetos en [53], a pesar de que sus orígenes se pueden ubicar en el modelo Actor de Carl Hewitt que describe las interacciones complejas que tienen lugar en los sistemas distribuidos [38]. Desde entonces los diagramas de secuencia son utilizados extensivamente en el

área de sistemas distribuidos para hacer claras las relaciones temporales que mantienen las interacciones entre las actividades que desarrollan los participantes.

Para las aplicaciones de comercio electrónico se sugiere que la información de cada participante así como los mensajes que intercambian sean desplegadas en los objetos correspondientes que los representan en el diagrama de secuencia UML que describe el proceso de negocio.

Identificación del contexto. Debido a la amplia disponibilidad y a las facilidades de programación que brindan, los navegadores Web son la elección natural para los sistemas cliente de las aplicaciones de comercio electrónico. Puesto que HTML es el lenguaje de marcado nativo para los navegadores Web, la identificación del contexto se realiza mediante enlaces hipermedia que permiten recuperar información adicional almacenada en el servidor BPIMS-WS. Al extender las capacidades gráficas de HTML con SVG, la presentación de diagramas UML se actualiza dinámicamente de acuerdo a la nueva información disponible. El navegador Web ofrece al administrador un escenario que permite explorar información de contexto siguiendo los enlaces de hipermedia ocultos en elementos gráficos como objetos y mensajes. Dicha exploración se caracteriza por preservar la estructura del diagrama de interacción, al mismo tiempo que permite localizar rápidamente los elementos sobre los cuales se puede requerir mayor información.

Reacción inmediata. Como se ha dicho, la capacidad de reacción inmediata es difícil de conseguir debido a que descansa su realización en las facilidades que brinda el sistema de intermediación. En su versión actual, BPIMS-WS no cuenta con mecanismos para el manejo de contingencias por lo que esta característica no es considerada dentro de esta propuesta. Sin embargo, el sistema de monitoreo ofrece el escenario para que en futuras versiones el administrador pueda encontrar allí, los controles adecuados para realizar diversas acciones de control de la aplicación como su suspensión temporal, reanudación, modificación de variables, envío de mensajes o cambio del curso de ejecución de un proceso. También puede extenderse el escenario con una amplia variedad de utilidades como medidores de rendimiento, de calidad, de servicio o de indicadores de contingencias como alarmas o guardianes de seguridad.

1.4. Contribuciones

Las aportaciones de este trabajo al comercio electrónico de negocio a negocio son las siguientes:

1. **Monitoreo de muy alto nivel de servicios Web.** Las interacciones entre los colaboradores de un proceso de negocio son presentadas tomando como base diagramas UML de secuencia. Este nivel de abstracción facilita la comprensión del desarrollo de las actividades sin que el usuario requiera tener conocimientos formales de computación o conocer los detalles de relativos al sistema operativo o a la infraestructura de comunicaciones.

2. **Validación de los procesos de negocio.** El desarrollo del comportamiento de los participantes se puede comparar aquel que ha sido definido como el comportamiento esperado. Los diagramas UML de secuencia facilitan enormemente esta labor ya que el usuario se puede concentrar en reconocer que el diagrama del comportamiento observado corresponde con el esperado, en lugar de tener que estudiar información puramente textual.

3. **Recuperación de información adicional.** El sistema MS4WS emplea diversas tecnologías que hacen posible la recuperación de información adicional teniendo como base el diagrama UML de secuencia. Estas tecnologías hacen posible extender el trabajo en numerosas direcciones incluida la posibilidad de agregar controles en las interfaces de usuario para realizar acciones sobre los procesos monitoreados.

1.5. Organización de la tesis

Esta es la organización de la tesis. El primer capítulo introduce de manera breve el contexto de la tarea de monitorear la ejecución de servicios Web dentro del comercio electrónico (procesos de negocios), y más aún, diferentes puntos de vista relacionados con esta misma tarea. Se introduce a MS4WS como solución a la problemática planteada y la contribución general de esta tesis.

En el segundo capítulo se describen los antecedentes relacionados con esta tesis. Se presenta en más detalle el contexto de monitorear servicios Web, el marco de trabajo que utilizan y descripciones breves de lenguajes que se usan en conjunto con los servicios Web. En este capítulo también se presenta el estado del arte, herramientas relacionadas con este trabajo y sus características.

En el tercer capítulo se describe a MS4WS. Se presenta el contexto en el que opera MS4WS y los participantes con que interactúa. También se describen los módulos que componen a MS4WS y su funcionalidad. Para ejemplificar la funcionalidad de MS4WS se presenta un caso de estudio. Este caso de estudio es el de un agente de compras electrónico basado en servicios Web. En este capítulo se introduce a BPIMS-WS, un sistema de intermediación de comercio electrónico basado en servicios Web. Finalmente se describe el funcionamiento de BPIMS-WS como agente de compras electrónico y el monitoreo de servicios Web con MS4WS.

En el cuarto capítulo se presenta el diseño de MS4WS. Para presentar el diseño de MS4WS se utilizan tres perspectivas. La perspectiva organizacional que muestra el contexto y los participantes con que interactúa MS4WS, así como la relación entre estos. La perspectiva de datos muestra los datos y tipos de datos que se manejan dentro de MS4WS. También se presenta la perspectiva de función que muestra el manejo y procesamiento de los datos y tipos de datos utilizados. Finalmente se presenta la funcionalidad de los módulos de MS4WS utilizando diagramas UML de secuencia.

En el quinto capítulo se presentan las implementaciones de las clases y librerías que se desarrollaron. Se presentan descripciones panorámicas de todas las clases y librerías. También se presentan las relaciones entre clases con diagramas UML de clases. Finalmente se describen algunos detalles que se consideraron durante la implementación de MS4WS.

En el sexto y último capítulo se presentan las conclusiones finales de este trabajo de investigación. Se presentan los resultados y contribuciones de esta tesis. También se menciona el trabajo a futuro que se tiene contemplado.

Capítulo 2

Antecedentes

2.1. Comercio electrónico

La tendencia de integrar empresas en los mercados globales ha permitido identificar nuevas necesidades en el comercio electrónico. La reorganización de las empresas hacia su descentralización así como su asociación temporal para formar organizaciones virtuales ha promovido el desarrollo de las tecnologías de Internet. Las nuevas tecnologías hacen posible la integración de organizaciones en base a los servicios que ellas ofrecen, utilizando la infraestructura de comunicación y de coordinación que caracteriza a la automatización de los procesos de negocios.

La necesidad de recuperar información y de presentarla adecuadamente durante la ejecución de los procesos de negocios cuya base tecnológica son los servicios Web es un problema importante como ya se analizó en el capítulo 1. En este capítulo discutiremos los antecedentes más relevantes a nuestra propuesta en relación a las tres perspectivas principales que se han planteado: monitoreo de servicios Web, representación en diagramas UML de secuencia y visualización con SVG.

2.2. Servicios Web

Los servicios Web surgen con la necesidad de automatizar procesos de negocio entre empresas y aplicaciones. La automatización permitirá desarrollar el potencial de la Web

utilizando lenguajes estándares derivados de XML como SOAP y WSDL. De acuerdo a esta tendencia, los servicios Web son la tecnología base para el desarrollo de nuevas aplicaciones Web.

La adopción de lenguajes y protocolos estándares en los servicios Web garantiza la interoperabilidad entre las aplicaciones. Para ilustrar la conveniencia de garantizar interoperabilidad entre programas podemos mencionar un ejemplo común: Un cliente desea realizar un viaje en avión por lo que acude a pedir información a una aerolínea. El encargado utiliza software para realizar consultas y se da cuenta de que los vuelos que se ofrecen no cubren el viaje completo. Dicha negativa desanima al cliente quien se ve en la necesidad de buscar en otras aerolíneas. A partir de este momento se hace evidente que falta coordinación entre demandas y ofertas: el cliente pierde tiempo en la búsqueda de las aerolíneas que cubran ese viaje, mientras que la aerolínea pierde una venta potencial. Estos problemas se pueden solucionar mediante servicios Web. Dado que los vuelos que ofrece la aerolínea no cubren el viaje completo del cliente, el uso de los servicios Web permitiría comunicar el software de la aerolínea con el software de otras, buscando arreglar el viaje completo. Esta asociación temporal entre organizaciones en base a los servicios que proveen unas a otras conduce a la integración de los procesos de negocios de las aerolíneas. El ejemplo demuestra la utilidad de los servicios Web y demuestra también la importancia que tiene automatizar la búsqueda de organizaciones que ofrezcan determinado tipo de servicios así como realizar la petición del servicio una vez identificado.

Este escenario hace patente la estructura de los servicios Web la cual se divide en tres aspectos fundamentales: publicación, localización (descripción) e invocación del servicio. Para cada uno de estos aspectos se han desarrollado lenguajes basados en XML y herramientas adecuadas para procesarlos. De acuerdo al aspecto que tratan, los lenguajes son:

- **Invocación:** SOAP (*Simple Object Access Protocol*) es el protocolo estándar basado en XML que describe el formato de las peticiones y respuestas de los servicios Web.

- **Localización (descripción):** WSDL (*Web Service Description Language*) es el lenguaje estándar para describir la interfaz de programación de los servicios Web. Un

documento WSDL describe tipos de datos, la estructura de los mensajes, los nombres de las operaciones y toda la información requerida para localizar al proveedor del servicio Web.

- **Publicación:** UDDI (*Universal Description, Discovery and Integration*) es el registro de servicios Web en el cual se pueden encontrar los servicios Web que han publicado las empresas.

XML

Al igual que HTML (*HyperText Markup Language*) [28], XHTML (*Extensible Markup Language*) [30] y DHTML (*Dynamic HyperText Markup Language*) [27], XML [48] es un lenguaje de marcado derivado del SGML (*Standard Generalized Markup Language*) [29]. Sin embargo, entre las características que poseen estos lenguajes con respecto a XML se encuentra el conjunto predeterminado de etiquetas que utilizan. En comparación, XML permite definir y utilizar conjuntos arbitrarios de etiquetas de acuerdo a las necesidades de cada aplicación. Actualmente XML tiene un creciente uso en el intercambio de información a través de Internet y en diversas áreas de computación como son descripción de datos estructurados y semi-estructurados, almacenamiento, graficación, servicios de mensajería, etc. Actualmente el consorcio W3C (*World Wide Web Consortium*) [24] es el encargado de revisar y mantener actualizado este lenguaje de acuerdo a nuevas necesidades y aplicaciones.

En la siguiente sección se presentan algunos de estos dialectos de XML que se utilizan comúnmente en el comercio electrónico.

SOAP

SOAP [45] es un protocolo basado en XML que hereda, por lo tanto, todas sus ventajas. SOAP se utiliza para comunicar aplicaciones con mensajes XML y también para realizar llamadas a procedimiento remoto.

La estructura de un mensaje SOAP es muy simple: posee un elemento XML denominado *sobre* (`<envelope>`), el cual contiene dos elementos, un *encabezado* (`<header>`) y el

cuerpo del mensaje (`<body>`). El contenido del encabezado y el contenido del cuerpo del mensaje son fragmentos de documentos XML válidos y bien formados. En la Figura 2.1 se muestra un ejemplo de un mensaje SOAP.

```
POST /travelservice HTTP/1.0
Host: xxx.xxx.xxx.xxx
Content-Type: text/xml; charset=utf-8
Content-Length: n
SOAPAction: "urn:travelservice-flightinfo"

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
        <SOAP-ENV:Body>
                <m:GetFlightInfo
                xmlns:m="http://www.acme-travel.com/flightinfo">
                SOAP-ENV:encodingStyle=
                "http://schemas.xmlsoap.org/soap/encoding/">
                        <airlineName xsi:type="xsd:string">UL</airlineName>
                        <flightNumber xsi:type="xsd:int">506</flightNumber>
                </m:GetFlightInfo>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 2.1: Petición SOAP

Este mensaje es una petición al procedimiento remoto *GetFlightInfo* posiblemente como servicio. Este procedimiento necesita dos parámetros para su invocación, *airlineName* de tipo `xsd:string` y *flightNumber* de tipo `xsd:int`. Utilizando las herramientas disponibles SAX y DOM para recuperar la información contenida en el mensaje SOAP, las aplicaciones Web son capaces de interpretar este mensaje, de manera que es innecesaria la intervención de un usuario para ejecutar el procedimiento *GetFlightInfo*.

Una respuesta SOAP posee una estructura similar al de una petición. En la Figura 2.2 se muestra un ejemplo de una respuesta, la cual contiene un *sobre* `<envelope>` (no se cuenta con un *encabezado* en este caso) y el *cuerpo* del mensaje `<body>`. Esta respuesta contiene la información solicitada en la petición anterior.

Los mensaje SOAP constituyen una base fundamental para la invocación de los servicios Web. Sin embargo, para que las aplicaciones sean capaces de interpretar los mensajes, los programadores necesitan determinar la estructura y el contenido de los mensajes consultando alguna forma de descripción del servicio.

```
HTTP/1.0 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: n

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
        <SOAP-ENV:Body>
                <m:GetFlightInfoResponse
                xmlns:m="http://www.acme-travel.com/flightinfo"
                SOAP-ENV:encodingStyle=
                "http://schemas.xmlsoap.org/soap/encoding/">
                        <flighInfo>
                                <gate xsi:type="xsd:int">10</gate>
                                <status xsi:type="xsd:string">ON TIME</status>
                        </flightInfo>
                </m:GetFlightInfoResponse>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 2.2: Respuesta SOAP

WSDL

WSDL [41] es un lenguaje para describir servicios Web. Al igual que SOAP, WSDL es un dialecto derivado de XML. Un documento WSDL se divide principalmente en tres secciones. La primera es el *vocabulario* en donde se encuentran las secciones de espacios de nombres y los nombres de los mensajes que se emplean . La segunda corresponde a la *definición* de los mensajes SOAP que se intercambian, la estructura de cada mensaje (elementos y parámetros de cada mensaje) y posiblemente el orden de los mensajes. La tercera y última parte de un documento WSDL corresponde a la *descripción* de la interacción de operaciones del servicio Web. En esta sección se definen las operaciones con su respectivo puerto y, lo esencial de estas operaciones, la descripción de los mensajes SOAP de entrada y de salida que usan las operaciones.

En la Figura 2.3 se presenta un fragmento de un documento WSDL. En primer lugar aparece el espacio de nombres, los prefijos *xsd*, *eticketxsd*, *fligthinfoxsd* y *tns*. Después aparecen las definiciones de los mensajes SOAP, *GetFlightInfoInput*, *GetFligthInfoOutput* y *CheckInInput*, junto cada uno con los elementos que lo compone. Por último tenemos la definición del puerto *AirPortService_PortType*. En este puerto se dan las definiciones de dos operaciones, *GetFlightInfo* y *CheckIn*, la operación *GetFlightInfo* recibe como entrada un mensaje de tipo *tns:GetFlightInfoInput* y regresa como salida un mensaje de tipo

tns:GetFlightInfoOutput, la última operación *CheckIn* que únicamente recibe un mensaje de tipo *tns:CheckInInput* sin regresar ningún mensaje como respuesta.

```
<wsdl:message name="GetFlightInfoInput">
        <part name="airlineName" type="xsd:string"/>
        <part name="flightNumber" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="GetFlightInfoOutput">
        <part name="flightInfo"
type="flightinfoxsd:FlightInfoType"/>
</wsdl:message>
<wsdl:message name="CheckInInput">
<       part name="eTicket" element="eticketxsd:TicketType"/>
</wsdl:message>
<wsdl:portType name="AirportService_PortType">
        <wsdl:operation name="GetFlightInfo">
                <wsdl:input message="tns:GetFlightInfoInput"/>
                <wsdl:output message="tns:GetFlightInfoOutput"/>
        </wsdl:operation>
        <wsdl:operation name="CheckIn">
                <wsdl:input message="tns:CheckInInput"/>
        </wsdl:operation>
</wsdl:portType>
```

Figura 2.3: Fragmento de un documento WSDL. Espacios de nombres, mensajes y puertos.

Además de describir la funcionalidad de un servicio Web, un documento WSDL especifica también el protocolo de transporte que se debe utilizar, cómo se implementa y dónde se localiza el servicio para su invocación. Los elementos `<binding>` describen esta información así como la codificación de los mensajes SOAP. El elemento `<service>` especifica el nombre del servicio Web y la dirección (*URL*) en la que se encuentra el servicio Web. En la Figura 2.4 se presenta un fragmento de un documento WSDL en donde se define la funcionalidad de un servicio Web.

UDDI

UDDI [40] es un sistema de directorio distribuido que se utiliza básicamente para mantener las descripciones de servicios Web (documentos WSDL) que hayan sido previamente publicados. UDDI ofrece un servicio de directorio que permite a clientes de compañías descubrir (buscar y localizar) servicios Web. Para esto, UDDI sigue las siguientes especificaciones:

- Contiene la descripción e información acerca de varios servicios Web, por ejemplo, cómo se puede utilizar un servicio Web.

```
<wsdl:binding name="AirportService_SoapBinding"
type="tns:AirportService_PortType">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
                <wsdl:operation name="GetFlightInfo">
                        <soap:operation style="rpc"
                        soapAction="urn:travelservice-flightinfo"/>
                        <wsdl:input>
                                <soap:body encodingStyle=
                                "http://schemas.xmlsoap.org/soap/encoding/"
                                namespace="urn:travelservice-flightinfo"
                                use="encoded"/>
                        </wsdl:input>
                        <wsdl:output>
                                <soap:body encodingStyle=
                                "http://schemas.xmlsoap.org/soap/encoding/"
                                namespace="urn:travelservice-flightinfo"
                                use="encoded"/>
                        </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="CheckIn">
                        <soap:operation style="document"
                        soapAction="urn:travelservice-checkin"/>
                                <wsdl:input>
                                        <soap:body encodingStyle=
                                        "http://schemas.xmlsoap.org/soap/encoding/"
                                        namespace="urn:travelservice-checkin"
                                        use="encoded"/>
                                </wsdl:input>
                </wsdl:operation>
        </soap:binding>
</wsdl:binding>
<wsdl:service name="travelservice">
        <wsdl:port name="travelservice_Port"
        binding="tns:AirportService_SoapBinding">
                <soap:address location=
                "http://www.acme-travel.com/travelservice"/>
        </wsdl:port>
</wsdl:service>
```

Figura 2.4: Fragmento de un documento WSDL. Funcionalidad de un servicio Web

- Permite consultar, localizar o publicar descripciones de servicios Web. También cuenta con APIs que permiten a los programadores implementar búsquedas en UDDI en las aplicaciones. Para los clientes de servicios Web, UDDI se puede acceder también mediante portales de Internet.

Además de seguir estas especificaciones, UDDI mantiene cierta estructura de categorización de servicios Web, esta estructura se divide en tres secciones:

- **Páginas blancas.** Incluyen información como el nombre y los contactos de la compañía en la que se ofrece el servicio Web.

- **Páginas amarillas.** Contienen información acerca de la categoría en la que se encuentra el servicio en base a su tipo y a la compañía proveedora del servicio.

- **Páginas verdes.** Contienen información técnica más especializada acerca del servicio.

Actualmente, UDDI es el único registro para hacer públicos servicios Web, la mayoría de servicios Web disponibles en la Web se encuentran publicados en UDDI.

Resumiendo el escenario de los servicios Web tenemos a SOAP como protocolo de comunicación, documentos WSDL para describir servicios Web y a UDDI como registro distribuido para publicar, buscar y localizar servicios Web. En la siguiente sección se discuten los problemas relacionados con el monitoreo de servicios Web, así como los lenguajes y las tecnologías que se utilizaron en este trabajo de investigación.

2.3. Monitoreo de servicios Web

La tarea de monitorear un servicio Web, es decir, visualizar la ejecución de un servicio Web ofrece prestaciones de muy alto nivel y abstracción para la toma de decisiones. Si un servicio Web entrega resultados erróneos o incorrectos, para diseñadores y programadores depurar este servicio puede tomarles un tiempo considerable. Históricamente, el monitoreo de sistemas (no únicamente servicios Web) ha evolucionado hacia tecnologías abiertas y distribuidas. En las siguientes secciones se describen aquellas que se consideran más importantes por su impacto.

2.3.1. UML

Los lenguajes de modelado Booch [39], OMT [52] y OOSE [51], por mencionar algunos, fueron creados también con la finalidad de modelar arquitecturas y estructuras de procesos y/o sistemas de software. La existencia de varios lenguajes de modelado originó problemas de comunicación entre los equipos de programación, ya que utilizaban frecuentemente diagramas de diferentes lenguajes para el desarrollo de un mismo sistema. Una de las metas de la creación de UML [49] fue unificar estos lenguajes, estableciéndose así UML como lenguaje estándar de modelado.

Los precursores de la creación de UML son Grady Booch, Jim Rumbaugh e Ivar Jacobson. El desarrollo de UML empezó en el año de 1994, cuando Grady, Jim e Ivar trabajaban para la corporación de Rational Software [18] y comenzaron a trabajar en la unificación de los actuales lenguajes de modelado. La primera versión oficial de UML fue presentada en 1995 como la versión 0.8.

Las metas principales que se tomaron en cuenta en el diseño de UML son:

1. Proveer un lenguaje de modelado visual que facilite y acelere el entendimiento de lo que se está modelando.

2. Proveer mecanismos de especialización que describan conceptos importantes en detalle.

3. Ser independiente del lenguaje de programación y de los tipos de procesos.

4. Proveer una base simple de entendimiento del lenguaje visual.

5. Diseminar el paradigma de Programación Orientada a Objetos.

6. Dar soporte y alentar conceptos de programación en grupos de trabajo como comunicación, colaboración y marcos de trabajo, por mencionar solo algunos.

UML permite modelar, especificar y visualizar sistemas de software, organizando metódicamente las actividades de programación y minimizando con ello el tiempo de desarrollo.

UML es una colección de doce tipos de diagramas, divididos en 3 categorías: 4 diagramas para representar la estructura de procesos (diagramas de clases, de objetos, de componentes y diagramas de puesta en marcha), 5 para describir aspectos de comportamiento (diagramas de casos de uso, de secuencia, de actividades, de colaboración y diagramas de estados) y 3 para describir detalles de implementación (diagramas de paquetes, de subsistemas y diagramas de modelos).

El uso de UML para desarrollar o documentar sistemas de software magnifica el valor de los resultados finales, ya que facilitan el entendimiento y mejoran la rapidez para realizar modificaciones en determinado aspecto, por ejemplo, en actualizaciones del sistema. Resulta evidente que no es lo mismo revisar sistemas de software con diagramas (acceder

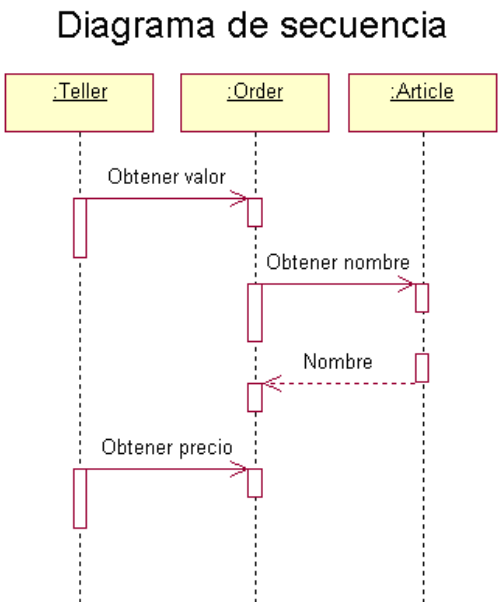


Figura 2.5: Diagrama de secuencia

a la información en un espacio bidimensional) a revisar líneas de código (acceder a la información en un espacio unidimensional). Lo anterior se traduce también en el ahorro de tiempo y de los costos de actualización del sistema.

Servicios Web y diagramas UML de secuencia

Los diagramas UML de secuencia son particularmente adecuados para modelar el comportamiento de procesos que involucran la concurrencia y sincronía entre eventos y actividades, presentando la secuencia temporal en que estos ocurren. En la Figura 2.5 se muestra un ejemplo sencillo de un diagrama de secuencia.

Este diagrama presenta la interacción de los participantes *Teller*, *Order* y *Article*. Los participantes se representan con rectángulos etiquetados con los nombres de las instancias y de las clases a las que pertenecen. La secuencia de mensajes enviados y recibidos por los participantes se muestra de arriba hacia abajo a lo largo de las líneas de tiempo representadas por líneas punteadas. En cada línea de tiempo se desarrolla el comportamiento individual de cada participante. Los recuadros pequeños y alargados que se encuentran encima de las líneas de tiempo de algún participante representan la realización de alguna actividad.

Entre las principales ventajas que ofrecen los diagramas de secuencia se encuentra por ejemplo, ofrecer documentación gráfica de un proceso de negocio lo que permite aclarar dudas sobre su diseño o implementación.

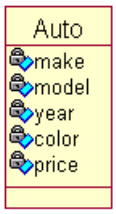
El uso de UML se ha extendido gracias a la amplia disponibilidad de herramientas, librerías de programación y entornos de modelado, los cuales hacen uso de algún dialecto de XML. Por ejemplo, XMI permite serializar diagramas UML en documentos estructurados con etiquetas XML y SVG permite crear elementos gráficos a partir de etiquetas predefinidas que se pueden visualizar en navegadores de Internet. Con estos elementos gráficos se pueden formar diagramas UML de secuencia lo que brinda un punto de partida para realizar una herramienta que permita monitorear la información de los procesos de negocio así como el comportamiento de sus participantes. En las siguientes secciones se describen estos lenguajes.

XMI

XMI [50] fue propuesto por la OMG en Junio de 1998. XMI surge de la necesidad que tienen empresas como IBM, Unisys y otras para poder definir, validar e intercambiar documentos escritos en XML que contienen descripciones de procesos modelados en UML.

Así el objetivo principal de XMI es compartir modelos UML de sistemas. XMI describe modelos UML en documentos estructurados los cuales se pueden transmitir a través de Internet, almacenar en bases de datos o repositorios de diagramas UML. Lo anterior se debe a que XMI remplaza la limitada manipulación de imágenes de los diagramas UML por el procesamiento de la estructura y del contenido de los documentos XML utilizando la amplia variedad de herramientas disponibles. En la Figura 2.6 se muestra un ejemplo sencillo de una clase y el documento XMI correspondiente a esta clase.

El recuadro de arriba a la izquierda representa en forma gráfica a la clase *Auto*. Arriba a la derecha se muestra una instancia de la misma clase *Auto* pero descrita en un documento XML, el cual se genera a partir del documento XMI DTD que aparece debajo de ellos. Aunque el modelo de la clase *Auto* se puede almacenar de distintas maneras utilizando por ejemplo diferentes formatos de imágenes (BMP, JPEG, etc.), la manera más eficaz de procesar la información allí contenida es usando algún lenguaje de descripción como




```
<Auto>
     <Make>   Ford          </Make>
     <Model>  Mustang  </Model>
     <Year>   99       </Year>
     <Color>  blue          </Color>
     <Price>  25000    </Price>
</Auto>
```

```
<!ELEMENT Auto (Auto.Make, Auto.Model, Auto.Year,
                Auto.Color, Auto.Price,
                XMI.extension*)? >
<!ATTLIST Auto %XMI.element.att; %XMI.link.att;>

<!ELEMENT Auto.Make  (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Model (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Year  (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Color (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Price (#PCDATA | XMI.reference)* >
```

Figura 2.6: Escenario de XMI

el formato MDL de Rational Rose [18]. La representación de datos como imágenes no permite recuperar la definición de la clase, además de que ésta representación ocupa cantidades considerables de espacio. Por otro lado, la representación de los modelos UML como documentos XMI permite recuperar y manipular la definición de la clase. Además la representación en documentos XMI es más económica ya que ocupan una cantidad considerablemente menor de espacio que aquella que usan los archivos de imágenes. Por lo anterior, XMI es un lenguaje estándar ampliamente utilizado para describir modelos UML en forma textual.

La gran ventaja de XMI es su estructura basada también en etiquetas XML. Estas etiquetas describen y almacenan la información contenida en los modelos UML, lo cual permite automatizar la lectura de la información por otras aplicaciones también de modelado con UML. En la Figura 2.7 se muestra la utilidad de XMI. En esta Figura se muestran dos aplicaciones que pueden intercambiar un mismo diagrama UML descrito en XMI.

Un enfoque alternativo a utilizar imágenes de los diagramas UML es utilizando un dialecto de XML que describa elementos gráficos (líneas, círculos, rectángulos, texto, color, etc.) para usarlos en la descripción de los diagramas. De esta forma, las herramientas de

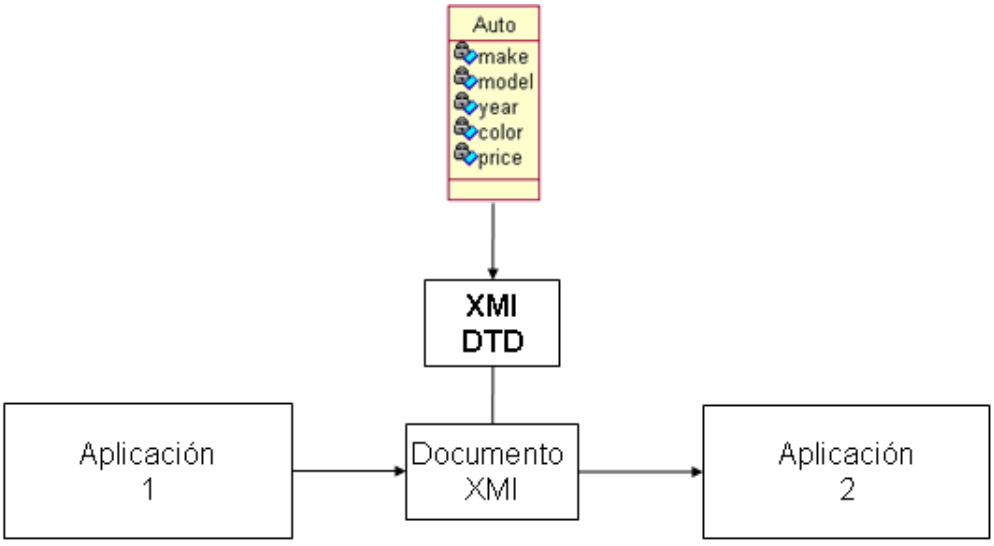
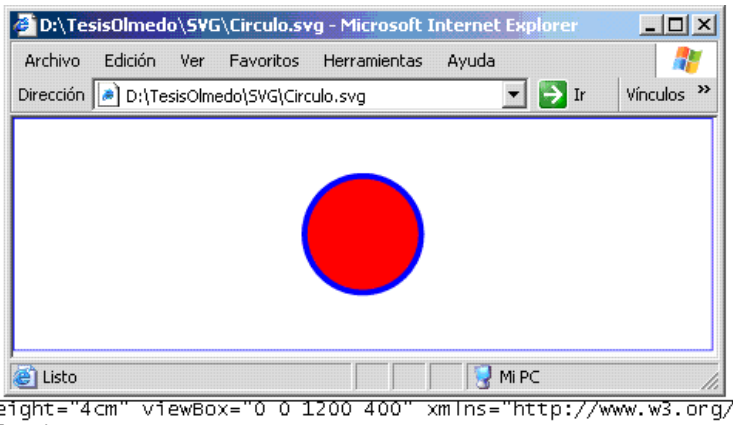


Figura 2.7: Documento XMI compartido por dos aplicaciones

visualización se encargan de interpretar los elementos gráficos y de graficar cada uno de ellos para realizar la composición del diagrama. A este tipo de lenguajes de graficación basados en XML pertenece SVG, el cual se describe a continuación.

2.3.2. Visualización de diagramas UML con SVG

SVG [42] es un dialecto de XML de graficación vectorial escalable creado por el consorcio de la Web (W3C) [24]. SVG difiere en la forma con que se representa información de graficación en formatos como JPEG y GIF que contienen descripciones de las imágenes enteras en mapas de píxeles. En comparación, las imágenes SVG se grafican en base a vectores lo cual permite proporcionar únicamente información esencial para los métodos de graficación (líneas, círculos, pintado de regiones, etc.) predefinidos en librerías (o *plug-ins*) que se instalan en los visualizadores. Aunque SVG tiene varias ventajas con respecto a otros formatos, su principal desventaja es que no todos los navegadores visualizan documentos SVG. Para remediar esta deficiencia, actualmente existen disponibles una gran variedad de *plug-ins* tales como el ASV (*Adobe SVG Viewer*) de Adobe [7], los cuales permiten visualizar las descripciones SVG en las ventanas de los navegadores de Internet. En la Figura 2.8 se muestra un ejemplo sencillo de un documento SVG visualizado en el navegador IExplorer con el ASV.




```
<svg width="12cm" height="4cm" viewBox="0 0 1200 400" xmlns="http://www.w3.org/2000/svg">
        <!-- Circulo -->
        <rect x="1" y="1" width="1198" height="398" fill="none" stroke="blue" stroke-width="2"/>
        <circle cx="600" cy="200" r="100" fill="red" stroke="blue" stroke-width="10" />
        <!-- Fin Cirulo -->
</svg>
```

Figura 2.8: Documento SVG visualizado en IE con ASV

Hasta este punto se ha presentado el escenario de los servicios Web así como la problemática actual de monitorear estos servicios y algunas tecnologías disponibles. A continuación se presenta el estado del arte dentro de este contexto, en el cual se presentan las posibles soluciones que actualmente se pueden encontrar para resolver los problemas relacionados con el monitoreo de servicios Web.

2.4. Revisión del estado del arte

Dentro de las soluciones que se encuentran dentro del contexto de servicios Web con UML, actualmente no se cuenta con herramientas que permitan realizar, en forma conjunta, la tarea de monitorear la ejecución de servicios Web con diagramas UML, únicamente se encuentran soluciones que realizan parte de estas tareas.

Esta sección se divide en dos partes, la primera presenta las descripciones de soluciones enfocadas en monitorear servicios Web y la segunda presenta las descripciones de soluciones enfocadas a modelar con UML.

2.4.1. Herramientas de monitoreo de servicios Web

Hoy en día se pueden encontrar varias herramientas de monitoreo de servicios Web y otras de monitoreo de protocolos de transporte basados en XML como SOAP. Cada una de

estas herramientas presenta de forma distinta los resultados de monitorear servicios Web. Algunas presentan resultados estadísticos, otras presentan en modo textual los mensajes SOAP, mientras que muy pocas presentan un monitoreo de forma gráfica.

A continuación se presentan de manera breve las especificaciones, capacidades y algunos resultados de monitorear servicios Web de estas herramientas.

AmberPoint Management Foundation

AmberPont Management Foundation (AMF) [1] de Roge Wave Software es una herramienta creada principalmente para manejar, monitorear y para mejorar la funcionalidad de servicios Web creados con LEIF (*Lightweight Enterpise Integration Framework*) [2] el cual es un marco de trabajo especializado para servicios Web. AMF permite mejorar estos servicios sin la necesidad de modificar o añadir código. Monitorear estos servicios incrementa la calidad y asegura la funcionalidad de estos.

A continuación se listan las capacidades y características que AMF ofrece para manejar y mejorar servicios LEIF:

- Administración de servicios Web.
- Monitoreo de ejecución de servicios Web.
- Alertas y notificaciones.
- Registro de servicios Web.
- Modificación de mensajes petición - respuesta.
- Seguridad.
- Enrutamiento de servicios Web.

La Figura 2.9 muestra los resultados de AMF de monitorear servicios Web. Estos resultados presentan en forma gráfica las estadísticas de los mensajes enviados y recibidos en la ejecución. Por ejemplo, se presentan gráficas que presentan relaciones del número de

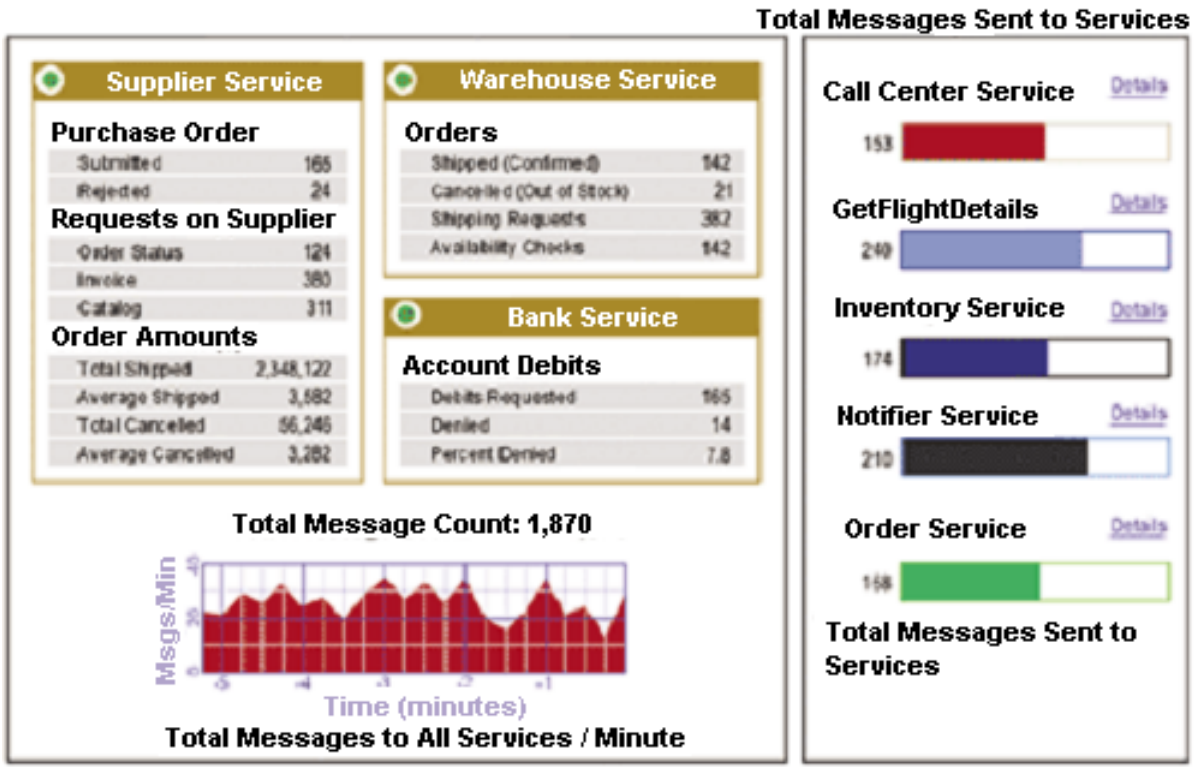


Figura 2.9: Resultados del monitoreo de servicios Web con AMF

mensajes enviados a determinados tiempos, así también se presenta el número de mensajes enviados y recibidos por cada servicio.

AMF es una herramienta de gran utilidad. Sin embargo, el monitoreo que AMF realiza es únicamente estadístico y orientado específicamente a resolver problemas de tráfico de redes, por lo que los resultados de la interacción entre servicios Web que ofrece este monitoreo son muy limitados. AMF no utiliza ningún método de diagramación para mostrar de manera más entendible la ejecución de servicios Web.

Cape Clear 5

Cape Clear 5 de Cape Clear [3] es un conjunto de herramientas diseñadas para la creación, diseño, puesta en marcha y prueba de servicios Web, facilitando la integración de aplicaciones Web de clientes y de compañías. Cape Clear 5 está conformado por Studio, Server, Data Interchange y Manager. Cape Clear Studio es un entorno gráfico para diseñar, probar y poner en marcha servicios Web, en tanto que Cape Clear Server es una

plataforma para servicios Web. Esta plataforma se utiliza para poner en marcha y llevar a cabo la integración de servicios Web. Cape Clear Data Interchange es una extensión de Cape Clear Server. Por último, Cape Clear Manager permite definir reglas y/o alarmas para poder administrar fácil y consistentemente servicios Web.

Cape Clear 5 permite monitorear servicios Web de dos maneras. Los componentes incluidos para monitorear servicios Web son:

1. Web Service Tester.

2. NetTool.

Web Service Tester permite monitorear los servicios Web en ejecución especificando las direcciones (*URL*) en donde se encuentran los documentos WSDL. Permite generar peticiones de prueba y presenta los mensajes SOAP de respuesta. Además, también permite monitorear servicios Web a partir de esquemas XML. Aunque el uso de estos documentos no es muy común, permite a diseñadores y programadores visualizar los mensajes SOAP tanto de peticiónes como de respuestas. En la Figura 2.10 se muestra un ejemplo de como Web Service Tester visualiza mensajes SOAP.

NetTool es un componente Java (conjunto de clases Java, archivos `.jar`) que permite interceptar y enviar mensajes SOAP a través de un túnel por el cual transitan mensajes SOAP. NetTool visualiza las peticiones y respuestas intercambiadas por un Cliente y un servicio Web, permitiendo examinar el contenido de éstos mensajes. Aunque NetTool permite realizar casi las mismas tareas que Web Service Tester, NetTool no permite trabajar con esquemas XML. En la Figura 2.11 se muestra la interfaz de NetTool. Arriba de la Figura 2.11 se muestran mensajes SOAP interceptados, mientras que abajo se muestra el envío de una petición SOAP y su respuesta.

Cape Clear 5 es una herramienta muy completa. Permite realizar tareas de gran utilidad. Sin embargo, los resultados de monitorear servicios Web con los componentes Web Service Tester o NetTool únicamente son útiles para especialistas o programadores de servicios Web. Aunque ésta puede ser su principal finalidad, para los clientes de servicios Web esta herramienta no les resulta de gran utilidad ya que no son expertos, ya es difícil interpretar

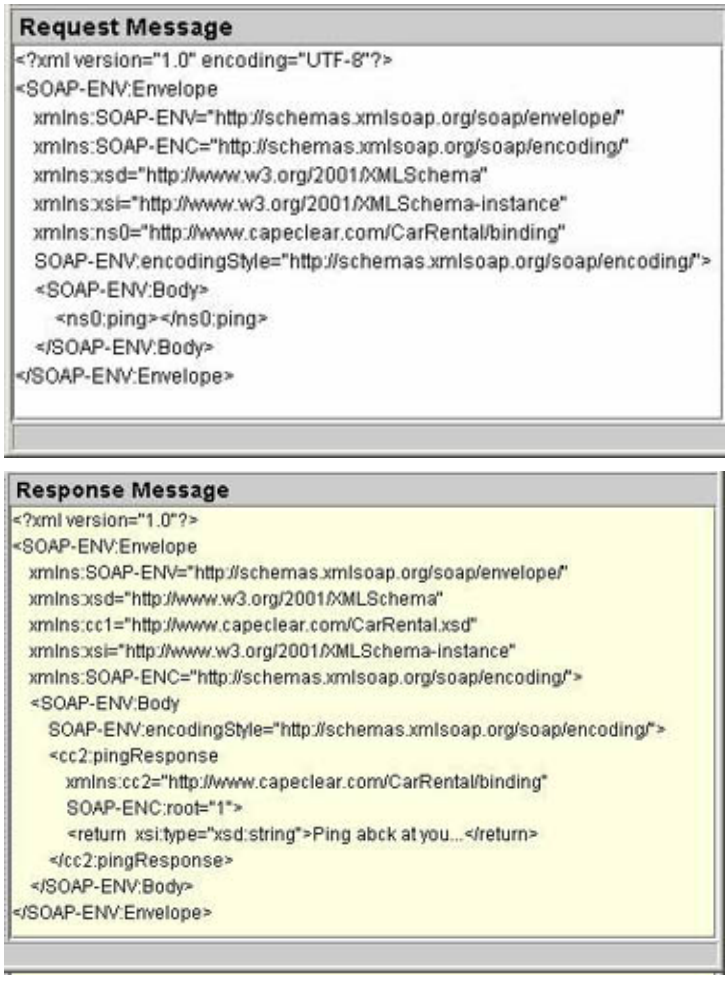


Figura 2.10: Presentación de mensajes SOAP con Web Service Tester

los resultados que se obtienen con Cape Clear 5. Cape Clear 5 no utiliza ningún lenguaje de diagramación que permita a los usuarios entender la ejecución de los servicios Web.

BPEL Process Manager

BPEL Process Manager de Oracle [4] es una herramienta enfocada principalmente a la composición de procesos BPEL (*Business Process Execution Language*) [46]. Estos procesos son composiciones de servicios Web, es decir, un proceso BPEL hace uso de varios servicios Web. Esta herramienta permite crear, monitorear, modificar parámetros de invocación y depurar estos procesos. La tarea de monitorear estos procesos se realiza utilizando diagramas, los cuales, aunque no utilizan un lenguaje de diagramación conocido, son bastante comprensibles.

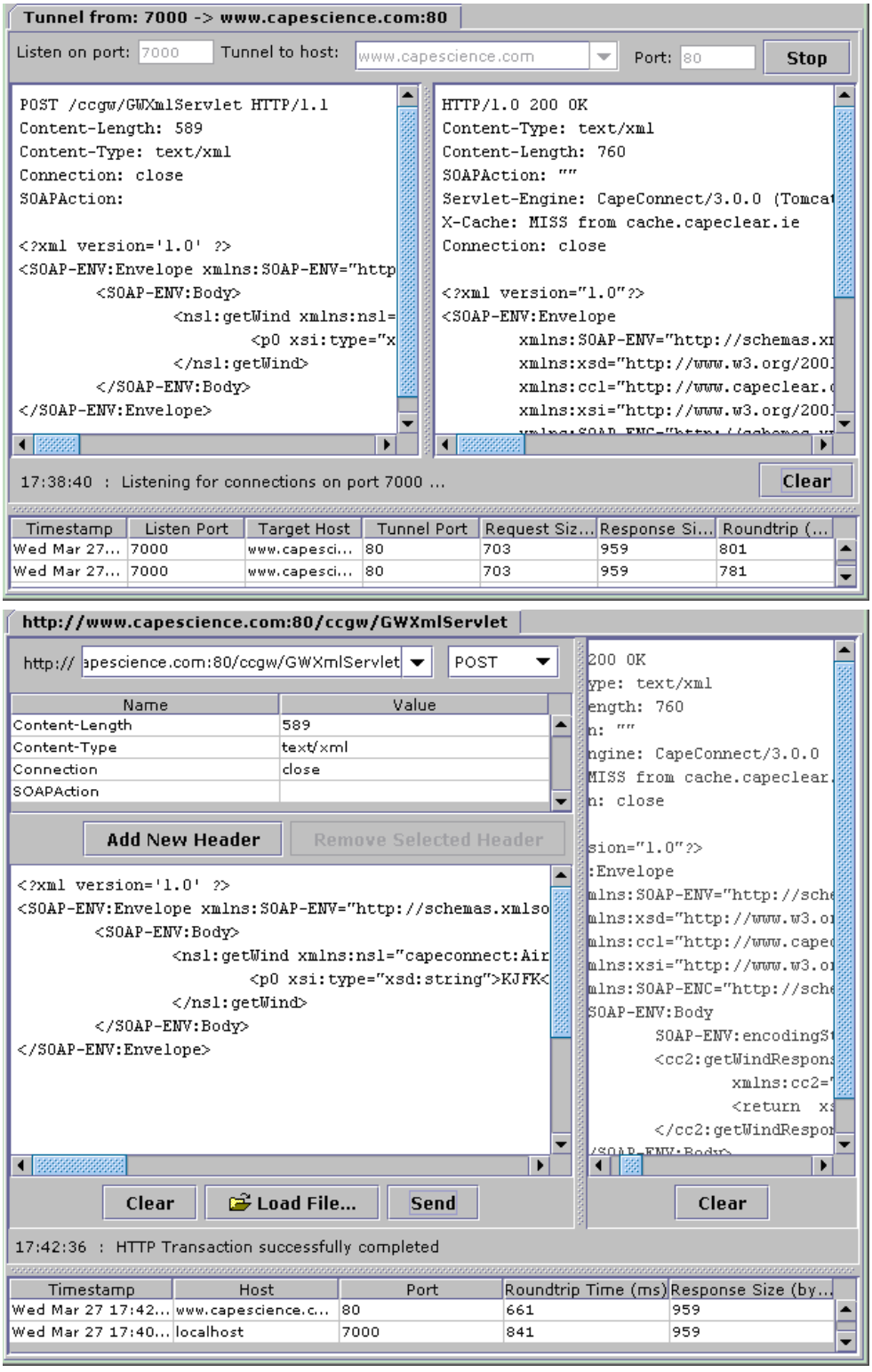


Figura 2.11: Mensajes SOAP y envío de una petición SOAP con NetTool

BPEL Process Manager permite invocar o crear instancias de estos procesos. El flujo de trabajo de estas instancias se visualiza con diagramas lo cual permite monitorear los procesos. Además de monitorear el flujo de trabajo de estos procesos se pueden verificar y modificar las invocaciones realizadas a nivel código, lo cual permite depurar la ejecución de estos procesos.

En la Figura 2.12 se presenta un ejemplo de un proceso BPEL llamado *LPriceAndQuantity* con BPEL Process Manager.

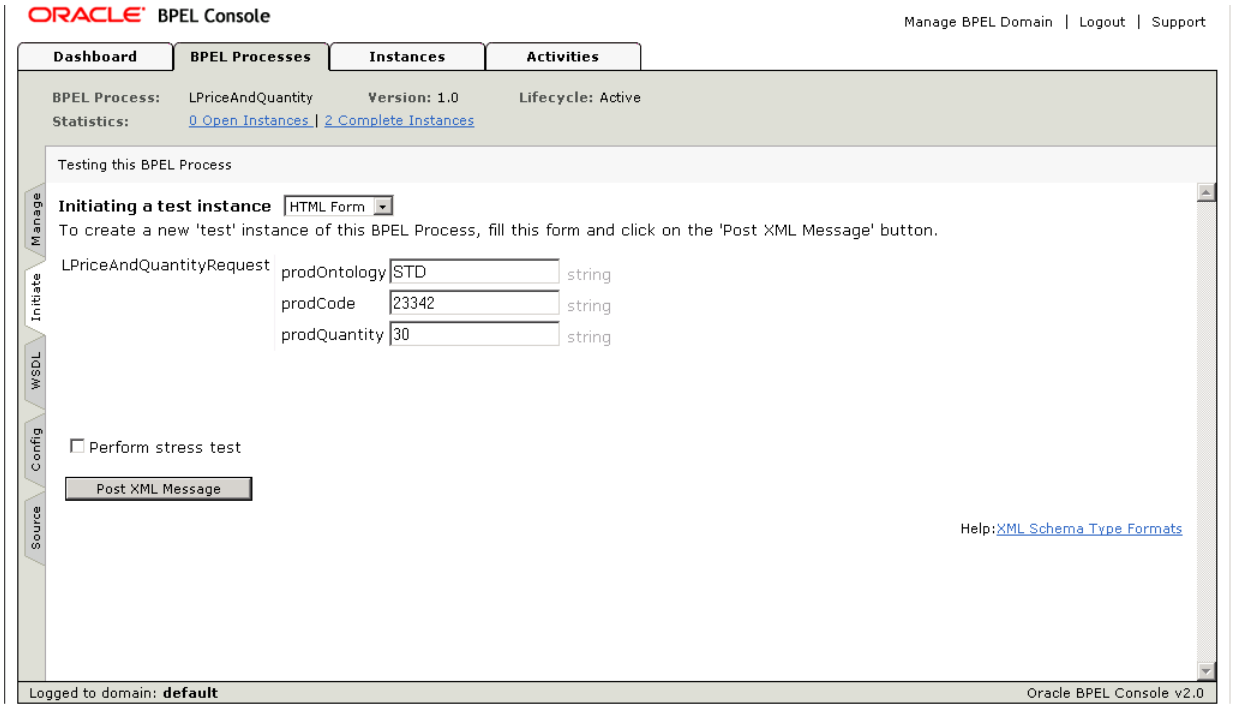


Figura 2.12: Parámetros de invocación del proceso *LPriceAndQuantity*

En la Figura 2.12 se presenta la forma que contiene los parámetros de invocación del proceso *LoanProcurement*. Después de llenarse la forma se puede ejecutar el proceso BPEL.

Arriba de la Figura 2.13 se muestra la instancia del proceso *LPriceAndQuantity*, en la cual se puede visualizar el flujo de trabajo del proceso BPEL, auditar sub-invocaciones realizadas durante el flujo de trabajo o depurar este flujo, es decir, podemos visualizar y/o modificar los parámetros de cada invocación realizada. Abajo de la misma Figura se muestra el diagrama de flujo generado por BPEL Process Manager, este diagrama presenta gráficamente el flujo de trabajo que realizó el proceso *LPriceAndQuantity*.

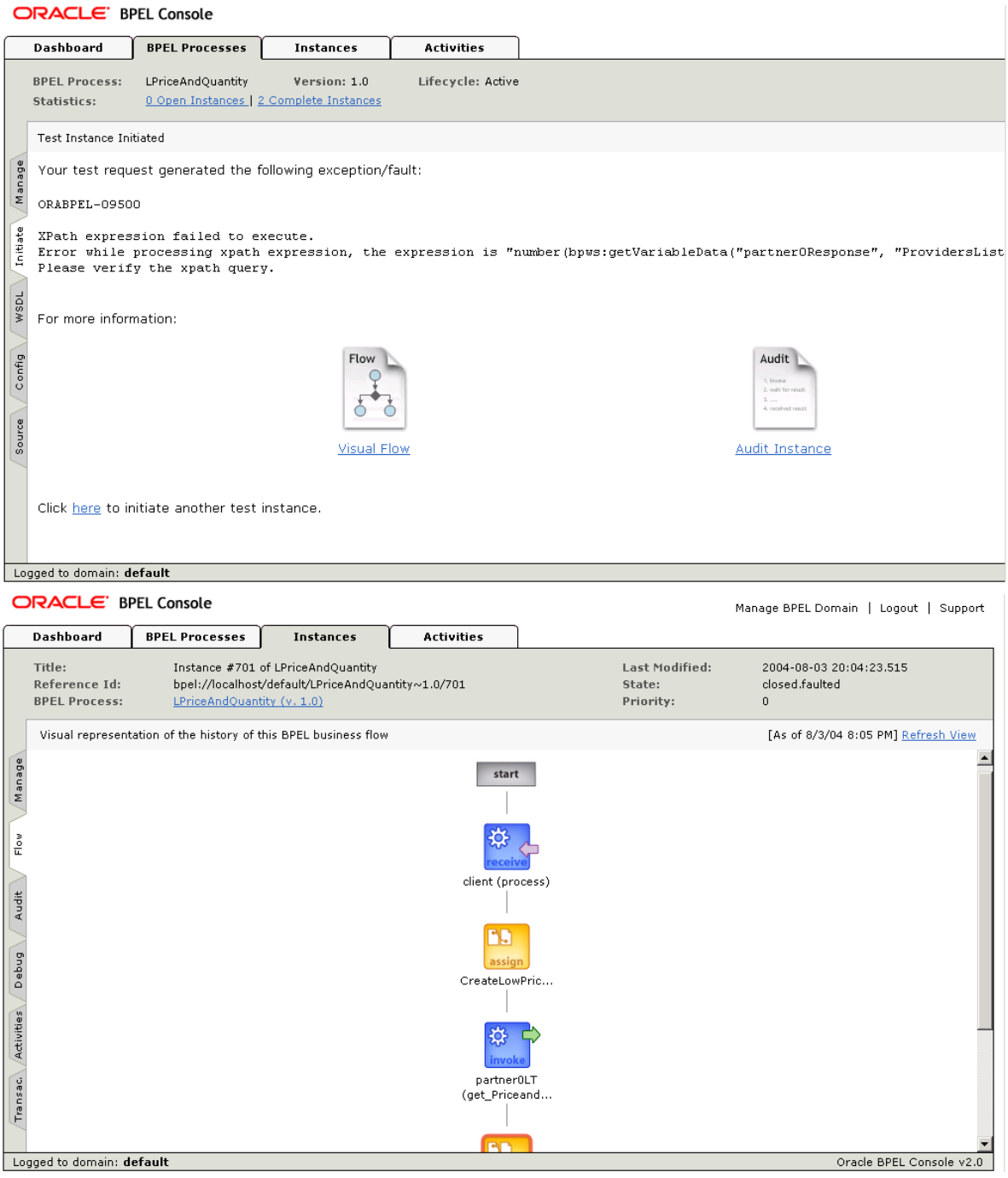


Figura 2.13: Instancia y diagrama del flujo del proceso *LPriceAndQuantity*

BPEL Process Management es una herramienta muy útil porque permite diseñar, poner en marcha, monitorear, probar y depurar procesos BPEL. La tarea de monitorear procesos utilizando diagramas es muy buena, sin embargo, el lenguaje de diagramación que se utiliza, aunque es entendible, sigue un enfoque muy especializado. Los diagramas que se

generan pueden ser entendibles únicamente por expertos en procesos BPEL y muy pocos clientes de estos procesos podrían entender el desempeño que han tenido los procesos que han utilizado. La diagramación empleada no sigue un lenguaje estándar, lo cual dificulta el entendimiento de estos diagramas por no contar con la documentación correspondiente.

Infravio X-Broker

Infravio X-Broker de Infravio [5] es una herramienta que forma parte de Infravio Ensemble. Esta herramienta es un intermediario entre servicios Web, entre aplicaciones clientes y proveedores. Realiza tareas como seguridad, enrutamiento y monitoreo de servicios Web en tiempo de ejecución establecidos en un contrato (documento escrito en WSDC, *Web Services Delivery Contract*) [6] definido también por Infravio. Infravio X-Broker soporta estándares definidos para servicios Web como XML y SOAP además de poderse utilizar con IBM WebSphere y BEA WebLogic. La disponibilidad de esta herramienta es muy restringida ya que no se cuenta con versiones de evaluación disponibles en Internet, por esta razón no se presentan vistas de esta herramienta.
Infravio X-Broker cuenta con características de gran utilidad como monitorear servicios Web en tiempo de ejecución y establecer alarmas para indicar el desempeño fallido de una operación de gran utilidad. Además, permite manipular la ejecución de un servicio Web en tiempo de ejecución. Sin embargo, los resultados de monitorear servicios Web son difíciles de comprender debido a que esta herramienta es útil únicamente para programadores. El uso de documentos WSDC para monitorear servicios Web limita de manera importante esta herramienta ya que WSDC no es un estándar de servicios Web. Infravio X-Broker no utiliza ningún tipo de diagramación que permita visualizar la ejecución de servicios Web lo cual ayudaría bastante a interpretar el comportamiento de los servicios Web.

WebServiceTester

WebServiceTester de Optimiz [25] es una herramienta muy útil para programadores y diseñadores de servicios Web. WebServiceTester permite monitorear servicios Web además de asistir en tareas de depuración. Permite realizar peticiones a servicios Web y verificar sus resultados. Estas tareas se pueden aplicar en servicios Web públicos, los cua-

les se pueden invocar proporcionando la *URL* del documento WSDL del servicio Web. WebServiceTester también permite orquestar servicios Web.
Desafortunadamente no se tienen versiones de evaluación disponibles en Internet, razón por la cual no se presentan vistas del funcionamiento de WebServiceTester. Sin embargo, dentro de la especificación de esta herramienta no se contempla diagramación en el monitoreo de servicios Web, lo cual dificulta en gran medida la interpretación de la ejecución de servicios Web para usuarios no expertos.

SOAPtest

SOAPtest de Parasoft [16] permite verificar varios aspectos de funcionamiento y desempeño de servicios Web. Además, SOAPtest es una herramienta multiplataforma que permite trabajar sobre protocolos como HTTP, HTTPS y TCP/IP. SOAPtest es de gran utilidad para diseñadores y programadores porque permite:

1. Generar pruebas a partir de WSDLs.
2. Generar pruebas de servicios Web asíncronos.
3. Monitorear tráfico de protocolos.
4. Generar reportes detallados de desempeño.
5. Preparar gráficas y tablas en tiempo de ejecución.
6. Revertir modificaciones no deseadas.
7. Realizar pruebas tanto en el lado del Servidor como en el Cliente.
8. Utilizarse como *proxy* o como intermediario de servicios Web con Clientes.

En las Figuras 2.14 y 2.15 SOAPtest se presentan vistas de SOAPtest. En la Figura 2.14 se muestran mensajes SOAP y una gráfica del tráfico de mensajes SOAP que transitan la red. En la Figura 2.15 se presenta una lista de las pruebas realizadas a servicios Web. Estos reportes se guardan en disco duro para una futura comparación con otros reportes.

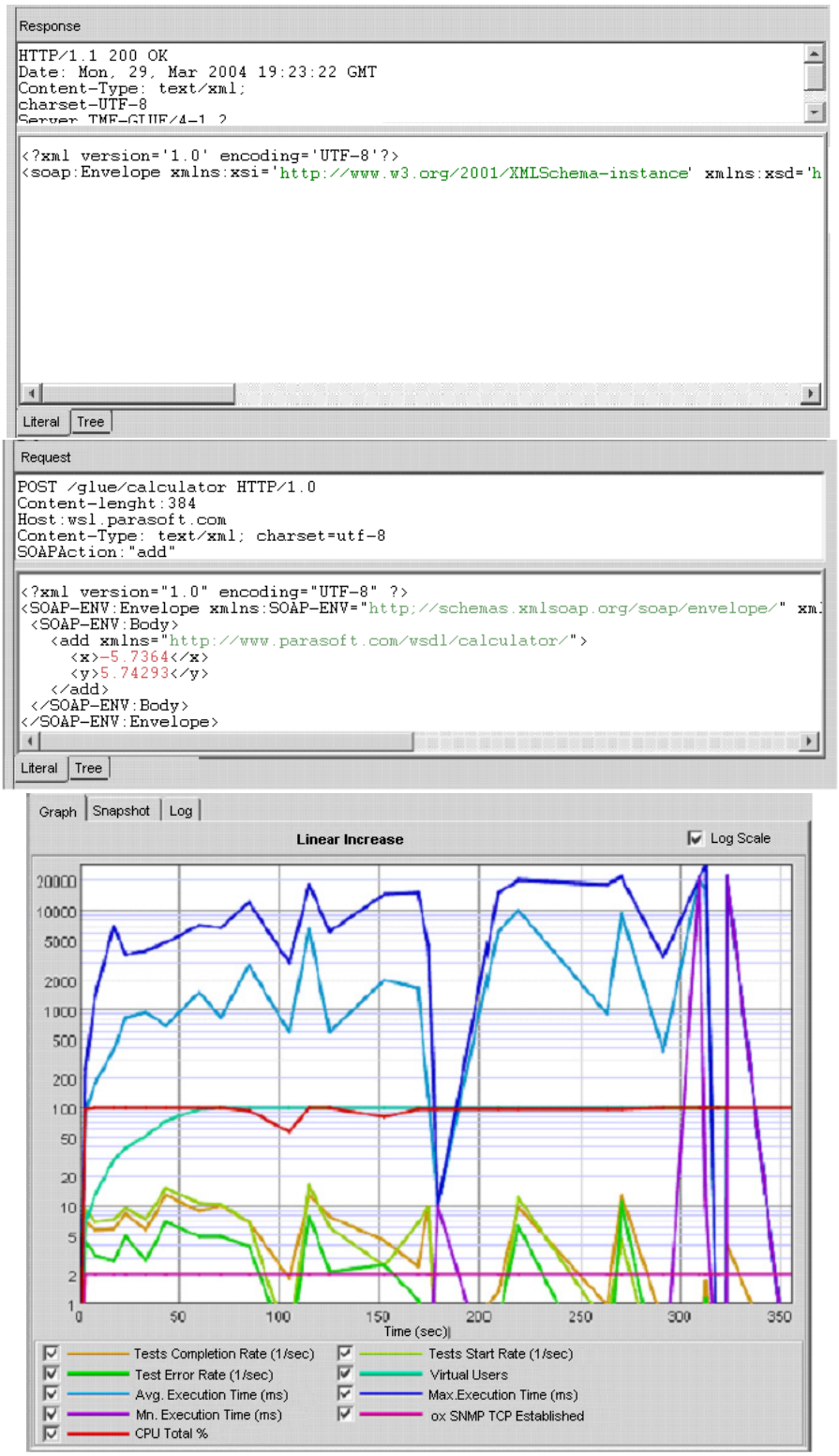
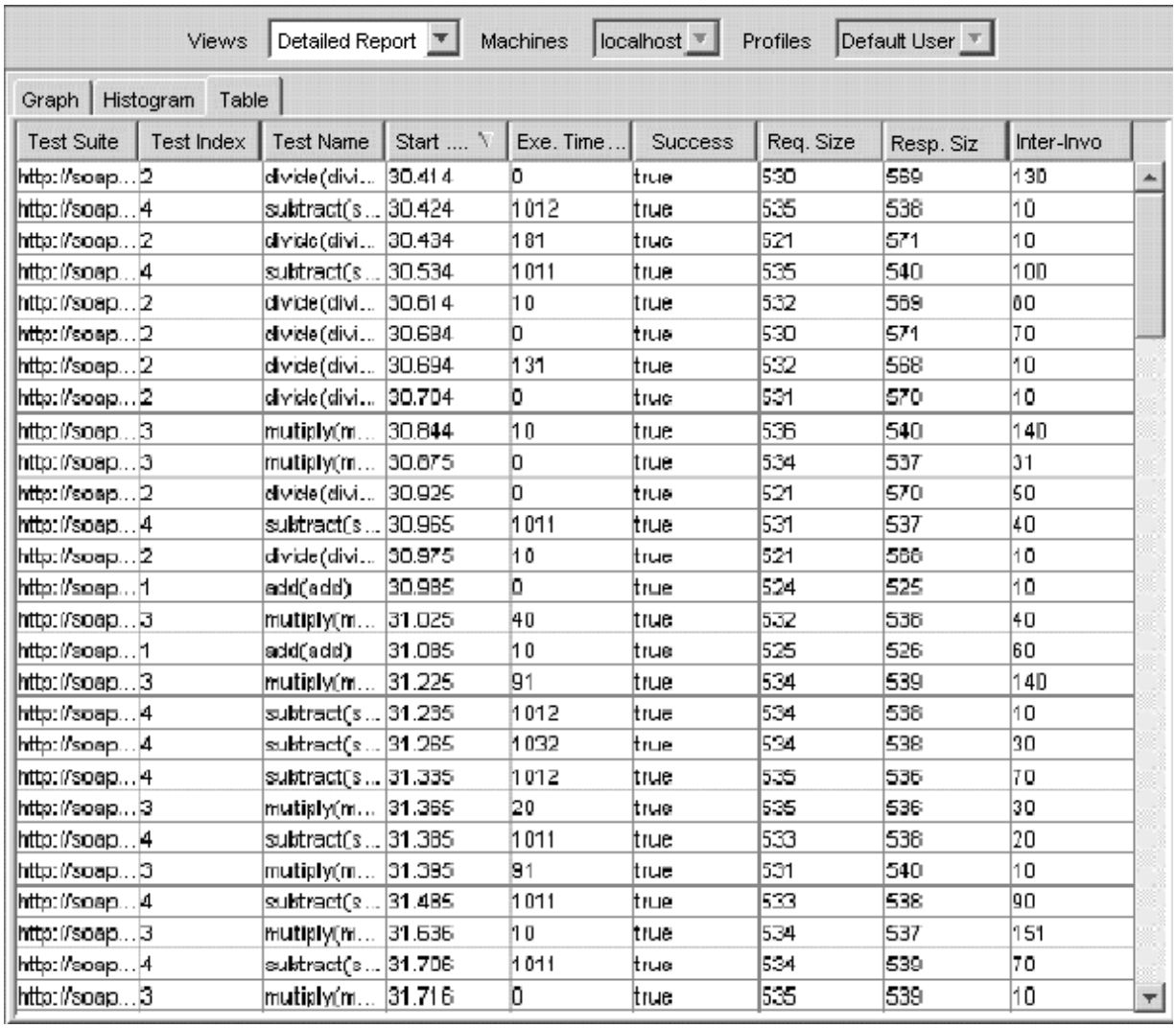


Figura 2.14: Funcionalidad en modo *proxy* y presentación del tráfico de mensajes SOAP con SOAPtest



Test Suite	Test Index	Test Name	Start	Exe. Time...	Success	Req. Size	Resp. Siz	Inter-Invo
http://soap...	2	divide(divi...	30.414	0	true	530	569	130
http://soap...	4	subtract(s...	30.424	1012	true	535	536	10
http://soap...	2	divide(divi...	30.434	101	true	521	571	10
http://soap...	4	subtract(s...	30.534	1011	true	535	540	100
http://soap...	2	divide(divi...	30.614	10	true	532	569	80
http://soap...	2	divide(divi...	30.684	0	true	530	571	70
http://soap...	2	divide(divi...	30.694	131	true	532	568	10
http://soap...	2	divide(divi...	30.704	0	true	531	570	10
http://soap...	3	mutiply(m...	30.844	10	true	536	540	140
http://soap...	3	mutiply(m...	30.875	0	true	534	537	31
http://soap...	2	divide(divi...	30.925	0	true	521	570	50
http://soap...	4	subtract(s...	30.965	1011	true	531	537	40
http://soap...	2	divide(divi...	30.975	10	true	521	566	10
http://soap...	1	add(add)	30.985	0	true	524	525	10
http://soap...	3	mutiply(m...	31.025	40	true	532	536	40
http://soap...	1	add(add)	31.085	10	true	525	526	60
http://soap...	3	mutiply(m...	31.225	91	true	534	539	140
http://soap...	4	subtract(s...	31.235	1012	true	534	536	10
http://soap...	4	subtract(s...	31.285	1032	true	534	538	30
http://soap...	4	subtract(s...	31.335	1012	true	535	536	70
http://soap...	3	mutiply(m...	31.365	20	true	535	536	30
http://soap...	4	subtract(s...	31.385	1011	true	533	536	20
http://soap...	3	mutiply(m...	31.395	91	true	531	540	10
http://soap...	4	subtract(s...	31.485	1011	true	533	538	90
http://soap...	3	mutiply(m...	31.636	10	true	534	537	151
http://soap...	4	subtract(s...	31.706	1011	true	534	539	70
http://soap...	3	mutiply(m...	31.716	0	true	535	539	10

Figura 2.15: Lista de pruebas realizadas por SOAPtest

SOAPtest es una herramienta muy completa, sin embargo, los resultados que obtiene de monitorear servicios Web son muy especializados enfocados únicamente para el entendimiento de diseñadores y programadores de servicios Web. Para usuarios de servicios Web que estén interesados en conocer el funcionamiento de los servicios Web, los resultados de SOAPtest serán difíciles de entender. Por otro lado SOAPtest no utiliza ningún tipo de diagramación que permita interpretar tanto para usuarios como programadores la ejecución de los servicios Web.

Analyzer

Analyzer de StrikeIron [8] es una herramienta que forma parte de StrikeIron Web Services Tools [8]. Analyzer permite realizar pruebas de funcionamiento a servicios Web siendo de gran utilidad ya que permite generar peticiones SOAP y verificar las respuestas de los servicios Web. Analyzer permite probar servicios Web propios así como también realizar invocaciones a servicios Web externos que se encuentren publicados en nodos UDDI.

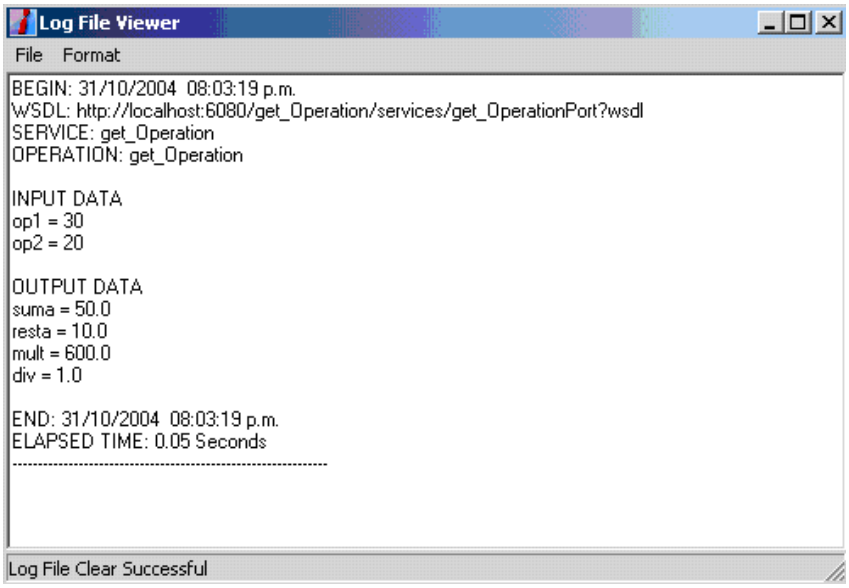


Figura 2.16: Archivo bitácora de pruebas realizadas con Analyzer.

En la Figura 2.16 se muestra un archivo de bitácora que se genera a partir de la invocación de un servicio Web. Estos archivos se almacenan en disco duro para llevar un historial del funcionamiento de los servicios Web.

En la Figura 2.17 se muestra el resultado de invocar un servicio Web. Los elementos de la respuesta obtenida se acomodan dentro de una cuadrícula para permitir discernir cuál de los elemento puede ser incorrecto. Estos resultados también se pueden exportar a archivos de MS Excel.

En la Figura 2.18 se presentan los mensajes de petición y de respuesta de un servicio Web. Estos mensajes se presentan en texto.

Analyzer es una herramienta muy útil para programadores de servicios Web ya que permite realizar pruebas de funcionamiento y revisar los resultados de estas en varias maneras.

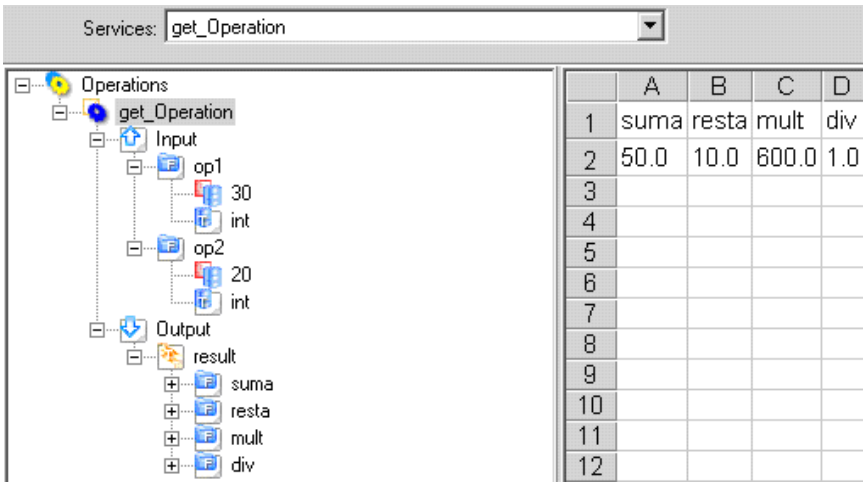


Figura 2.17: Resultados de pruebas realizadas con Analyzer.

Figura 2.18: Presentación de mensajes SOAP con Analyzer

Sin embargo, solamente los programadores tienen el conocimiento técnico para utilizar Analyzer ya que a los usuarios de servicios Web les es difícil interpretar los resultados que Analyzer ofrece. Además, Analyzer no utiliza ningún lenguaje de diagramación que permita interpretar, tanto para usuarios como diseñadores, la ejecución de los servicios Web que utilizan.

Dentro del estado del arte de herramientas que monitorean servicios Web ninguna de

las herramientas presentadas describe gráficamente la interacción y el intercambio de mensajes SOAP utilizando lenguajes estándares de diagramación como UML, lo cual permitiría un mejor entendimiento y análisis de la ejecución de estos servicios. En la siguiente sección se presentan algunas herramientas de diagramación con UML.

2.4.2. Herramientas de diagramación con UML

Hoy en día se pueden encontrar disponibles muchas herramientas que permiten modelar procesos con UML. Sin embargo, en este trabajo no se presentan varias de ellas porque no son muy conocidas o porque su disponibilidad es muy restringida. Únicamente se presentan las herramientas más conocidas y las más utilizadas actualmente.

Rational Rose

Rational Rose de Rational Software [18] es la herramienta de modelado con UML más conocida actualmente. Rational Rose maneja todos los diagramas definidos por la especificación de UML. Permite generar código fuente a partir de diagramas y exportar estos diagramas a documentos XMI por medio de componentes (*plug-ins*). Una característica importante de Rational Rose es que utiliza archivos de texto para almacenar sus diagramas lo cual permite utilizar implementaciones externas para extraer información de los diagramas contenidos en un archivo de Rational Rose.

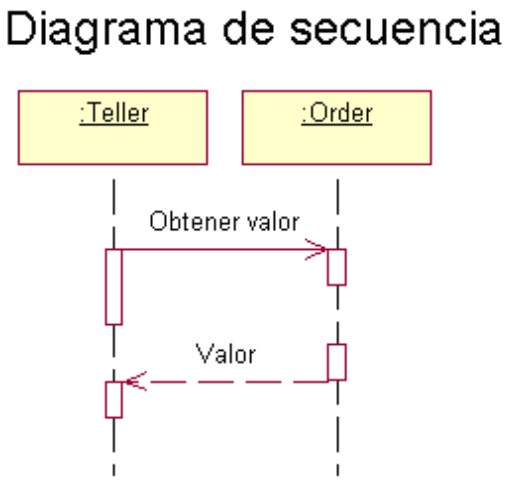


Figura 2.19: Diagrama UML de secuencia creado con Rational Rose

En la Figura 2.19 se presenta un ejemplo de un diagrama UML de secuencia creado con Rational Rose.

Rational Rose no cuenta con alguna opción o componente que esté relacionado con servicios Web. El principal uso de Rational Rose es el de modelar procesos de software con UML pero no monitorearlos.

Visio

Visio [23] de Microsoft [14] es una herramienta de diagramación de propósito general, es decir, se utiliza para crear diagramas de cualquier tipo a partir de galerías de figuras predeterminadas. Visio cuenta con extensiones que permiten generar únicamente diagramas UML de clases, por tal razón Visio no se relaciona directamente con UML. Actualmente Visio cuenta con componentes, en forma de *plug-ins*, los cuales permiten exportar diagramas UML de clases en documentos escritos en XMI o en SVG para visualizar sus diagramas en otras aplicaciones.

Visio no cuenta con componentes relacionados con servicios Web.

ArgoUML

ArgoUML [9] es una herramienta de distribución gratuita. ArgoUML es un proyecto que actualmente está en desarrollo y que cuenta con varias versiones disponibles que se pueden utilizar para modelar en UML. ArgoUML maneja varios diagramas UML, también permite exportar sus diagramas a documentos XMI y SVG. Desafortunadamente ArgoUML usa un formato propietario en codificación binaria de sus diagramas, por lo cual es difícil leer y dar interpretación a la información.

En la Figura 2.20 se presenta un ejemplo de un diagrama UML de secuencia creado con ArgoUML.

ArgoUML no cuenta con componentes relacionados con servicios Web, únicamente se utiliza para modelar en UML.

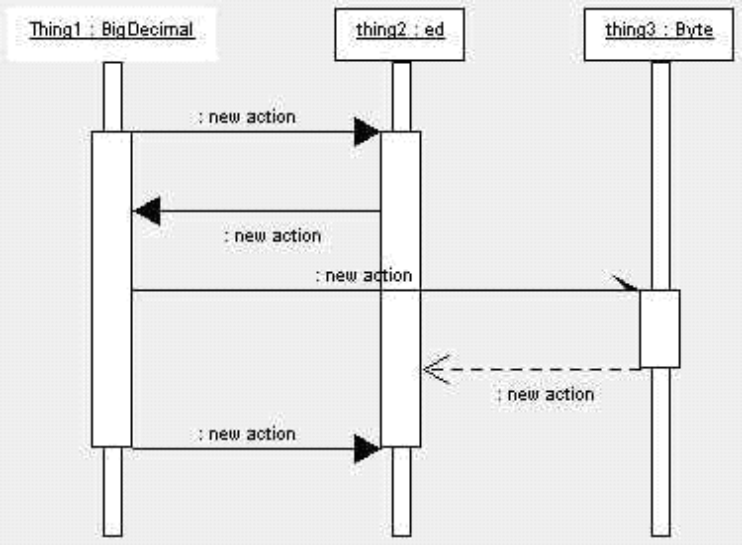


Figura 2.20: Diagrama UML de secuencia creado con ArgoUML

Poseidon

Poseidon [17] de Gentleware es la versión comercial de ArgoUML, aunque cuenta con algunas otras características y mejoras. Poseidon permite modelar procesos empleando varios tipos de diagramas UML. Al igual que ArgoUML, Poseidon utiliza un formato propietario en codificación binaria de sus diagramas que dificulta interpretar el contenido de los diagramas almacenados.

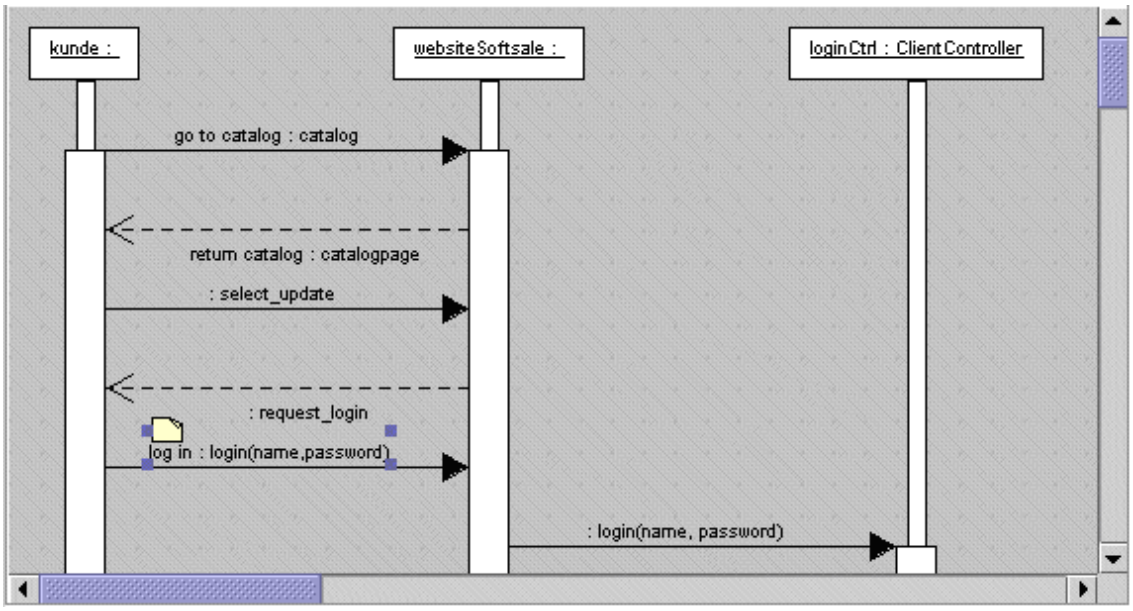


Figura 2.21: Diagrama UML de secuencia creado con Poseidon

En la Figura 2.21 se muestra un ejemplo de un diagrama UML de secuencia creado con Poseidon.

Poseidon no tiene relación alguna con servicios Web ya que no cuenta con componentes enfocados con estos servicios. Esta herramienta únicamente se utiliza para diagramar con UML.

UML2BPEL

UML2BPEL [20] del ETTK de IBM se encuentra en forma de *plug-in* dentro del proyecto Eclipse. Aunque se cuenta con el código fuente de esta herramienta, su uso se encuentra estrechamente ligado con este proyecto. UML2BPEL permite generar documentos WSDL y documentos BPEL a partir de modelos UML. Estos modelos UML se encuentran almacenados en documentos XMI. UML2BPEL se encuentra en desarrollo por lo que no se cuenta con versiones de evaluación ni tampoco con documentación suficiente. Únicamente se cuenta con documentos relacionados con UML2BPEL dentro del ETTK de IBM. UML2BPEL no permite monitorear la ejecución de estos.

Describe

Describe [11] de Embarcadero es una herramienta que permite modelar usando varios tipos de diagramas UML. Describe permite generar código fuente a partir de sus diagramas y utiliza archivos XML para almacenar sus diagramas lo cual permite desarrollar implementaciones externas capaces de manipular diagramas. Describe permite importar modelos UML generados en Rational Rose con lo cual extiende considerablemente su funcionalidad. Sin embargo, Describe no permite almacenar diagramas en documentos XMI.

En la Figura 2.22 se presenta un ejemplo de un diagrama UML de secuencia utilizando Describe.

Describe no tiene ninguna relación con servicios Web ya que no cuenta con opciones o componentes relacionados con estos servicios. Describe únicamente se utiliza para modelar procesos con UML.

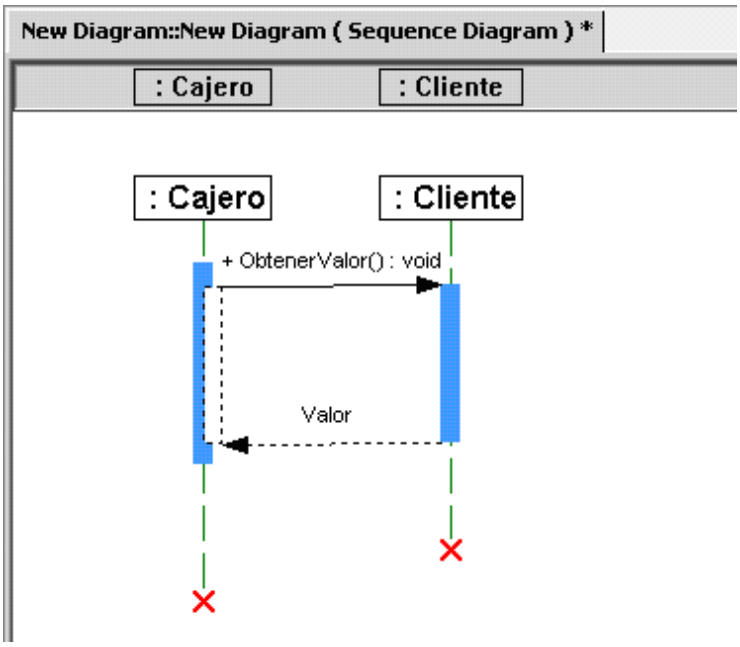


Figura 2.22: Diagrama de UML de secuencia creado con Describe

Ideogramic UML

Ideogramic UML [12] de Ideogramic permite diagramar procesos de software con UML, sin embargo, esta herramienta no soporta todos los diagramas UML. Ideogramic UML almacena únicamente sus diagramas en documentos escritos en XMI.

En la Figura 2.23 se muestra un ejemplo de un diagrama UML de secuencia creado con Ideogramic UML.

Ideogramic UML no tiene componentes relacionados con servicios Web. No cuenta con opciones ni componentes externos que permitan su utilización con estos servicios. Ideogramic UML únicamente se utiliza para modelar con UML.

MagicDraw

MagicDraw [13] es una herramienta muy completa ya que maneja todos los diagramas de UML. MagicDraw genera código fuente a partir de sus diagramas y permite compartir diagramas en red para la interacción con otros diseñadores. Además, permite importar y exportar documentos escritos en XMI y exportar sus diagramas a documentos SVG. MagicDraw es una de las herramientas más recomendadas para modelar con UML ya que

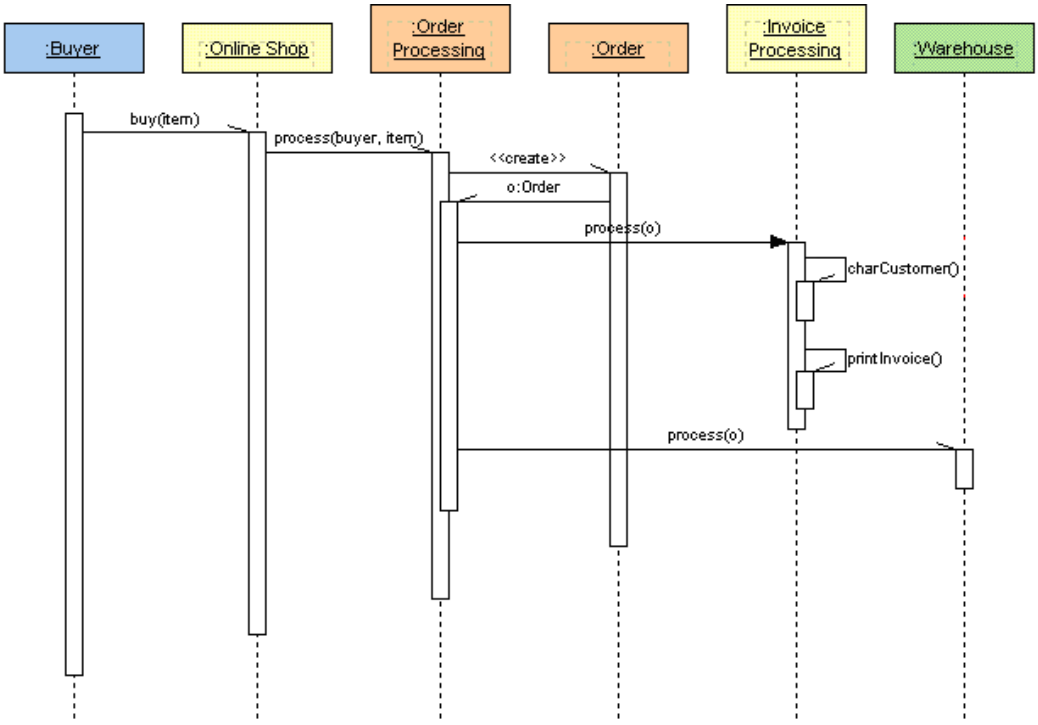


Figura 2.23: Diagrama UML de secuencia creado con Ideogramic

ofrece características de gran utilidad.

En la Figura 2.24 se presenta un ejemplo de un diagrama UML de secuencia creado con MagicDraw.

MagicDraw no permite desarrollar o visualizar la ejecución de un servicio Web. MagicDraw se utiliza principalmente para modelar con UML y diseñar servicios Web utilizando diagramas WSDL.

Estas son algunas de las herramientas que se encuentran disponibles en Internet. Varias de estas herramientas son muy completas, realizan excelentes tareas de monitoreo de servicios Web o modelan procesos de software con UML. Sin embargo, ninguna de estas herramientas soluciona por completo la problemática planteada en este trabajo de investigación.

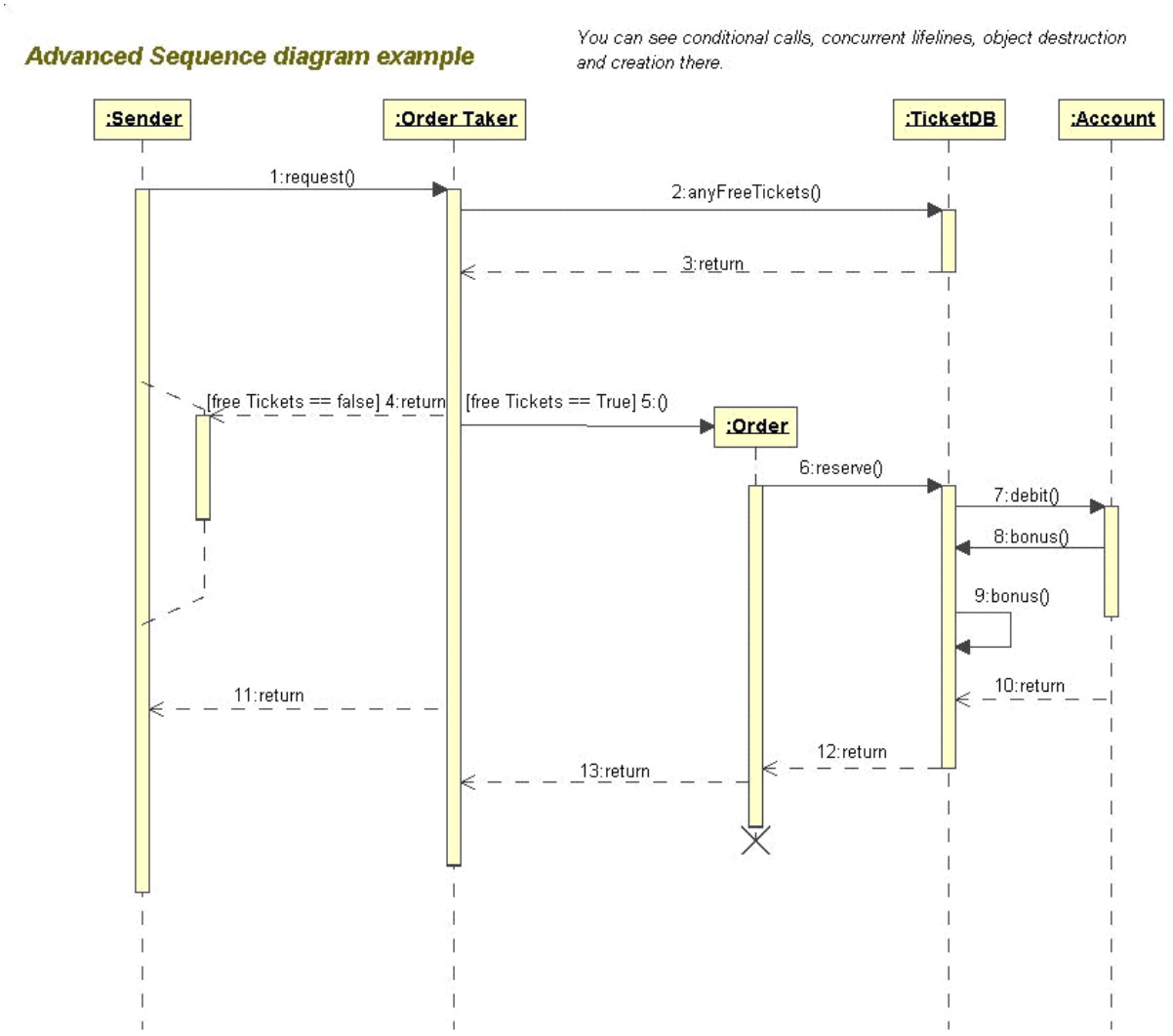


Figura 2.24: Diagrama UML de secuencia creado con MagicDraw

2.5. Conclusiones

A pesar de que los servicios Web son una tecnología relativamente nueva, la implementación de éstos en el comercio electrónico ha crecido de manera significativa, tanto así que los servicios Web son la base de actuales y futuras aplicaciones Web. La principal ventaja de los servicios Web es el uso de XML como lenguaje base, además de los varios dialectos catalogados como estándares de la Web.

Dentro del estado del arte se pueden encontrar varias y muy buenas herramientas de monitoreo de servicios Web y de modelado con UML. Sin embargo, los enfoques de estas herramientas son muy especializados. Por ejemplo, tenemos herramientas de moni-

toreo que presentan resultados enfocados al tráfico de mensajes, difíciles de interpretar para usuarios de servicios Web. Como se puede apreciar también, estas herramientas presentan la información en formato propietario en muchos casos. Por otra parte, tenemos herramientas de modelado en UML que solo realizan esta tarea pero no tienen relación alguna con servicios Web, o bien, algunas que si contemplan componentes relacionados con servicios Web pero no realizan tareas de monitoreo.

Capítulo 3

MS4WS: Sistema de Monitoreo de Servicios Web

En el capítulo anterior se revisó el estado del arte en el monitoreo de servicios Web, haciéndose evidente la importancia del problema abordado debido al gran número de sistemas que han sido desarrollados en estrecha relación con nuestra propuesta así como su originalidad ya que ninguno de ellos se ha enfocado al problema que buscamos resolver: cómo utilizar diagramas UML de secuencia para monitorear el desarrollo de las actividades comerciales. En este capítulo se presenta a MS4WS, un sistema de monitoreo de servicios Web que resuelve este problema. En la primera parte se presenta el contexto en el que operan los servicios Web y la posición de MS4WS en este contexto. Posteriormente se presentan los módulos que lo componen. Finalmente se presenta un caso de estudio enfocado al monitoreo de un sistema de compras en el comercio electrónico utilizando servicios Web.

3.1. Contexto de los servicios Web

Los servicios Web operan sobre una infraestructura de protocolos de comunicación. En la Figura 3.1 se muestra esta infraestructura. Los protocolos de comunicación en la capa de transporte son SOAP, HTTP y TCP/IP. Para lograr la ejecución de servicios Web se intercambian mensajes SOAP a través de estos protocolos de acuerdo a la estructura de

capas que se muestra en la Figura 3.1. En esta misma Figura se muestra la comunicación de dos servicios Web A y B por medio de mensajes SOAP. En este mismo contexto tenemos a MS4WS, el cual se encuentra situado en medio de la comunicación de los servicios Web A y B capturando todos los mensajes que se transfieren. Estos mensajes SOAP se procesan por MS4WS para generar con el menor retraso posible diagramas UML de secuencia, los cuales representan el monitoreo de los servicios Web.

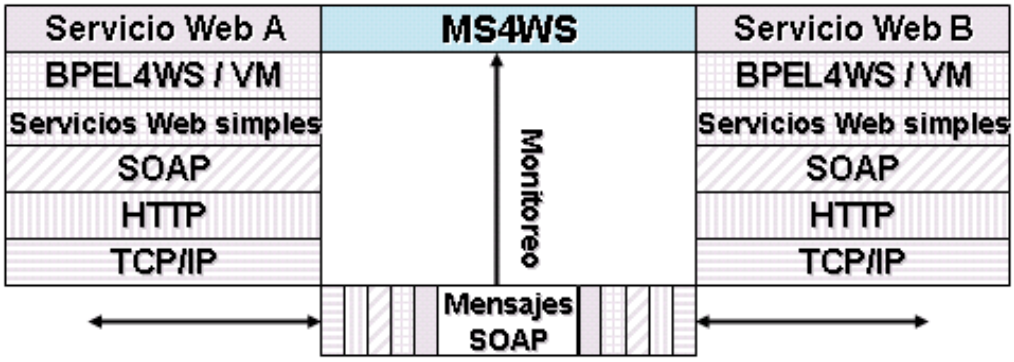


Figura 3.1: Contexto de los servicios Web y MS4WS

3.2. Monitoreo de servicios Web

Para presentar el monitoreo de la ejecución de estos servicios Web MS4WS genera diagramas UML de secuencia, los cuales muestran representaciones UML de los participantes (objetos UML) y de los mensajes SOAP (mensajes UML) intercambiados. Estas representaciones UML se generan con información que se extrae de los mensajes SOAP capturados. Para generar los diagramas UML de secuencia se utilizan documentos escritos en SVG, los cuales permiten visualizar diagramas UML de secuencia a partir de figuras geométricas básicas en varios navegadores de Internet. Los diagramas que MS4WS genera siguen las especificaciones de las actividades comerciales propuestas por el consorcio de RosettaNet [19], un consorcio de compañías especializadas en telecomunicaciones, componentes electrónicos y tecnologías de información. RosettaNet se encarga de establecer estándares ampliamente aceptados por la industria del comercio electrónico. Ejemplos de estos estándares son los PIPs (*Partner Interface Process*) los cuales básicamente definen procesos de negocios entre socios comerciales.

3.3. Módulos de MS4WS

En esta sección se describen los módulos que conforman a MS4WS. MS4WS se compone de 3 módulos. El primer módulo se encarga de la presentación de los resultados del monitoreo de servicios Web. Este módulo se encarga de generar los diagramas UML, de presentarlos en el navegador de Internet y de realizar las actualizaciones necesarias en estos mismos diagramas. El segundo módulo se encarga de transformar los diagramas UML en documentos XMI. El tercer módulo se encarga de visualizar los mensajes SOAP intercambiados por la ejecución de servicios Web.

3.3.1. Generación de diagramas UML

Para presentar el monitoreo de servicios Web MS4WS genera diagramas UML de secuencia de dos formas distintas: la primera a partir del formato propietario de Rational Rose (archivos `.mdl`) y la segunda a partir de documentos XMI generados por alguna herramienta de modelado con UML. La generación de diagramas UML a partir de archivos de Rational Rose o de documentos XMI tiene la finalidad de incluir a MS4WS en el contexto de herramientas de modelado con UML como un componente externo de estas herramientas. Finalmente MS4WS puede generar diagramas UML de secuencia mediante archivos XML de configuración y un mecanismo que posiciona automáticamente los objetos UML en un diagrama.

Diagramas UML de Rational Rose

Los diagramas de Rational Rose se almacenan en forma textual en archivos con extensión `.mdl`. Estos archivos tienen un formato de listas anidadas las cuales se delimitan por paréntesis. Para manipular archivos `.mdl` MS4WS utiliza una librería llamada CrazyBeans [10], la cual contiene las clases necesarias para extraer información de estos archivos. La información que se extrae de estos archivos son, en primer lugar, los diagramas UML se secuencia. Posteriormente, para cada diagrama se extraen los elementos UML con todas sus características, por ejemplo, la posición que ocupan los objetos dentro del diagrama.

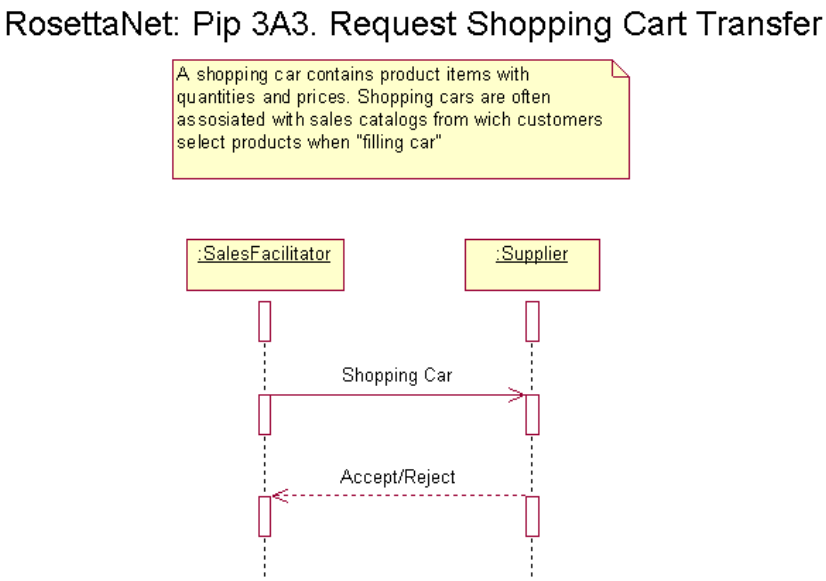


Figura 3.2: Modelo UML de Rational Rose. PIP 3A3 de RosettaNet

En la Figura 3.2 se muestra un diagrama UML de secuencia creado con Rational Rose. Este diagrama modela el PIP 3A3 de RosettaNet. Este diagrama se almacena en un archivo `.mdl` dentro del cual se guardan el diagrama y los elementos UML. Todos los elementos UML se extraen generándose sus respectivas representaciones SVG para los nuevos diagramas. En la Figura 3.3 se muestra el diagrama UML que MS4WS genera a partir del diagrama mostrado en la Figura 3.2.

Diagramas UML en documentos XMI

En la Figura 3.2 se mostró el modelo UML del PIP 3A3 de RosettaNet creado con Rational Rose. Este diagrama se pudo haber creado con alguna otra herramienta, por ejemplo ArgoUML o Poseidon. Estas herramientas, incluyendo a Rational Rose, almacenan sus modelos en diferentes tipos de archivos, algunas de ellas incluso en archivos binarios. Afortunadamente estas herramientas permiten guardar sus diagramas en documentos XMI. El diagrama de la Figura 3.2 se pudo crear con ArgoUML o Poseidon y guardar en un documento XMI del cual se puede extraer información de diagramas y elementos UML.

A partir de un documento XMI MS4WS extrae los elementos XML que describen a los elementos UML. Estos elementos XML contienen las características de los elementos UML las cuales se utilizan para generar las representaciones SVG para el nuevo diagrama.

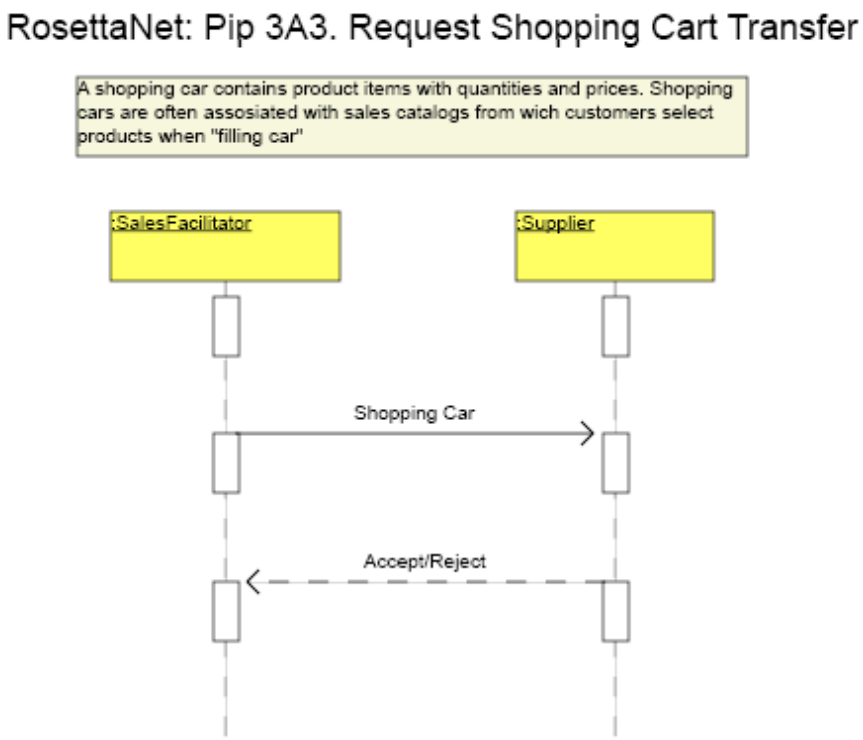


Figura 3.3: Diagrama SVG a partir de un modelo de Rational Rose. PIP 3A3 de RosettaNet

En el apéndice A sección A.1 se muestra el documento XMI del diagrama de la Figura 3.2. El diagrama que MS4WS genera a partir de este documento XMI se muestra en la Figura 3.4.

Este diagrama en principio debería ser idéntico al que se muestra en la Figura 3.3, sin embargo no lo es. Existen diferencias producidas por la serialización del diagrama en XMI. Ejemplos de estas diferencias son las posiciones de los elementos UML, la fuente de letra, etc. Hay otras diferencias en los diagramas que se aprecian a simple vista, por ejemplo, faltan algunos elementos UML. En este nuevo diagrama SVG falta el encabezado del diagrama y focos de control en los objetos UML. La falta de estos elementos UML en el diagrama se debe a que se excluyeron al ser transcritos en XMI, es decir, desde un principio estos elementos UML no aparecen en la serialización del diagrama en XMI.

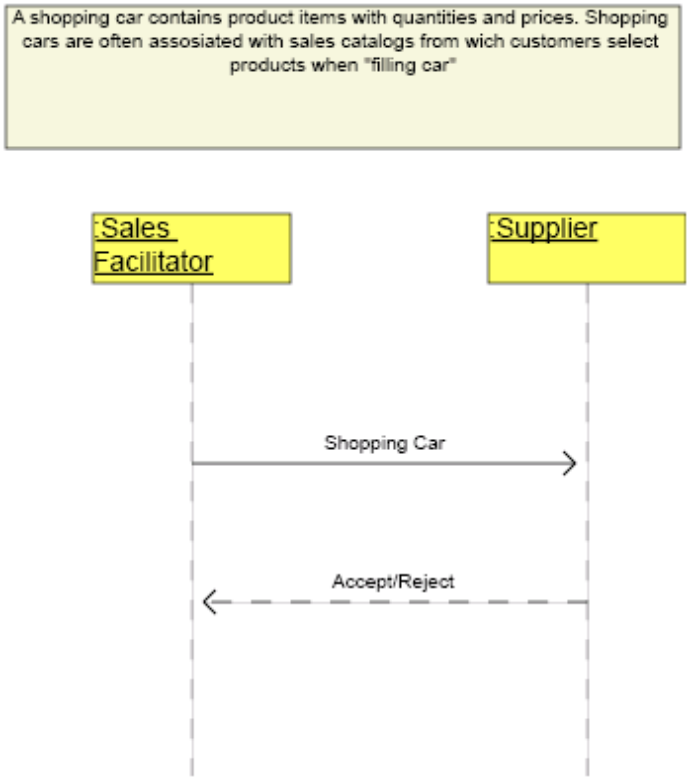


Figura 3.4: Diagrama UML a partir de un documento XMI

Mecanismo para generar diagramas UML

MS4WS permite generar diagramas UML con SVG a partir de objetos abstractos de clases Java. A diferencia de utilizar archivos de Rational Rose o documentos escritos en XMI, MS4WS tiene una librería llamada SVGDiagram la cual contiene clases Java para representar elementos UML de un diagrama UML de secuencia. MS4WS genera diagramas con esta librería creando los objetos abstractos Java que se desean incluir en los diagramas. Estos objetos se almacenan en estructuras de datos de tipo lista. Estas listas se encuentra dentro de la clase SVGDiagram. SVGDiagram a su vez utiliza otras clases para generar todas las representaciones SVG de los objetos Java almacenados.

Las características de los objetos UML se especifican de dos formas: la primera al crearse los objetos Java y la segunda es utilizando archivos XML de configuración. Al crear los objetos Java se pueden especificar las características propias de un objeto UML por medio de los constructores. Otra opción es utilizar un archivo de configuración el cual contiene valores por defecto de las características de objetos UML. La Figura 3.5 muestra

un diagrama UML creado con archivos XML de configuración.

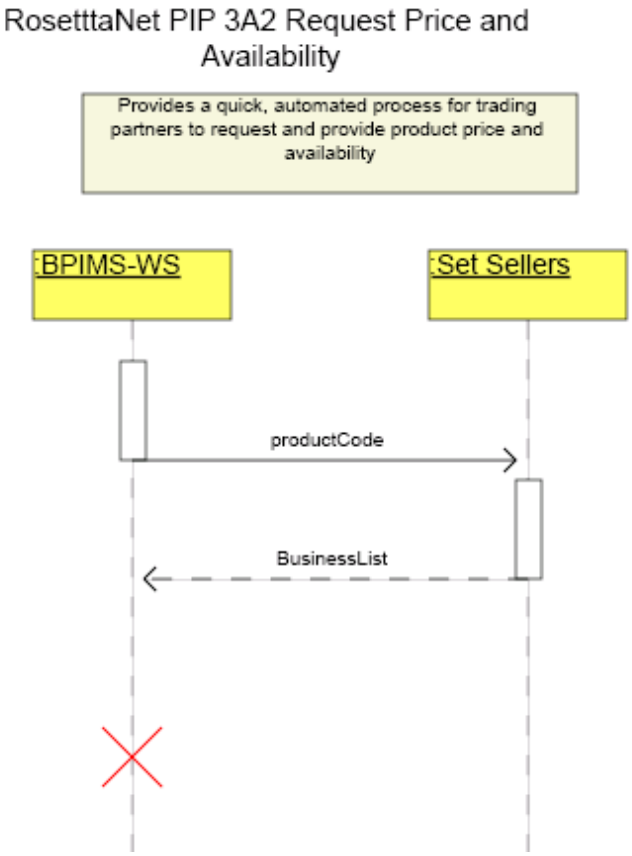


Figura 3.5: Diagrama UML creado con archivos XML de configuración

MS4WS genera diagramas UML de mejor apariencia utilizando únicamente objetos Java que cuando se utilizan archivos de Rational Rose o documentos XMI. El uso de archivos XML de configuración permite adecuar de forma precisa, las características de los objetos UML. Además, en los archivos de configuración también se pueden establecer propiedades de diseño, por ejemplo, la ubicación del primer elemento UML en el diagrama, la separación entre elementos, mensajes, etc.

3.3.2. Representación de diagramas UML en XMI

La funcionalidad de este módulo es representar diagramas UML en XMI. De manera inversa, este módulo genera los elementos XML de un documento XMI a partir de representaciones SVG de elementos UML. Para esta tarea MS4WS cuenta con una librería Java

llamada SVG2XMI. Esta librería utiliza un documento XMI como plantilla la cual contiene elementos XMI de elementos UML. MS4WS modifica estos elementos con los valores de las representaciones SVG de elementos UML. El diagrama de la Figura 3.5 muestra un diagrama UML generado por MS4WS. En el apéndice A se muestra la serialización en XMI de este diagrama.

3.3.3. Visualización de mensajes SOAP

MS4WS permite visualizar los mensajes SOAP presentando éstos en páginas HTML. Utilizando las facilidades que brindan la mayoría de los navegadores de Internet habilitados con SVG, en MS4WS se ha implementado un visualizador de mensajes SOAP. Este visualizador de mensajes SOAP se puede utilizar para realizar tareas de depuración. La visualización de los mensajes SOAP en código XML permite a diseñadores y programadores de servicios Web cerciorarse de que estos mensajes SOAP y los parámetros de invocación son los correctos. Sin embargo, la presentación de los mensajes SOAP en código XML únicamente es difícil de analizar, por esta razón el visualizador también permite presentar los mensajes SOAP con un formato de tablas HTML. Estas presentaciones de los mensajes SOAP en formato de tablas HTML se realizan utilizando hojas XML de estilo, las cuales se generan dinámicamente y generan las tablas HTML necesarias para presentar los mensajes SOAP.

Hasta este punto se han presentado los módulos y el contexto en el que trabaja MS4WS. A continuación se presenta un caso de estudio para ejemplificar la funcionalidad de MS4WS.

3.4. Caso de estudio

Como caso de estudio se presenta una aplicación de comercio electrónico con agentes de compras el cual permite localizar y realizar pedidos de productos en diferentes empresas. En primer lugar se describe brevemente al sistema de intermediación BPIMS-WS [36] para la integración de empresas que ofrecen sus servicios como servicios Web. Posteriormente se describe como BPIMS-WS da soporte a los agentes de compras y los pasos necesarios para realizar una orden de pedido de productos utilizando servicios Web. En esta interacción

se describe en que punto BPIMS-WS comienza a ejecutar los servicios Web y el monitoreo que MS4WS realiza de estas ejecuciones. Finalmente se presentan las vistas del monitoreo de servicios Web con diagramas SVG, el visualizador de mensajes SOAP en forma textual y en tablas HTML y la representación de un diagrama SVG en XMI.

3.4.1. El sistema de intermediación de comercio electrónico

Un sistema de intermediación de comercio electrónico permite coordinar los procesos de negocios de diferentes organizaciones comerciales. Como se mencionó antes, el sistema de intermediación que se utiliza en este trabajo es BPIMS-WS (*Business Process Integration and Monitoring System*), el cual es un sistema de intermediación de comercio electrónico basado en servicios Web.

BPIMS-WS

BPIMS-WS [36] permite la integración, composición y monitoreo de procesos de negocios basados en servicios Web, el cual ofrece funcionalidad adicional para la integración dinámica de empresas, descubrimiento e invocación de procesos de negocio accesibles como servicios Web.

Características de BPIMS-WS

A continuación se presentan las características de BPIMS - WS:

- Contiene un nodo UDDI donde se encuentran registradas empresas comerciales, servicios Web y productos.
- Mantiene clasificaciones de productos siguiendo ontologías ampliamente aceptadas en la Web, como por ejemplo NAICS [15], UNSPSC [22], y RosettaNet.
- Permite registrar, publicar y descubrir nuevos servicios Web.
- Ofrece la composición de servicios Web utilizando documentos escritos en BPEL4WS. Se crean y ejecutan estos servicios en un motor de documentos BPEL4WS.

- Sigue el modelo de un nodo UDDI para mediar entre servicios Web.
- Permite monitorear la ejecución de servicios Web utilizando diagramas UML de secuencia. Este proceso de monitoreo lo realiza MS4WS.

Para usar BPIMS-WS, se puede proceder de dos maneras:

- Mediante el uso de APIs actuando como *proxy*, es decir, mediante aplicaciones Web que implementen búsquedas o invocaciones a nodos UDDI a nivel código al momento de ejecutarse.
- Mediante el uso de páginas HTML actuando como portal de Internet.

BPIMS-WS facilita la integración de aplicaciones de organizaciones comerciales mediante el uso de APIs, permitiendo a programadores desarrollar aplicaciones Web que puedan interactuar dinámicamente con BPIMS-WS. Por otra parte, para clientes de servicios Web, BPIMS-WS ofrece diversos criterios de búsqueda, localización e invocación de servicios Web mediante un portal de Internet.

3.4.2. BPIMS-WS y MS4WS

Para monitorear la ejecución de los servicios Web de BPIMS-WS se ha integrado MS4WS como una parte fundamental. En la Figura 3.6 se muestra la incorporación de MS4WS en BPIMS-WS.

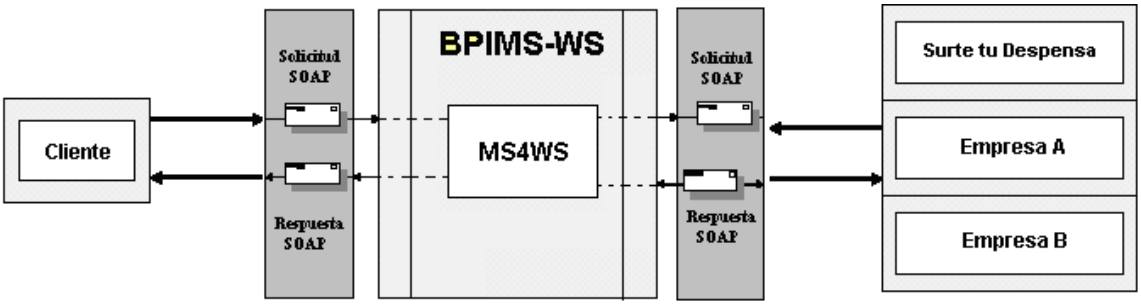


Figura 3.6: MS4WS y BPIMS-WS como intermediario entre un cliente y empresas.

MS4WS monitorea servicios Web generando diagramas UML de secuencia los cuales describen bastante bien la interacción entre BPIMS-WS y otros participantes. Para monitorear servicios Web es necesario:

- Interceptar todos los mensajes SOAP intercambiados.
- Identificar a los participantes y el papel que desempeñan durante la interacción.
- Generar diagramas UML de secuencia utilizando información extraída de mensajes SOAP.

Por cada mensaje SOAP se genera una representación gráfica de éste dentro de un diagrama UML, por ejemplo, objetos de participantes, mensajes simples, mensajes de respuesta, etc. Este proceso de monitoreo de servicios Web permite a los usuarios de BPIMS-WS visualizar la ejecución de los servicios Web en tiempo real.

BPIMS-WS agente de compras electrónico

En este caso de estudio se describe a BPIMS-WS como agente de compras electrónico de productos disponibles en la Web o como un intermediario de empresas de negocios. En esta sección introducimos a Surte tu Despensa (STD de aquí en adelante), un prototipo de empresa electrónica que provee sus productos a través de la Web mediante BPIMS-WS. En la Figura 3.7 se muestran algunas vistas de STD.

Para mostrar el monitoreo entre los mensajes intercambiados entre BPIMS-WS y STD se asume el siguiente escenario:

- STD provee productos a través de la Web.
- STD ha registrado sus productos en BPIMS-WS.
- Los clientes potenciales de STD realizan pedidos utilizando servicios Web.

En este escenario los clientes de STD necesitan invocar servicios Web para realizar pedidos, para esto, los clientes utilizan a BPIMS-WS. BPIMS-WS ofrece dentro de un portal de Internet las interfaces HTML gráficas necesarias para realizar invocaciones de servicios Web.

Figura 3.7: Vistas de Surte tu Despensa

Pasos para realizar un pedido de compras con BPIMS-WS utilizando servicios Web

Para realizar pedidos dentro del portal, se tiene un apartado llamado *buy* que permite realizar órdenes de pedidos de productos registrados. Supongamos que se desea ordenar un pedido; los pasos que el Cliente necesita seguir son:

1. El Cliente elige el producto que desea comprar.

2. Una vez seleccionado el producto, BPIMS-WS presenta criterios que guían al cliente a tomar la decisión de elegir la empresa en la que desea comprar.

- En este punto BPIMS-WS muestra el diagrama UML de secuencia del PIP 3A2 de RosettaNet, el cual describe la solicitud y disponibilidad de productos (imagen de arriba de la Figura 3.8).

3. El Cliente elige a STD.

   - Ahora se muestra el diagrama UML de secuencia correspondiente al PIP 3A1 de RosettaNet, el cual describe a una petición de pedido (imagen de abajo de la Figura 3.8).

4. A partir de este punto es cuando BPIMS-WS empieza a interactuar con STD utilizando servicios Web. Por su parte, MS4WS comienza a monitorear la interacción de BPIMS-WS con STD.

   - En este punto MS4WS se encarga de presentar los diagramas UML de secuencia con información extraída de los mensajes SOAP que se capturan.

5. Ahora el Cliente necesita especificar la cantidad de productos que desea comprar.

   - Una vez establecida la cantidad de productos a comprar, BPIMS-WS se comunica nuevamente con STD para realizar la compra de los productos solicitados. Este proceso de solicitud sigue la especificación del PIP3A4 de RosettaNet (imagen de arriba de la Figura 3.9), el cual describe una solicitud de una orden de pedido.

6. Finalmente, BPIMS-WS genera dinámicamente una solicitud con los elementos necesarios para completar la compra de los productos seleccionados.

   - Una vez que se realiza esta solicitud, BPIMS-WS presenta la respuesta SOAP final de STD (imagen de abajo de la Figura 3.9).

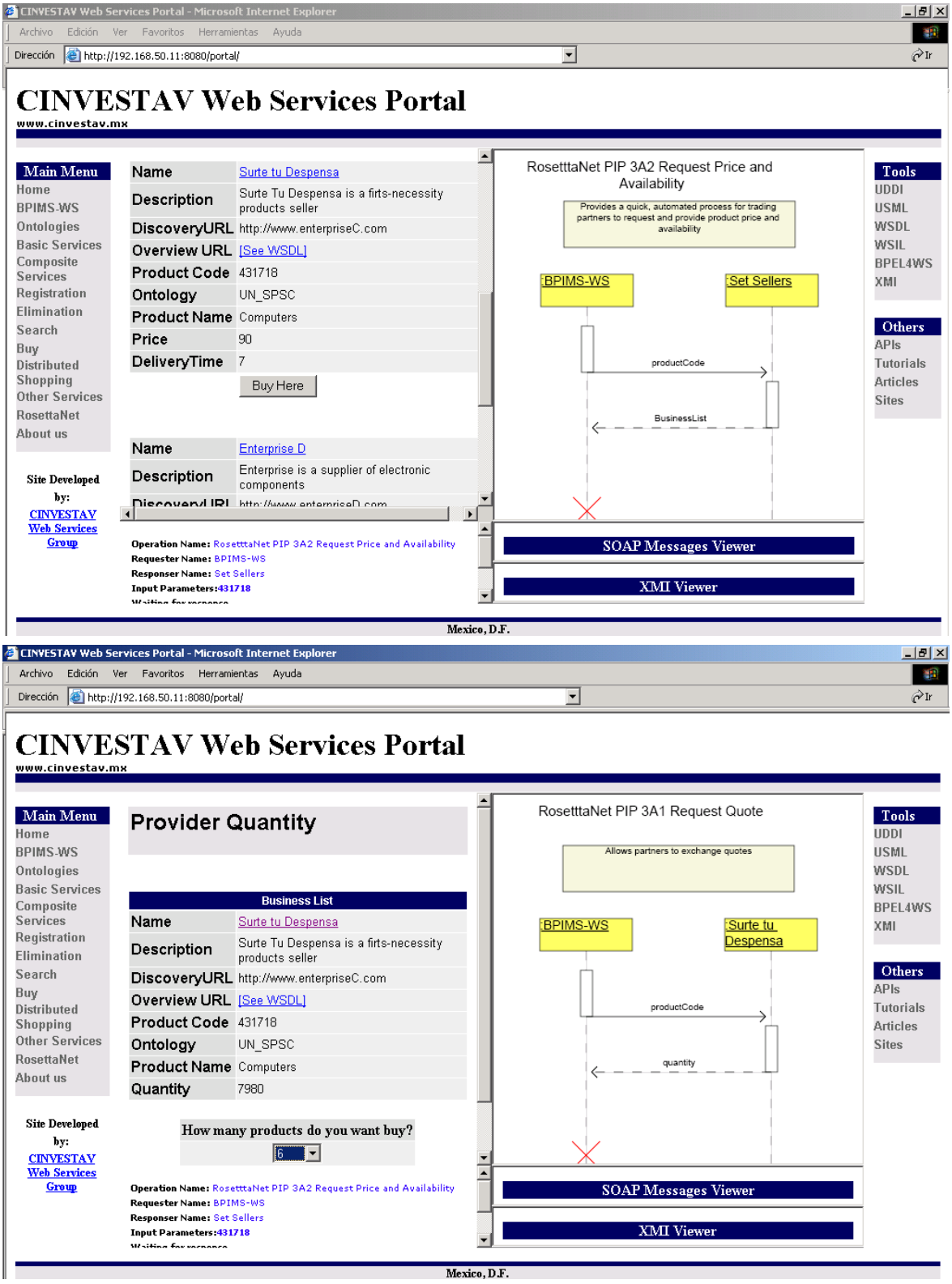


Figura 3.8: Monitoreo de servicios Web con MS4WS (1)

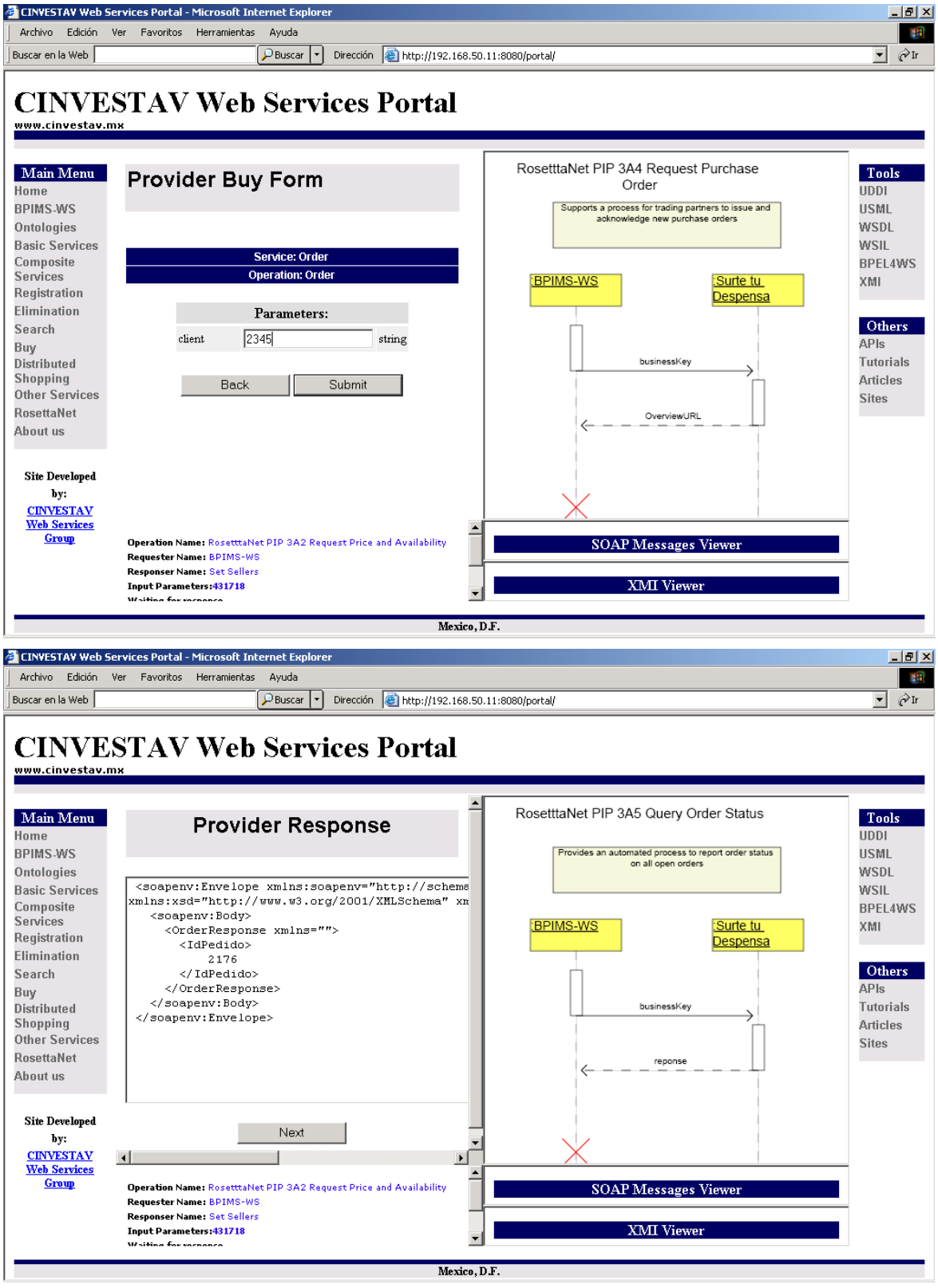


Figura 3.9: Monitoreo de servicios Web con MS4WS (2)

3.4.3. Visualizador de mensajes SOAP

Para visualizar mensajes SOAP, MS4WS proporciona un visualizador de estos mensajes el cual se muestra en las Figuras 3.10 y 3.11. Las imágenes de la Figura 3.10 presentan de forma textual los mensajes SOAP de petición y de respuesta. En esta misma Figura, el visualizador muestra de dos maneras los mensajes SOAP, de manera vertical y horizontal. Las imágenes de la Figura 3.11 presentan los mismos mensajes SOAP y de las dos maneras también mencionadas (horizontal y vertical), sin embargo, en estas imágenes se aprecia que se utilizan tablas HTML para mostrar los mensajes SOAP de una mejor forma más entendible.

3.4.4. Representación de los diagramas de MS4WS en XMI

Los diagramas que se generan con el monitoreo de los servicios Web se pueden serializar en documentos XMI. MS4WS permite serializar cada diagrama generado. En la Figura 3.12 se muestra un diagrama SVG generado por el monitoreo de los servicios de Web de BPIMS-WS. La serialización de este diagrama se muestra en la Figura 3.13.

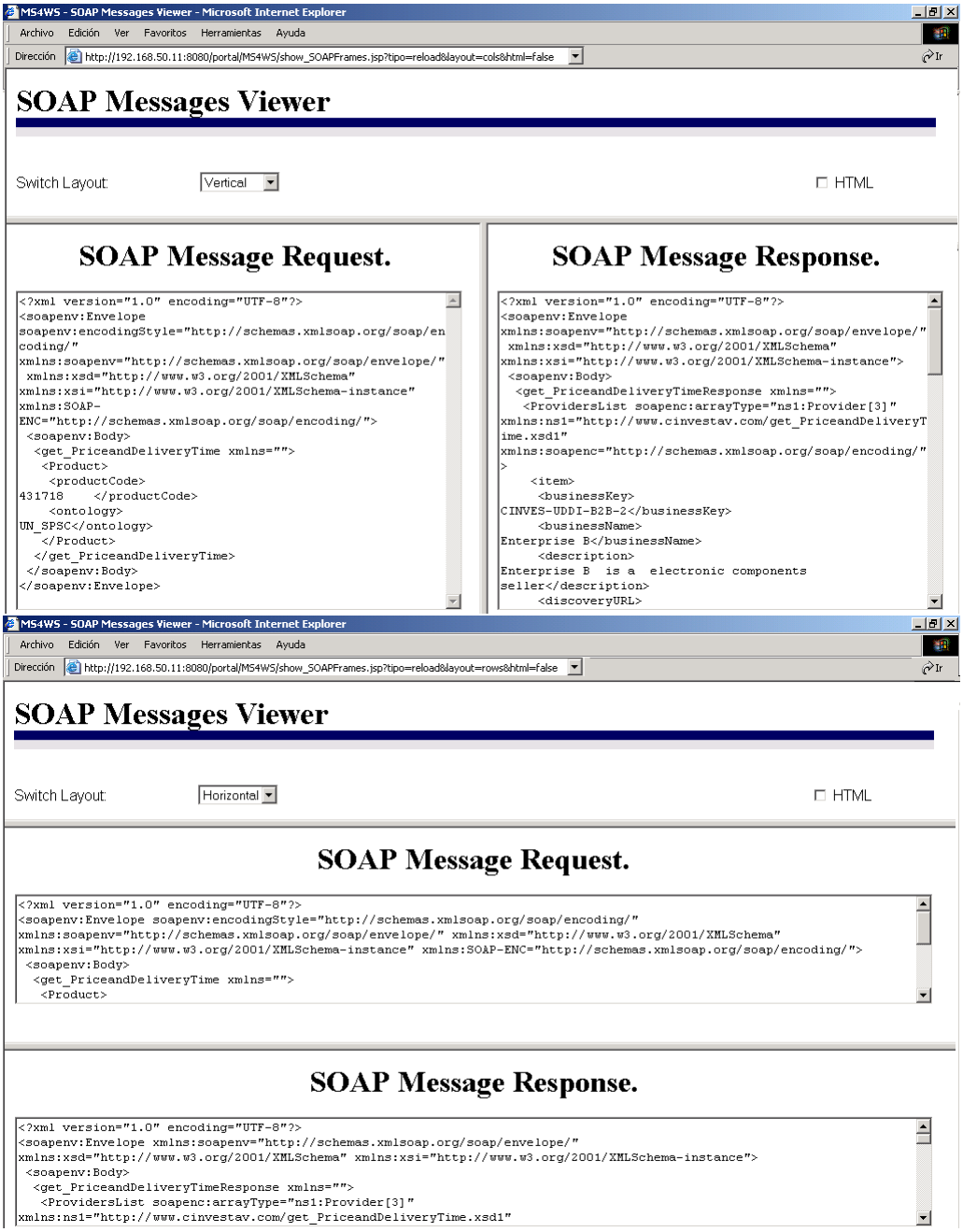


Figura 3.10: Visualizador de mensajes SOAP (1)

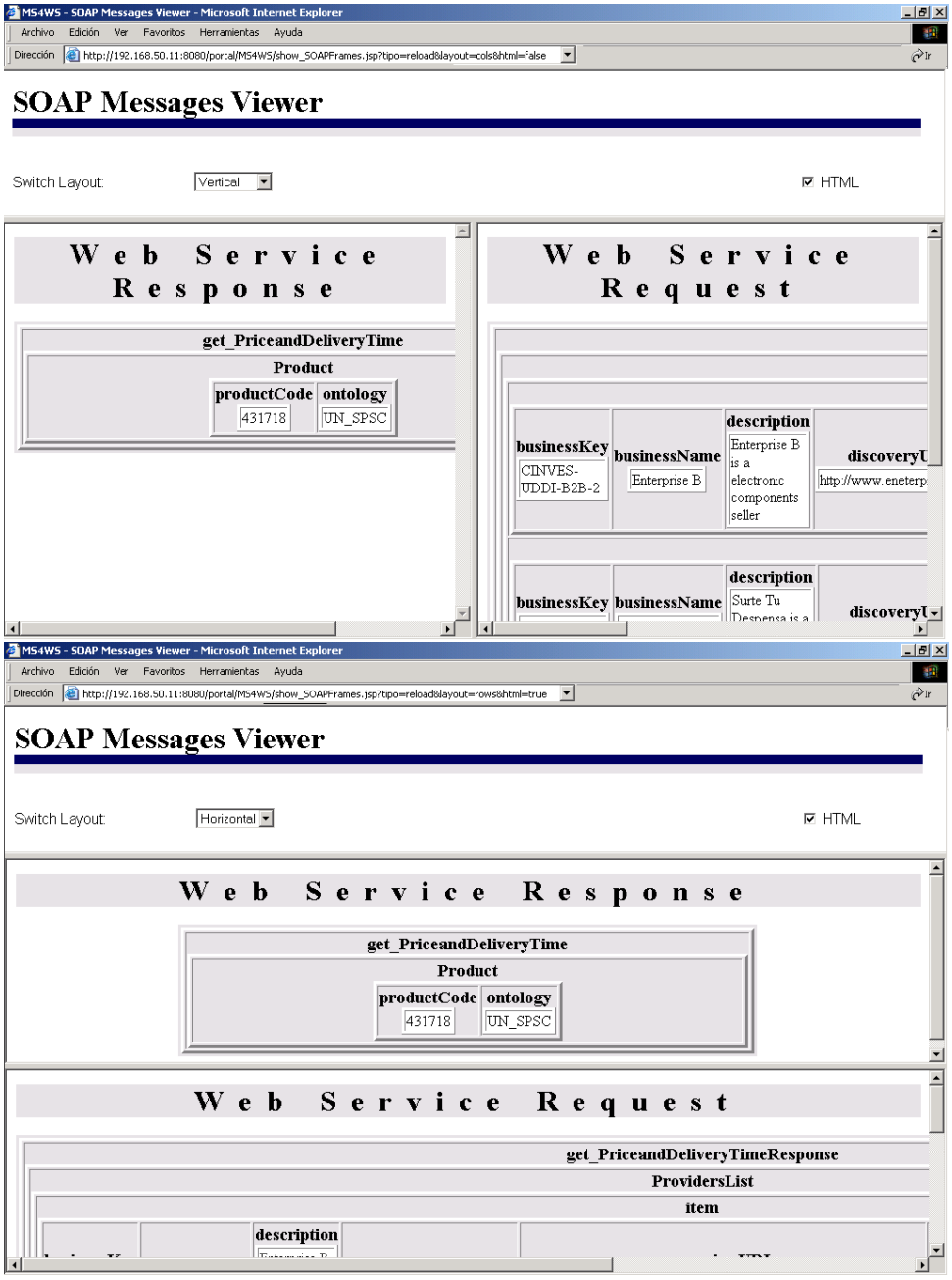


Figura 3.11: Visualizador de mensajes SOAP (2)

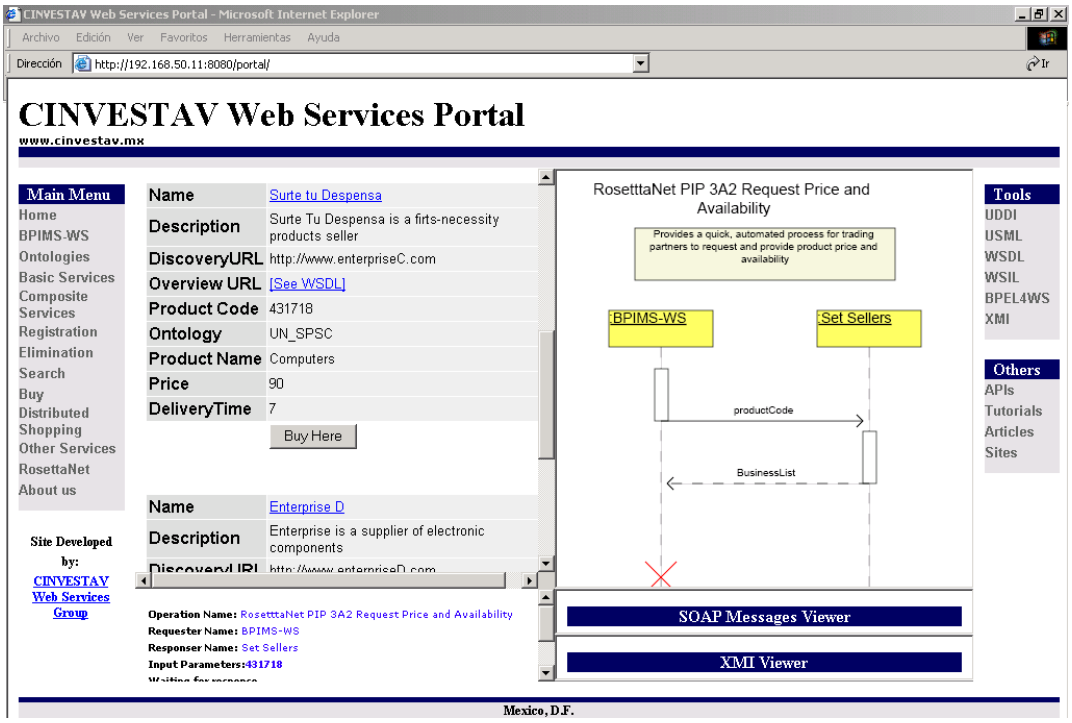


Figura 3.12: Monitoreo de los servicios Web de BPIMS-WS

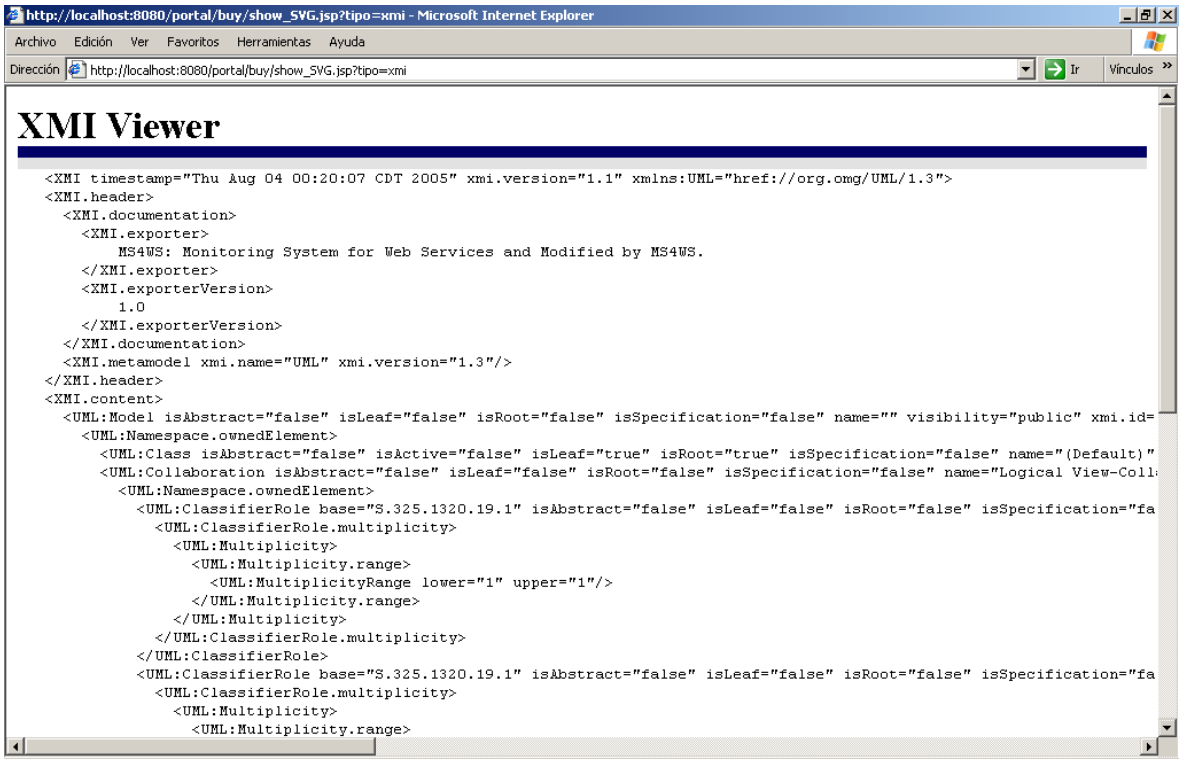


Figura 3.13: Documento XMI del diagrama de la Figura 3.12

3.5. Conclusiones

En este capítulo se presentó el contexto en el que opera MS4WS junto con servicios Web. También se presentaron cada uno de los módulos que componen a MS4WS y la funcionalidad de cada uno de estos. Para ejemplificar el funcionamiento general de MS4WS y de sus módulos se presentó el caso de estudio de un agente de compras electrónico basado en servicios Web. En este caso de estudio se describieron los pasos necesarios para realizar un pedido de productos utilizando servicios Web.

Se puede concluir que MS4WS permite monitorear servicios Web de una manera comprensible, tanto para usuarios de servicios Web como para diseñadores y programadores de servicios Web. MS4WS permite a los usuarios interpretar las operaciones que están realizando utilizando diagramas UML de secuencia cuyo nivel de descripción es suficientemente abstracto e intuitivo para mostrar solamente información relevante del proceso que se realiza (procesos de negocios). Esta forma de presentar la ejecución de servicios Web es muy útil también para diseñadores y programadores de servicios Web. El visualizador de mensajes SOAP permite verificar varios aspectos de depuración, por ejemplo, se pueden verificar parámetros de invocaciones, el direccionamiento de los mensajes SOAP, etc. Además, para los usuarios no expertos el visualizador de mensajes SOAP permite mostrar el contenido de los mensajes SOAP utilizando tablas HTML, las cuales son bastante más legibles en comparación con aquellas de XML. Finalmente se presentó la representación de un diagrama SVG en XMI.

En el capítulo siguiente se presenta el diseño de MS4WS.

Capítulo 4

Diseño de MS4WS

En este capítulo se presenta el diseño de MS4WS. Para su descripción se utilizan diferentes perspectivas: la organizacional, la de datos y la funcional. Con la perspectiva organizacional se presenta el contexto en el que opera MS4WS y los participantes con los que interactúa. Posteriormente, con las perspectivas de datos y de función se presentan las descripciones de las clases y de las librerías que se desarrollaron y que conforman los módulos de MS4WS. Para describir estas clases se utiliza un formato de tablas las cuales permiten describir muy completamente clases Java. Algunas de las librerías desarrolladas utilizan los paquetes externos DOM [47] y CrazyBeans [10]. El paquete DOM cuenta con una amplia documentación disponible en la Web, por esta razón no se incluye la descripción de este paquete. Por otro lado, el paquete CrazyBeans no cuenta con documentación suficiente en la Web, lo cual hace necesario incluir la descripción de este paquete. Sin embargo, CrazyBeans es muy extenso, por lo que únicamente se presentan las descripciones de las clases que se utilizan. Estas descripciones de las clases de CrazyBeans se encuentran en el apéndice C sección C.2. Finalmente en este capítulo se presentan las funciones de MS4WS y de sus módulos utilizando diagramas UML de secuencia.

4.1. Perspectiva organizacional

En esta perspectiva se describen a los participantes que se encuentran dentro del contexto de MS4WS, así como el papel que realiza cada uno de ellos y las relaciones que existen

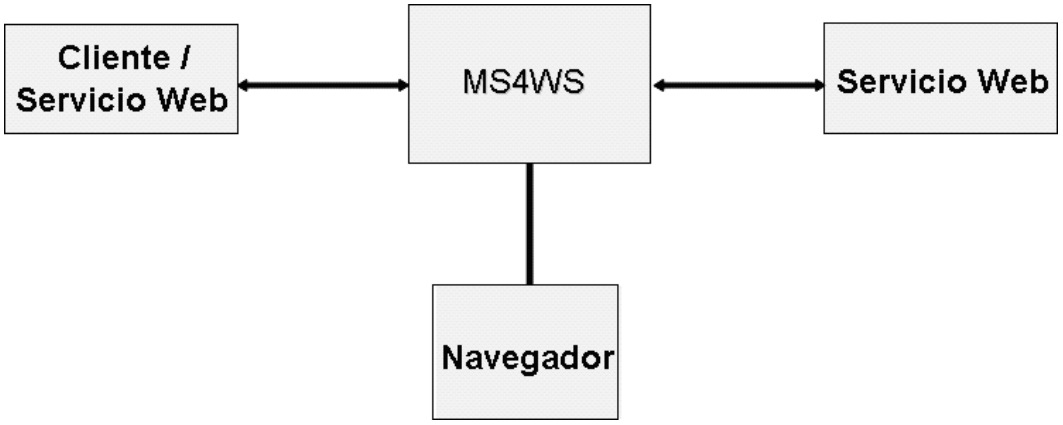


Figura 4.1: MS4WS y los participantes con que interactúa

entre ellos.

Los participantes de este contexto son: el cliente de los servicios Web, el prestador de servicios Web y MS4WS. En este nivel no se considera la interacción de un cliente con servicios Web porque esto es realizado por BPIMS-WS como se explicó antes. En la Figura 4.1 se muestran los participantes y las relaciones que guardan entre ellos.

La posición central de MS4WS en este diagrama refleja su propósito de monitorear los mensajes SOAP que intercambian el cliente y el proveedor del servicio Web. El monitoreo debe cumplir con los requisitos de no interferir con el desarrollo del proceso comercial observado y de mostrar los resultados como diagramas UML en forma inmediata mediante un navegador de Internet.

4.2. Perspectivas de datos y funcional

En estas perspectivas se describen los tipos de datos y funciones que se manejan con MS4WS. Se describen todas las clases Java, librerías y paquetes externos que se utilizan junto con MS4WS. Estas descripciones se presentan en forma de tablas las cuales contienen descripciones breves de los atributos y métodos de cada clase. Se han omitido las descripciones de algunas clases de paquetes externos debido a que no se utilizan. Finalmente se describen los archivos XML de configuración que se utilizan para generar diagramas UML de secuencia.

4.2.1. Generación de diagramas UML

Este módulo se utiliza para generar los diagramas UML de secuencia en formato SVG para monitorear la ejecución de servicios Web. Este módulo lo conforma la librería SVG-Diagram, la cual se utiliza para crear las representaciones SVG de elementos UML especificando todas las características visuales de estos. Otra utilidad de esta librería es crear diagramas UML a partir de objetos Java abstractos que representan elementos UML utilizando archivos XML de configuración.

Librería SVGDiagram

Esta librería se encarga de generar diagramas UML de secuencia en formato SVG. Contiene las clases necesarias para representar los elementos de los diagramas UML de secuencia con elementos SVG. Otra librería externa que se usa es DOM. DOM se usa para crear y manipular documentos XML, XSL y SVG.

Las clases que conforman a esta librerías son: `SVGItem`, `SVGObject`, `SVGDestructor`, `SVGLabel`, `SVGNote`, `SVGFocusOFControl`, `SVGMsg`, `SVGDiagram` y `SVGCreator`.

En el apéndice B sección B.1 se presentan las descripciones de las clases de esta librería.

Generación de diagramas UML con modelos UML de Rational Rose

Este módulo se utiliza para generar las diagramas UML con SVG a partir de diagramas UML de Rational Rose. Se utilizan archivos de Rational Rose debido a que esta herramienta es la más usada para modelar sistemas con UML. Este módulo lo conforma la librería MDL2SVG, la cual se encarga de manipular archivos `.mdl` y de generar los documentos SVG.

Librería MDL2SVG

Esta librería se encarga de convertir diagramas UML de secuencia en el formato MDL de Rational Rose en diagramas UML de secuencia en formato SVG. Esta librería la integra únicamente una clase, la cual se encarga de abrir y extraer diagramas UML de secuencia incluyendo todos sus elementos. Esta librería hace uso del paquete CrazyBeans

y de la librería SVGDiagram. CrazyBeans se utiliza para manipular los elementos UML contenidos en los archivos `.mdl` creados con por Rational Rose. Por otra parte, la librería SVGDiagram se utiliza para crear las representaciones SVG de los elementos UML de archivos `.mdl` y crear los diagramas UML con SVG.

La clase que conforma esta librería es `MDL2SVG`. En el apéndice B sección B.2 se presenta la descripción de la clase de esta librería.

4.2.2. Representación de diagramas UML en XMI

Este módulo se utiliza para describir los diagramas UML de secuencia generados por MS4WS en formato XMI. La ventaja que ofrece este formato es que permite intercambiar los diagramas con casi cualquier herramienta de modelado con UML. Este módulo lo conforma la librería SVG2XMI la cual se encarga de convertir los diagramas SVG en documentos XMI.

Librería SVG2XMI

Esta librería se encarga de serializar diagramas SVG en documentos XMI. Esta librería utiliza al paquete externo DOM para la manipulación de documentos XML.

La clase que conforma esta librería es `SVG2XMI`. En el apéndice B sección B.3 se presenta la descripción de la clase de esta librería.

4.2.3. Visualización de diagramas UML descritos en XMI

Este módulo se utiliza para visualizar los diagramas UML descritos en XMI. En este trabajo únicamente se seleccionaron y procesaron los diagramas UML de secuencia. Este módulo permite visualizar diagramas UML usando SVG sin importar la herramienta de modelado con que fueron creados. Este módulo lo conforma la librería XMI2SVG la cual se encarga de convertir las descripciones XMI en diagramas SVG.

Librería XMI2SVG

Esta librería se encarga de convertir diagramas UML descritos en XMI al formato correspondiente SVG para su visualización. Esta librería utiliza al paquete externo DOM para la manipulación de documentos XML y a la librería SVGDiagram para generar diagramas SVG.

La clase que conforma esta librería es `XMI2SVG`. En el apéndice B sección B.4 se presenta la descripción de la clase de esta librería.

4.2.4. Visualización de mensajes SOAP

Este módulo se utiliza para visualizar mensajes SOAP en páginas HTML y para visualizar el contenido de los mensajes SOAP en código XML o en formato de tablas HTML utilizando hojas de estilo XSL. Para presentar los mensajes SOAP en código XML únicamente presenta el mensaje SOAP tal y como se captura. Para presentar los mensajes SOAP con formato de tablas HTML en este módulo se incluye a la librería SOAP2HTML la cual se encarga de crear hojas de estilo XSL para la generación de tablas HTML.

Librería SOAP2HTML

Esta librería se encarga de crear hojas de estilo XML para mensajes SOAP. Estas hojas de estilo permiten visualizar los mensajes SOAP con tablas HTML. Esta librería utiliza al paquete externo DOM para manipular documentos XML.

La clase que conforma esta librería es `Stylizer`. En el apéndice B sección B.5 se presenta la descripción de la clase de esta librería.

Las librerías que se desarrollaron así como los paquetes externos que se utilizaron para el funcionamiento de MS4WS son las que se presentaron hasta este momento. Sin embargo, algunas librerías utilizan archivos XML de configuración. En la siguiente sección se describen estos archivos de configuración y sus elementos XML.

4.2.5. Archivos de configuración

Los archivos de configuración que se utilizan contienen valores iniciales para comenzar a crear diagramas SVG. En esta sección se presentan lo archivos XML de configuración que se utilizan para generar estos diagramas.

Archivo monitor.xml

En `monitor.xml` se encuentran todos los valores iniciales necesarios para generar un diagrama UML de secuencia en un documento SVG.

En las Figura 4.2 a 4.9 se presentan los elementos que componen a `monitor.xml` y la relación que tienen en los diagramas UML. Arriba de la Figura 4.2 se tienen los valores del nodo `<diagram>`, mientras que abajo se muestran estos valores en el diagrama UML de secuencia correspondiente. Para los demás elementos a la izquierda se tienen los valores de los nodos y a la derecha la relación que tienen dentro de los diagramas.

```
<!-- Defaults para la creacion de un diagrama SVG -->
<!-- Defaults para el svg -->
<diagram>
     <width value="300"/>
     <height value="700"/>
     <background value="white"/>
     <o_height value="35"/>
     <o_x value="100"/>
     <o_y value="150"/>
     <dist_obj value="100"/>
     <dist_msg value="70"/>
     <first_msg value="110"/>
     <line_time value="200"/>
     <msg_time value="0.8"/>
</diagram>
```

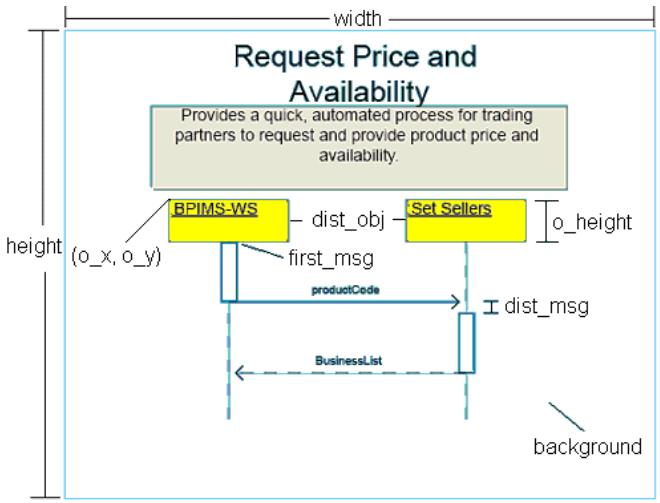


Figura 4.2: Relación de los valores del nodo `<diagram>` en el diagrama

```
<!-- Defaults para los objetos "object" -->
<object>
     <width value="100"/>
     <height value="300"/>
     <long value="100"/>
     <background value="FFFF66"/>
     <font>
          <type value="Arial"/>
          <size value="14"/>
          <color value="5"/>
     </font>
</object>
```

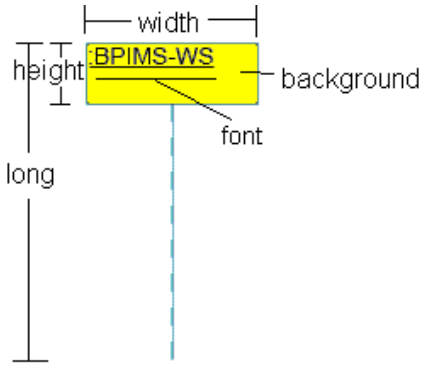


Figura 4.3: Relación de los valores del nodo `<object>` en el diagrama

```
<!-- Defaults para los objetos "mensaje" -->
<message>
     <width value="2"/>
     <arrow value="6"/>
     <font>
          <type value="Arial"/>
          <size value="10"/>
          <color value="0"/>
     </font>
</message>
```

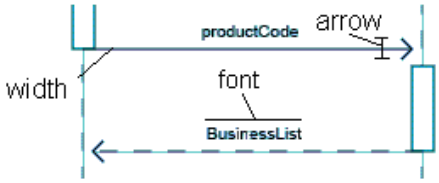


Figura 4.4: Relación de los valores del nodo `<message>` en el diagrama

```
<!-- Defaults para los objetos "etiqueta" -->
<label>
     <background value="blue"/>
     <font>
          <type value="Arial"/>
          <size value="15"/>
          <color value="0"/>
     </font>
</label>
```

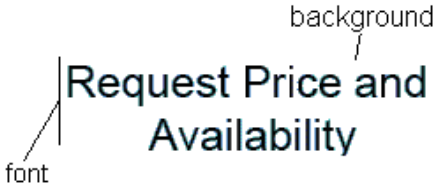


Figura 4.5: Relación de los valores del nodo `<label>` en el diagrama

```
<!-- Defaults para los objetos "Nota" -->
<note>
     <background value="beige"/>
     <align value="center"/>
     <font>
          <type value="Arial"/>
          <size value="10"/>
          <color value="0"/>
     </font>
</note>
```

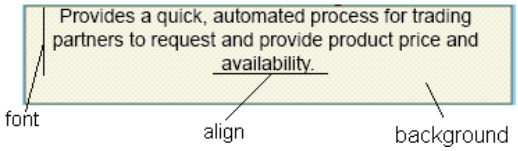


Figura 4.6: Relación de los valores del nodo `<note>` en el diagrama

```
<!-- Defaults para los objetos "Destructor" -->
<destructor>
     <width value="10"/>
     <color value="red"/>
</destructor>
```

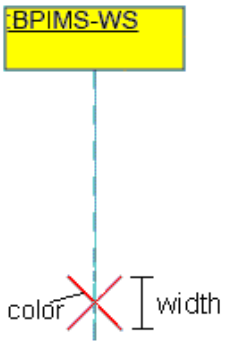


Figura 4.7: Relación de los valores del nodo `<destructor>` en el diagrama

```
<!-- Defaults para los objetos "FocusOfControl"-->
<focusofcontrol>
     <width value="7"/>
     <height value="10"/>
     <background value="white"/>
</focusofcontrol>
```

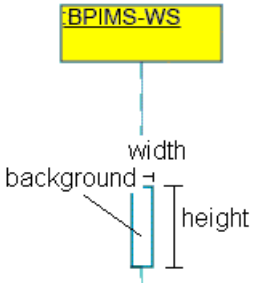


Figura 4.8: Relación de los valores del nodo `<focusofcontrol>` en el diagrama

```
<popup>
     <background value="tan"/>
     <width value="240"/>
     <font>
          <type value="Arial"/>
          <size value="10"/>
          <color value="0"/>
     </font>
</popup>
```

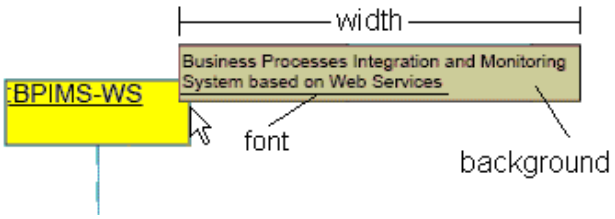


Figura 4.9: Relación de los valores del nodo `<popup>` en el diagrama

Estas descripciones permiten identificar con facilidad los lugares donde se deben realizar los cambios en el archivo de configuración de acuerdo a las adecuaciones que el usuario quiera reflejar en los diagramas UML.

4.3. Funcionalidad de MS4WS

En esta sección se presenta el funcionamiento de MS4WS y las interacciones con otros participantes utilizando diagramas UML de secuencia. También se presenta el funcionamiento de cada uno de los módulos de MS4WS.

4.3.1. Interacción de MS4WS y servicios Web

Como se mencionó antes, los participantes con los que interactúa MS4WS son: un cliente de servicios Web, un prestador de servicios Web y un navegador de Internet. En la Figura 4.10 se muestran estos participantes y las relaciones entre los mismos. MS4WS interactúa mediante servicios Web con clientes y proveedores de servicios Web a través de los mensajes SOAP que se capturan. Por otra parte, MS4WS interactúa con el navegador de Internet a través de los diagramas UML de secuencia que se generan.

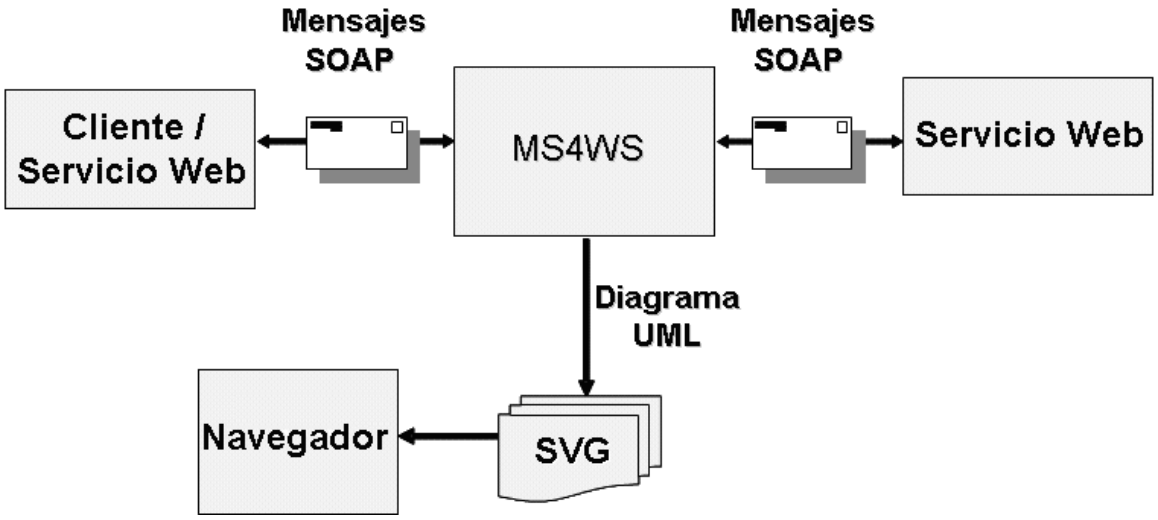


Figura 4.10: Participantes que interactúan con MS4WS y sus relaciones

En la Figura 4.11 se muestra la generación de un diagrama UML de secuencia como el resultado de las interacciones de los participantes. En este diagrama tenemos a un Cliente

que solicita la ejecución de un servicio Web. La interacción inicia cuando el Cliente hace la petición del servicio Web codificado como mensaje SOAP y recibiendo de la misma forma la respuesta correspondiente. Cuando estos mensajes son recibidos por BPIMS-WS antes de redirigirlos a los proveedores correspondientes, MS4WS extrae la información relativa a quien realizó la petición, a quien está dirigida y cuales son los parámetros de la petición. Con esta información se generan los elementos SVG correspondientes de los diagramas UML de secuencia para agregarlos a la historia de los mensajes monitoreados hasta ese momento. Durante la interacción, MS4WS interactúa con el Navegador indicándole que cargue el documento escrito con SVG de acuerdo a las instrucciones dadas en un *script*.

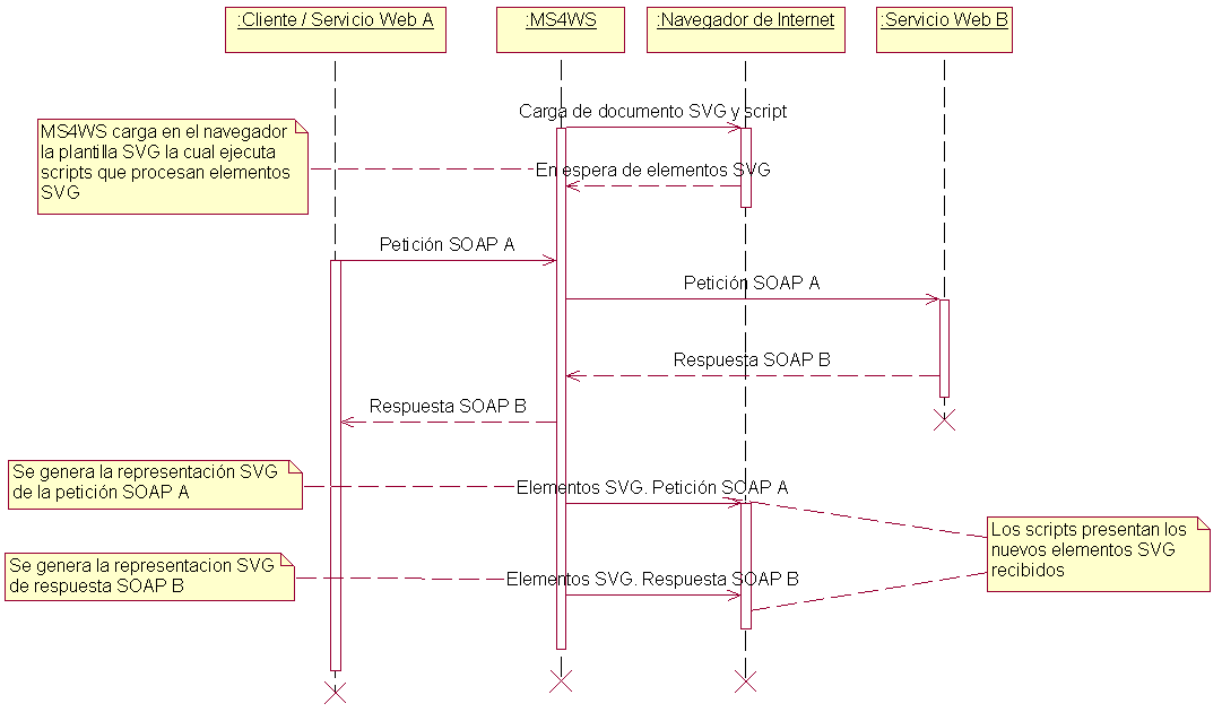


Figura 4.11: Interacción de participantes y MS4WS

La ejecución del *script* en el navegador genera más peticiones a MS4WS de nuevos elementos SVG para añadirlos al documento histórico SVG. Eventualmente, cuando el Cliente envía una petición SOAP (Petición SOAP A) la cual MS4WS captura y reenvía a su destino original, el servicio Web correspondiente procesa la petición y regresa su

respuesta (Respuesta SOAP B). La respuesta también es capturada por MS4WS para su procesamiento y redirigida después al Cliente. Una vez enviados y recibidos los mensajes SOAP, el Cliente y el prestador del servicio Web terminan su interacción. Mientras tanto, MS4WS genera las representaciones SVG de la petición y respuesta SOAP. Una vez que el *script* solicita nuevos elementos SVG, MS4WS le envía las nuevas representaciones SVG que el *script* se encarga de añadir al diagrama UML para su visualización.

La secuencia de las interacciones que se presentan en la Figura 4.11 pueden variar. Se pueden tener algunas variantes, por ejemplo, no siempre MS4WS genera y envía las representaciones SVG tan pronto ha recibido ambas. MS4WS puede generar la representación SVG de la petición SOAP y enviarla al Navegador aún cuando no se ha recibido la respuesta SOAP.

Generación de diagramas UML

En esta sección se presenta la funcionalidad de MS4WS para generar diagramas UML en documentos escritos con SVG a partir de diagramas UML creados con Rational Rose y también de diagramas UML descritos en XMI.

Generación de diagramas UML a partir de modelos UML de Rational Rose

La Figura 4.12 muestra en un diagrama UML de secuencia la generación de diagramas UML con SVG a partir de diagramas UML de Rational Rose. En primer lugar se modela el proceso de negocio o algún PIP de RosettaNet (por ejemplo el PIP 3A2) con Rational Rose utilizando diagramas UML de secuencia. Después Rational Rose almacena este diagrama con formato propietario en un archivo `.mdl`. Entonces MS4WS recibe este archivo y lo analiza. Después MS4WS extrae los diagramas UML de secuencia y los elementos UML de cada diagrama. Para cada elemento UML se genera su representación SVG la cual se añade al diagrama SVG que pertenece. Finalmente MS4WS crea documentos SVG para almacenar los diagramas generados.

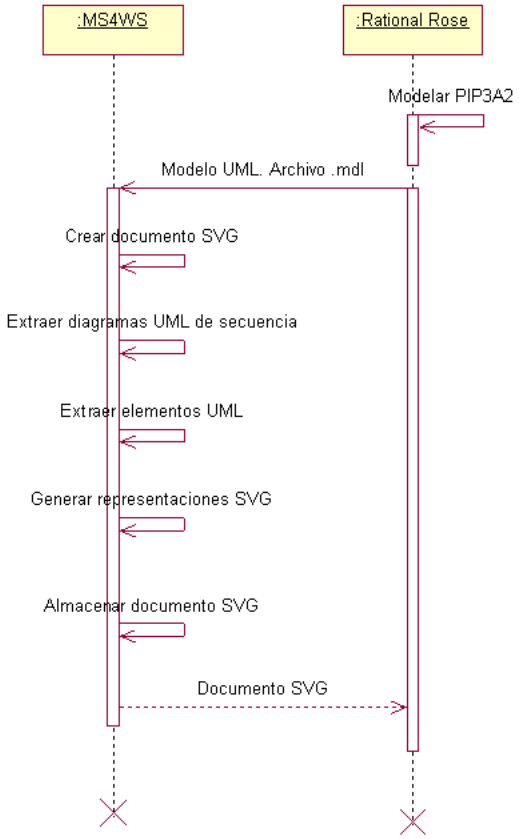


Figura 4.12: Generación de un diagrama UML. PIP 3A2 de RosettaNet

Generación de diagramas UML a partir de documentos XMI

En el diagrama UML de secuencia de la Figura 4.13 se muestra el proceso de generación de diagramas UML en SVG a partir de un modelo UML descrito en XMI. En primer lugar se modela un proceso de negocio o algún PIP de RosettaNet con alguna herramienta de modelado utilizando diagramas UML de secuencia. Después los diagramas se serializan en documentos XMI. MS4WS recibe este documento y lo analiza para extraer los diagramas UML de secuencia y los elementos UML de cada uno de ellos. Para cada elemento UML se genera su representación SVG, la cual se añade al diagrama SVG al que pertenece. Finalmente MS4WS crea documentos SVG para almacenar los diagramas generados.

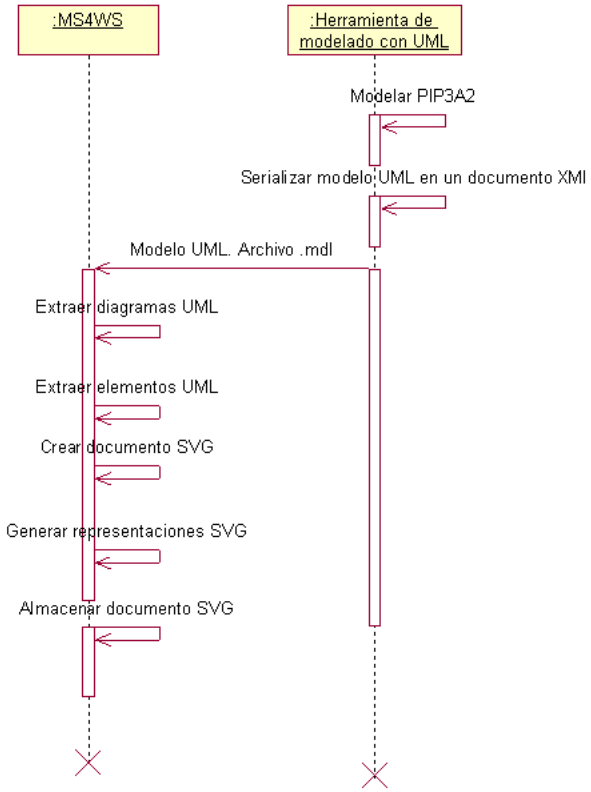


Figura 4.13: Generación de un diagrama UML con SVG a partir de un documento XMI

Serialización de diagramas UML en documentos XMI

En el diagrama UML de secuencia de la Figura 4.14 se muestra el proceso de traducción de diagramas SVG a XMI. En primer lugar el Cliente solicita serializar un diagrama SVG en un documento XMI. MS4WS carga la plantilla XMI que contiene los elementos XMI correspondientes para un diagrama UML de secuencia y todos sus elementos. Después MS4WS extrae todos los elementos UML del documento SVG y genera sus representaciones XMI utilizando la plantilla XMI. MS4WS almacena estas representaciones en un documento XMI. MS4WS presenta el documento XMI al Cliente quien pude visualizar el diagrama serializado en alguna herramienta de modelado con UML.

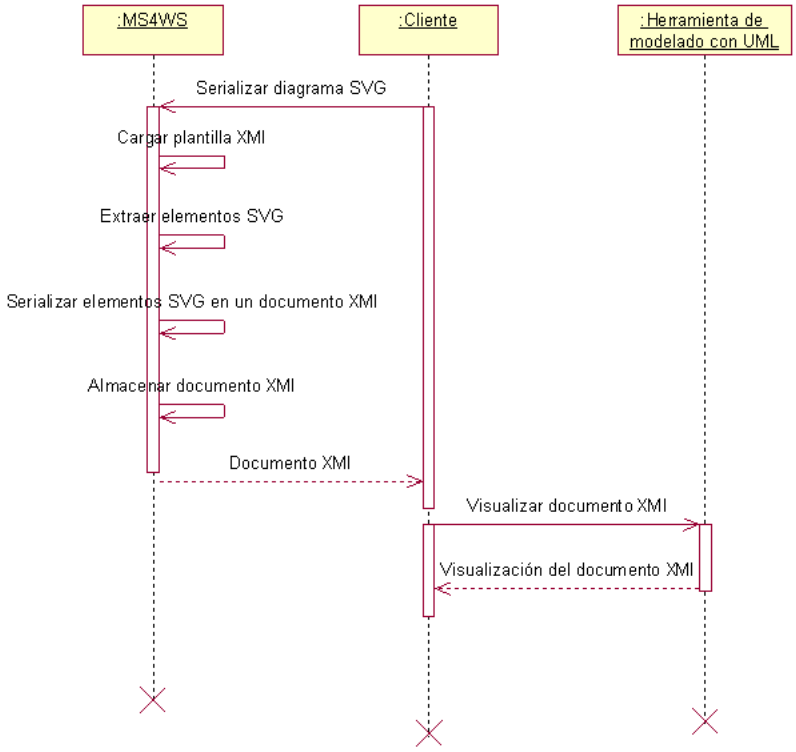


Figura 4.14: Serialización de diagramas SVG en documentos escritos con XMI

Visualización de mensajes SOAP

MS4WS permite visualizar los mensajes SOAP capturados mediante el visualizador de mensajes SOAP que proporciona. Este visualizador permite presentar los mensajes SOAP en código XML o en forma de tablas HTML. En el diagrama UML de secuencia de la Figura 4.15 se muestra el proceso de representación de un mensaje SOAP ya sea en código XML o en forma de tabla HTML.

Como se muestra en este diagrama, MS4WS captura el mensaje SOAP y lo envía al visualizador para presentar el mensaje en el Navegador. Hasta este punto el mensaje se puede presentar en código XML. Sin embargo, el mensaje SOAP también se puede presentar con un formato de tablas HTML. Para presentar el mensaje SOAP con un formato de tablas HTML se utiliza una hoja de estilo (archivo `.xsl`) la cual se crea dinámicamente. Para crear esta hoja de estilo el visualizador de mensajes SOAP carga una plantilla XSL en la cual se especifica el formato de tablas y colores con que se desea

presentar el mensaje SOAP. El visualizador analiza el mensaje SOAP, es decir, verifica la estructura jerárquica de todos los elementos del mensaje. Dependiendo de la estructura del mensaje SOAP se crean o modifican elementos de la plantilla XSL. Finalmente, esta plantilla XSL modificada se almacena en un nuevo documento XSL y al mensaje SOAP se le añade una etiqueta que hace referencia a la nueva hoja de estilo XSL. Esta etiqueta sirve para indicarle al Navegador que utilice la hoja de estilo para presentar el mensaje SOAP.

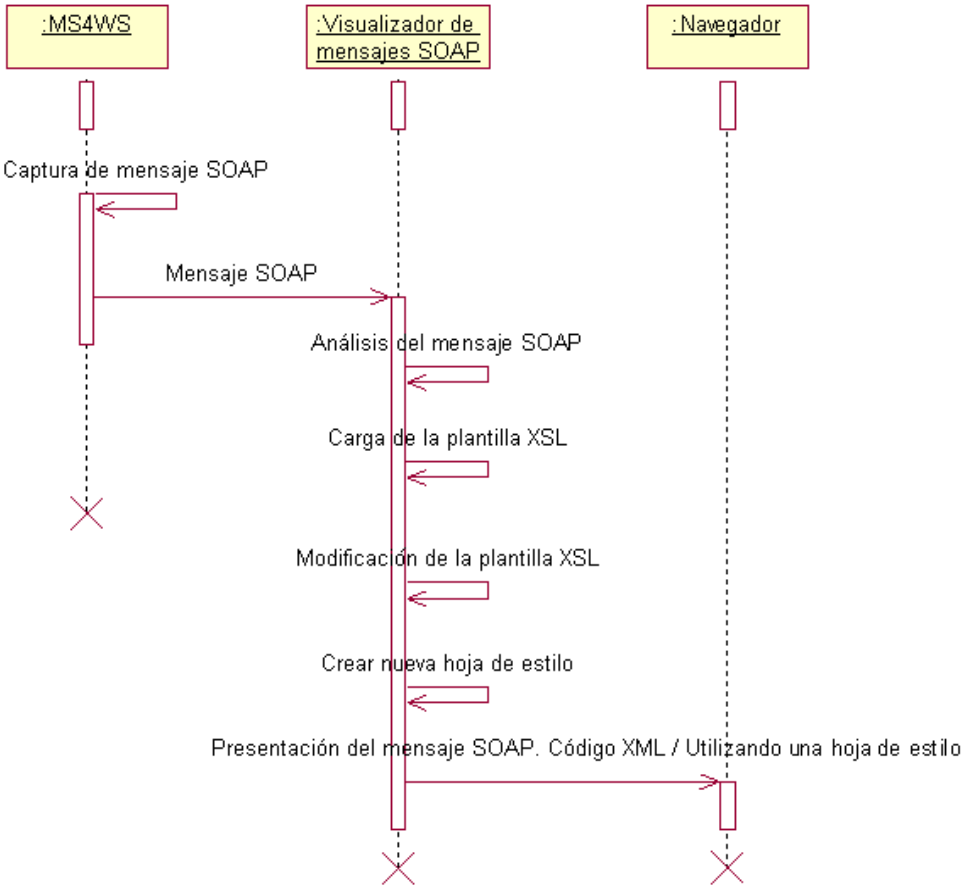


Figura 4.15: Visualización de mensajes SOAP

4.4. Conclusiones

En este capítulo se presentaron las perspectivas organizacional, de datos y funcional para describir el diseño de MS4WS. En la perspectiva organizacional se presentó el contexto

de los servicios Web, la ubicación de MS4WS dentro de este contexto y los participantes con los que interactúa.

En la perspectiva de datos y funcional se presentaron los módulos, librerías, paquetes y clases que conforman a MS4WS. Se describieron los datos y tipos de datos que manejan estas clases. También se presentaron los archivos de configuración que se utilizan presentando los elementos de éstos y la relación que tienen los diagramas.

Finalmente se presentaron las funciones de MS4WS y de sus módulos. Se utilizaron diagramas UML de secuencia para describir las interacciones de MS4WS con los participantes del contexto de los servicios Web. También se presentó la funcionalidad de cada módulo de MS4WS.

En el siguiente capítulo se presenta la implementación de MS4WS.

Capítulo 5

Implementación de MS4WS

La implementación de MS4WS se desarrolló utilizando lenguajes estándares de la Web. La enorme disponibilidad de las herramientas desarrolladas para estos lenguajes permitieron implementar a MS4WS sin muchos problemas. El lenguaje base de programación que se utilizó es Java, el cual se ajusta a las necesidades del contexto distribuido en el que opera MS4WS. Para presentar los mensajes SOAP en páginas HTML con el visualizador de mensajes SOAP se utilizaron JSPs y hojas XML de estilo. En lo que respecta a la funcionalidad de los servicios Web, MS4WS utiliza los dialectos de XML establecidos como estándares. Se utilizaron diagramas UML de Rational Rose como plantillas y documentos XMI del *plug-in* de Unisys instalado en Rational Rose.

A partir del diseño presentado en el capítulo 4, en este capítulo se presentan las implementaciones Java de las librerías SVGDiagram, MDL2SVG, SVG2XMI, XMI2SVG y SOAP2HTML con diagramas UML de clases. También se incluyen descripciones panorámicas de la utilidad de cada una de las clases. Nuevamente, del paquete externo DOM no se presentan las descripciones de las clases que se utilizan. Por otra parte, las implementaciones de CrazyBeans se encuentran en el apéndice C sección C.1.

5.1. Librería SVGDiagram para generar diagramas UML

La librería SVGDiagram tiene la finalidad de generar diagramas UML de secuencia en documentos SVG. Para esto se desarrollaron clases Java que representan a los elemen-

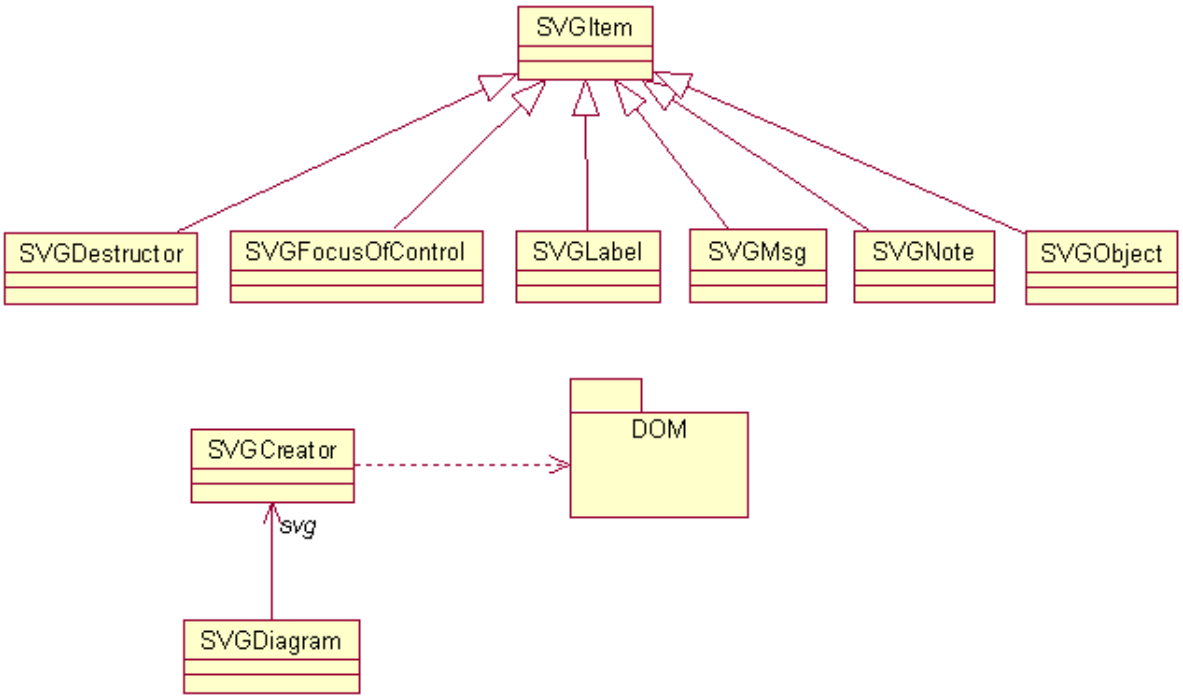


Figura 5.1: Diagrama de clases de la librería SVGDiagram

tos UML de estos diagramas. Esta librería se utiliza para generar diagramas UML en documentos SVG a partir de diagramas UML contenidos en archivos `.mdl`. También se utiliza esta librería para monitorear la ejecución de servicios Web con diagramas UML de secuencia.

En la Figura 5.1 se muestra el diagrama de clases de la librería SVGDiagram. En este diagrama se muestran las relaciones que existen entre clases y otros paquetes. Esta librería utiliza el paquete externo DOM para crear y manipular elementos XML de documentos escritos con SVG. Las clases `SVGDestructor`, `SVGFocusOfControl`, `SVGLabel`, `SVGMsg`, `SVGNote` y `SVGObject` heredan de la misma clase `SVGItem`. Las clases `SVGDestructor`, `SVGFocusOfControl`, `SVGLabel`, `SVGMsg`, `SVGNote` y `SVGObject` representan a diferentes elementos UML en un diagrama UML de secuencia. Por su parte `SVGItem` es un elemento UML general que representa genéricamente a cualquier elemento UML. Las clases `SVGDiagram` y `SVGCreator` se asocian a través del atributo `svg` de `SVGDiagram` el cual es de tipo `SVGCreator`.

`SVGDiagram` es la clase principal junto con `SVGCreator`. `SVGDiagram` se utiliza para

inicializar un diagrama y añadir elementos UML al diagrama. `SVGDiagram` junto con `SVGCreator` crean los documentos escritos en SVG y todos los elementos SVG necesarios para construir diagramas UML de secuencia.

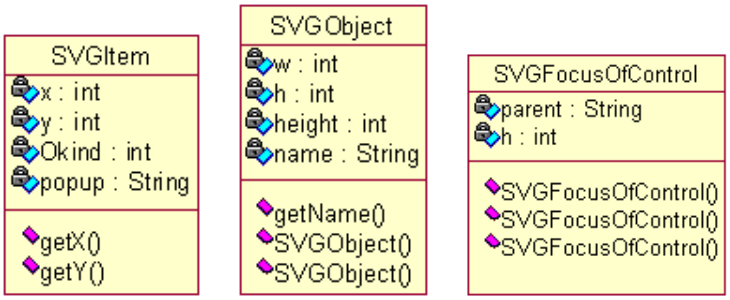


Figura 5.2: Clases SVGItem, SVGObject y SVGFocusOfControl

5.1.1. La clase SVGItem

A la izquierda de la Figura 5.2 se muestra la estructura de la clase `SVGItem`. Esta es la superclase de todas las otras clases que representan elementos UML de diagramas UML de secuencia. Todos los elementos UML de un diagrama son instancias de clases derivadas de `SVGItem`. `SVGItem` se utiliza como elemento genérico, contiene los atributos necesarios para representar a un elemento gráfico simple. Los métodos de `SVGItem` son muy sencillos, únicamente se utilizan para acceder a los valores de los atributos. En la Tabla B.1 del apéndice B se presentan los métodos y atributos de esta clase.

5.1.2. La clase SVGObject

En el centro de la Figura 5.2 se muestra la estructura de la clase `SVGObject`. Esta clase representa un objeto UML de un diagrama UML de secuencia. `SVGObject` es un elemento UML de un diagrama, por lo que hereda de `SVGItem` los atributos necesarios para representar a un elemento gráfico. `SVGObject` se utiliza para generar representaciones SVG de objetos UML. Los atributos de `SVGObject` se utilizan para guardar las propiedades gráficas de un objeto UML. El único método de `SVGObject` es `getName`, el cual se utiliza para obtener el nombre del objeto UML que se representa. `SVGObject` tiene 2 constructores

debido a que SVGDiagram permite generar diagramas de distintas formas, por ejemplo utilizando archivos de configuración o generando diagramas UML sin ninguna referencia. En la Tabla B.2 del apéndice B se presentan los métodos y atributos de esta clase.

5.1.3. La clase SVGFocusOfControl

A la derecha de la Figura 5.2 se muestra la estructura de la clase `SVGFocusOfControl`. Esta clase representa un foco de control desplegado a lo largo de la línea de tiempo de un objeto en un diagrama UML de secuencia. `SVGFocusOfControl` es un elemento UML de un diagrama, por lo que hereda de `SVGItem` los atributos necesarios para representar un elemento gráfico. `SVGFocusOfControl` se utiliza para generar representaciones SVG de los focos de control UML. Los atributos de `SVGFocusOfControl` se utilizan para guardar las propiedades gráficas de un foco de control UML. `SVGFocusOfControl` no tiene métodos propios. `SVGFocusOfControl` tiene 3 constructores por la misma razón que se mencionó en la descripción de la clase `SVGObject`. En la Tabla B.5 del apéndice B se presentan los métodos y atributos de esta clase.

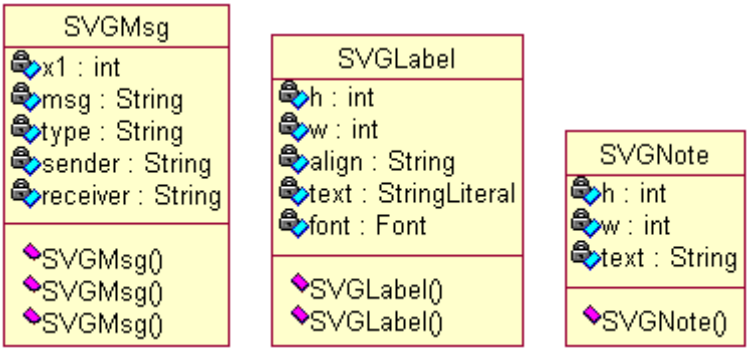


Figura 5.3: Clases SVGMsg, SVGLabel y SVGNote

5.1.4. La clase SVGMsg

A la izquierda de la Figura 5.3 se muestra la estructura de la clase `SVGMsg`. Esta clase representa un mensaje de un diagrama UML de secuencia. `SVGMsg` es un elemento UML de un diagrama, por lo que hereda de `SVGItem` los atributos necesarios para representar un

elemento gráfico. Los atributos de `SVGMsg` se utilizan para guardar las propiedades gráficas de un mensaje UML. `SVGMsg` no tiene métodos propios. `SVGMsg` tiene 2 constructores por la misma razón que se mencionó en la descripción de la clase `SVGObject`. En la Tabla B.6 del apéndice B se presentan los métodos y atributos de esta clase.

5.1.5. La clase SVGLabel

En el centro de la Figura 5.3 se muestra la estructura de la clase `SVGLabel`. Esta clase representa una etiqueta de un diagrama UML de secuencia. `SVGLabel` es un elemento UML de un diagrama, por lo que hereda de `SVGItem` los atributos necesarios para representar un elemento gráfico. Los atributos de `SVGLabel` se utilizan para guardar las propiedades gráficas de una etiqueta UML. `SVGLabel` tiene 2 constructores por la misma razón que se mencionó en la descripción de la clase `SVGObject`. En la Tabla B.4 del apéndice B se presentan los métodos y atributos de esta clase.

5.1.6. La clase SVGNote

A la derecha de la Figura 5.3 se muestra la estructura de la clase `SVGNote`. Esta clase representa una nota UML de un diagrama UML de secuencia. `SVGNote` es un elemento UML de un diagrama, por lo que hereda de `SVGItem` los atributos necesarios para representar un elemento gráfico. Los atributos de `SVGNote` se utilizan para guardar las propiedades gráficas de una nota UML. `SVGNote` no tiene métodos propios, únicamente un constructor. En la Tabla B.3 del apéndice B se presentan los métodos y atributos de esta clase.

5.1.7. La clase SVGDestructor

A la izquierda de la Figura 5.4 se muestra la estructura de la clase `SVGDestructor`. Esta clase representa un destructor en un diagrama UML de secuencia. `SVGDestructor` es un elemento UML, por lo que hereda de `SVGItem` los atributos necesarios para representar un elemento gráfico. Los atributos de `SVGDestructor` se utilizan para guardar las propiedades gráficas de un destructor UML. `SVGDestructor` no tiene métodos propios. `SVGDestructor` tiene 3 constructores por la misma razón que se mencionó en la descripción de la clase

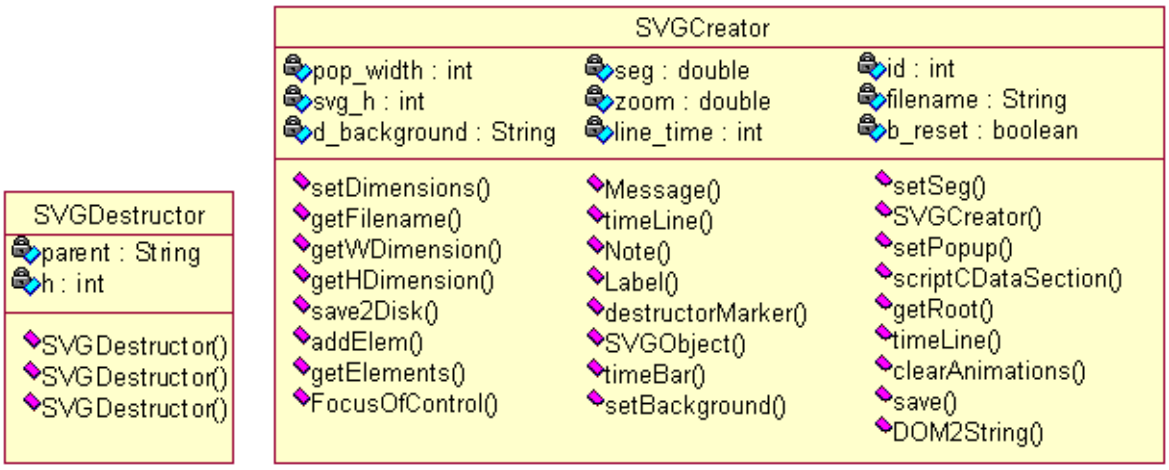


Figura 5.4: Clases SVGDestructor y SVGCreator

`SVGObject`. En la Tabla B.7 del apéndice B se presentan los métodos y atributos de esta clase.

5.1.8. La clase SVGCreator

Esta clase es una de las clases más importantes de la librería SVGDiagram. A la derecha de la Figura 5.4 se muestra la estructura de esta clase. `SVGCreator` es la interfaz o puente entre la representación abstracta de los objetos Java con elementos SVG de un diagrama SVG. `SVGCreator` junto con `SVGDiagram` se encargan de realizar el proceso de generación de diagramas UML a partir de objetos abstractos Java a elementos SVG de documentos SVG. `SVGCreator` no hereda de ninguna otra clase, por lo que `SVGCreator` contiene atributos y métodos propios. `SVGCreator` por su parte se encarga de generar y almacenar elementos SVG en documentos SVG. También se encarga de establecer propiedades gráficas de los diagramas UML generados (por ejemplo el ancho y alto). `SVGCreator` ofrece los métodos necesarios para crear elementos SVG que convierten objetos abstractos Java (instancias de las clases `SVGObject`, `SVGNote`, `SVGLabel`, `SVGFocusOfControl` y `SVGDestructor`) en elementos SVG de un documento SVG. En la Tabla B.9 del apéndice B se presentan los métodos y atributos de esta clase.

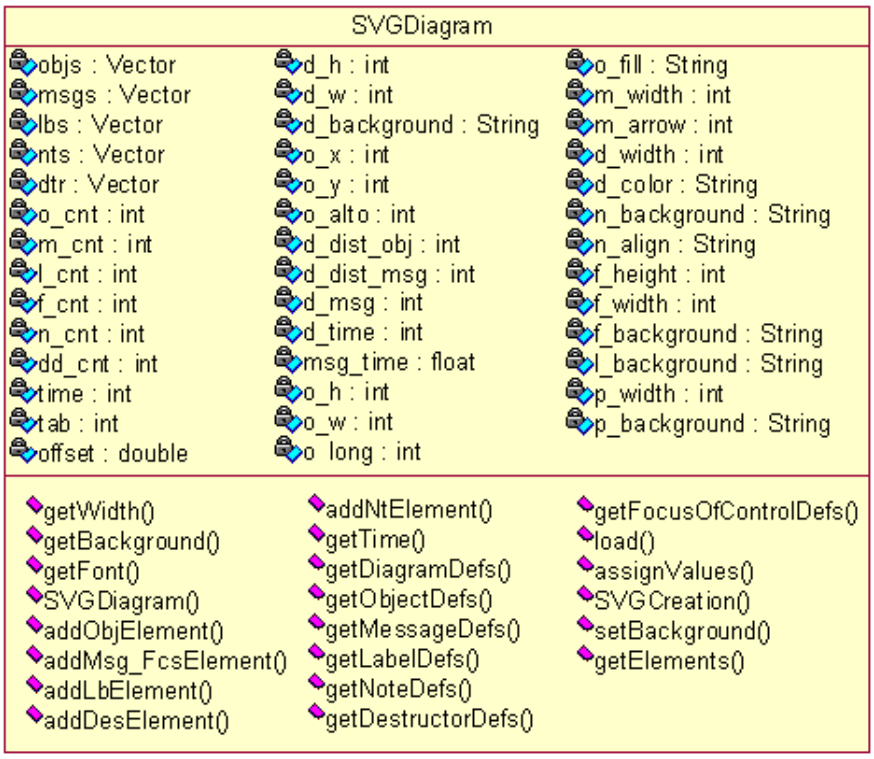


Figura 5.5: Clase SVGDiagram

5.1.9. La clase SVGDiagram

Esta clase se encarga de almacenar todos los objetos abstractos Java de un diagrama UML de secuencia. En la Figura 5.5 se muestra la estructura de esta clase. Los objetos abstractos Java se mantienen en estructuras de datos *listas* para su futura conversión en elementos SVG. Estas *listas* contienen objetos de las clases `SVGObject`, `SVGMsg`, `SVGDestructor`, `SVGLabel`, `SVGNote` y `SVGFocusOfControl`. `SVGDiagram` contiene atributos que se usan para leer de archivos de configuración características gráficas de diagramas UML. Estos archivos de configuración contienen valores y características para generar elementos UML, por ejemplo, el color del fondo del diagrama, el tipo de letra usado en etiquetas UML, la alineación del texto contenido en notas UML, etc. Los métodos de esta clase se utilizan para establecer propiedades de los diagramas UML, añadir nuevos elementos al diagrama, cargar archivos de configuración y por supuesto, generar las representaciones SVG de todos los objetos abstractos Java contenidos en las *listas*. En la Tabla B.8 del apéndice B se presentan los métodos y atributos de esta clase.

5.2. Librería MDL2SVG

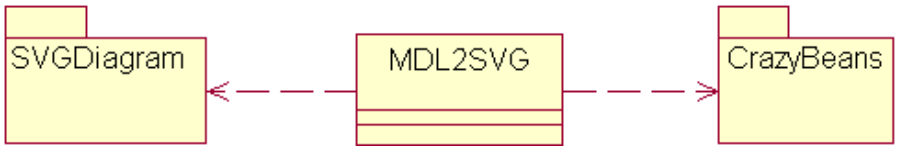


Figura 5.6: Diagrama de clases de la librería MDL2SVG

La librería MDL2SVG tiene la finalidad de generar diagramas UML en documentos SVG a partir de diagramas UML creados con Rational Rose. Estos diagramas se utilizan para monitorear la ejecución de procesos en tiempo real. Se utilizan los diagramas de la herramienta Rational Rose, debido a que los archivos que utiliza para almacenar sus diagramas son archivos textuales. Sin embargo, el formato de estos archivos no es sencillo de entender. Para manipular estos archivos afortunadamente se cuenta con el paquete llamado CrazyBeans, el cual permite manejar archivos de Rational Rose como un conjunto de objetos Java.

En la Figura 5.6 se muestra el diagrama de clases de la librería MDL2SVG. En este diagrama se muestran las relaciones que existen entre clases y otras librerías. Esta librería se integra por una sola clase Java `Mdl2Svg`, la cual se encarga de convertir diagramas de secuencia a partir de archivos de Rational Rose (archivos `.mdl`) en diagramas UML de secuencia de documentos SVG. Para realizar estas conversiones `Mdl2Svg` utiliza a la librería SVGDiagram y al paquete CrazyBeans. La librería SVGDiagram se utiliza para generar diagramas UML de secuencia en documentos SVG. Por otra parte, el paquete CrazyBeans se utiliza para analizar y extraer información de los elementos UML contenidos en los archivos de Rational Rose.

5.2.1. La clase MDL2SVG

En la Figura 5.7 se muestra la estructura de la clase `Mdl2Svg`. Esta clase se utiliza para generar las representaciones de diagramas UML de secuencia de archivos `.mdl` en documentos SVG. Para esto, `Mdl2Svg` utiliza el paquete CrazyBeans para extraer diagramas de archivos `.mdl` y a la librería SVGDiagram para generar las representaciones SVG de

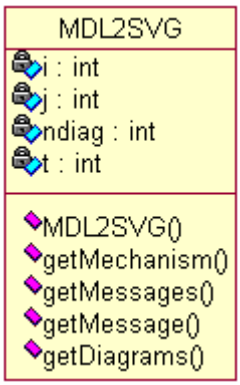


Figura 5.7: La clase MDL2SVG

elementos UML. En la Tabla B.10 del apéndice B se presentan los métodos y atributos de esta clase.

5.3. Librería SVG2XMI

Figura 5.8: Diagrama de clases de la librería SVG2XMI

La librería SVG2XMI tiene la finalidad de serializar los diagramas SVG generados por las librerías MDL2SVG o SVGDiagram en documentos XMI. Esto con la finalidad de que los diagramas SVG se puedan visualizar en varias herramientas de modelado con UML. Actualmente los documentos XMI generados siguen la versión 1.1 de XMI, la cual no es la más reciente, sin embargo, a la fecha varias herramienta solo soportan esta versión de XMI.

En la Figura 5.8 se muestra el diagrama de clases de la librería SVG2XMI. En este diagrama se muestran las relaciones que existen entre clases, paquetes y otras librerías. Esta librería se integra por una sola clase Java `SV2XMI`, la cual se encarga de serializar diagramas SVG en documentos XMI. `SVG2XMI` utiliza el paquete DOM para manejar elementos de documentos XML.

5.3.1. La clase SVG2XMI

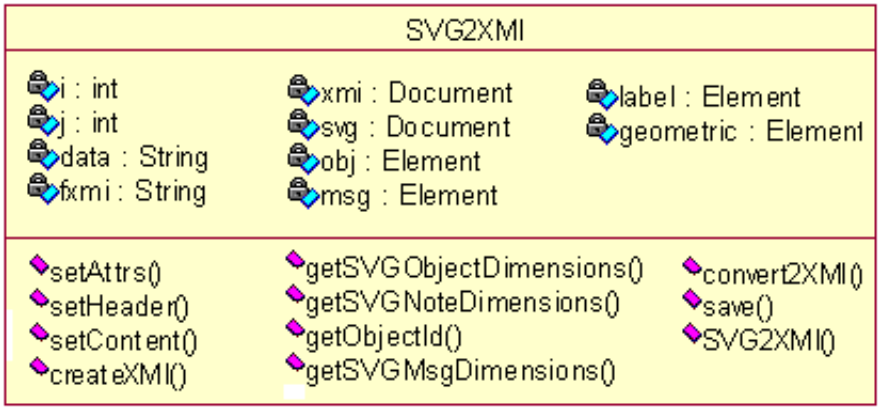


Figura 5.9: La clase SVG2XMI

En la Figura 5.9 se muestra la estructura de la clase `SVG2XMI`. Esta clase se utiliza para representar elementos UML de diagramas SVG en documentos XMI. En la Tabla B.11 del apéndice B se presentan los métodos y atributos de esta clase.

5.4. Librería XMI2SVG

Figura 5.10: Diagrama de clases de la librería XMI2SVG

La finalidad principal de esta librería es generar diagramas SVG a partir de documentos XMI. Esto permite generar diagramas a partir de diagramas UML creados por cualquier herramienta que permita serializar sus diagramas en documentos XMI.

En la Figura 5.10 se muestra el diagrama de clases de la librería XMI2SVG. En este diagrama se muestran las relaciones que existen entre clases, paquetes y otras librerías. Esta librería únicamente la conforma la clase `XMI2SVG`, la cual se encarga de convertir

los modelos UML de documentos XMI en diagramas SVG. `XMI2SVG` utiliza DOM para manejar documentos y elementos XML.

5.4.1. La clase XMI2SVG

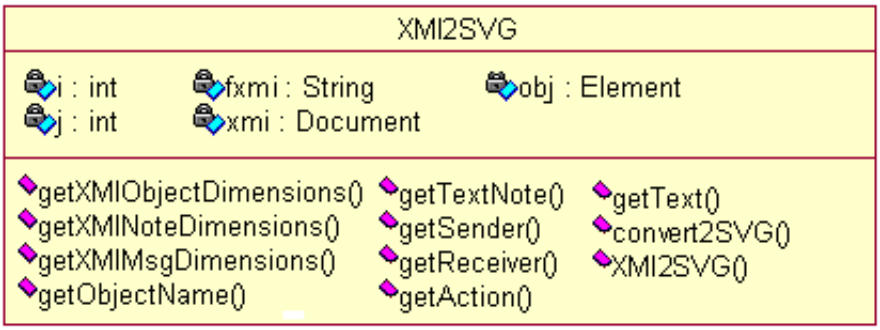


Figura 5.11: La clase XMI2SVG

En la Figura 5.11 se muestra la estructura de la clase `XMI2SVG`. Esta clase se utiliza para convertir serializaciones de diagramas UML de documentos XMI, en diagramas SVG. En la Tabla B.12 del apéndice B se presentan los métodos y atributos de esta clase.

5.5. Librería SOAP2HTML

Figura 5.12: Diagrama de clases de la librería SOAP2HTML

La librería SOAP2HTML tiene la finalidad de crear hojas XML de estilo para presentar mensajes SOAP. Esto permite presentar los mensajes SOAP con un formato de tablas HTML. En la Figura 5.12 se muestra el diagrama de clases de la librería SOAP2HTML. En este diagrama se muestran las relaciones que existen entre clases, paquetes y otras librerías. Esta librería únicamente la conforma la clase `Stylizer`, la cual se encarga de

analizar mensajes SOAP y crear hojas XML de estilo. `Stylizer` utiliza el paquete DOM para manejar elementos de documentos XML.

5.5.1. La clase Stylizer

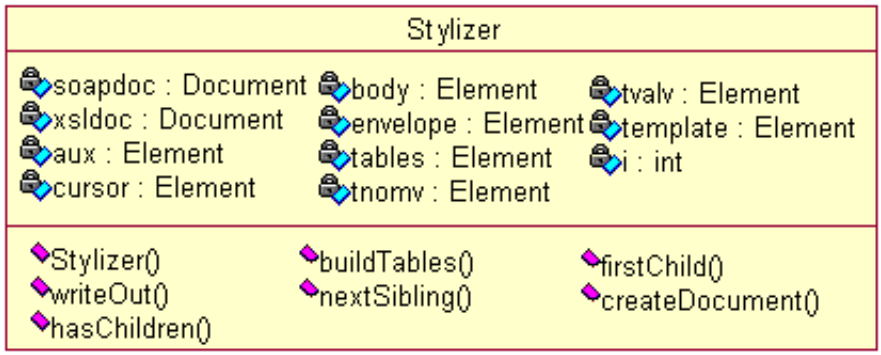


Figura 5.13: La clase Stylizer

En la Figura 5.13 se muestra la estructura de la clase `Stylizer`. Esta clase se utiliza para analizar y crear hojas XML de estilo. Para esto, `Stylizer` modifica las hojas de estilo de acuerdo a los mensajes SOAP que se desean presentar. En la Tabla B.13 del apéndice B se presentan los métodos y atributos de esta clase.

5.6. Detalles de la implementación

En esta sección se presentan brevemente algunos detalles que se presentaron durante el desarrollo e implementación de MS4WS.

5.6.1. Captura de mensajes SOAP

Para monitorear la ejecución de servicios Web, MS4WS captura todos los mensajes SOAP que intercambian servicios Web. Conforme se capturan estos mensajes, MS4WS genera diagramas UML de secuencia y las representaciones SVG de cada mensaje para presentar el monitoreo de la ejecución de servicios Web. Las tareas de capturar mensajes SOAP, crear diagramas UML con SVG y generar representaciones SVG tienen diferencias significativas

en su realización. Por un lado, el intercambio y por consiguiente, la captura de mensajes SOAP es muy rápida, mientras que la creación y generación de representaciones SVG de estos mensajes es más lenta. Las diferencias de tiempo entre la captura del mensaje recibido y la generación de los elementos gráficos correspondientes degradan el desempeño de los servicios Web.

Para asegurar que MS4WS no interfiera con la ejecución de un servicio Web, las tareas de creación de representaciones SVG de los mensajes SOAP se realizan en *scripts* del lenguaje JavaScript, los cuales se ejecutan de manera independiente por cada sesión iniciada por el usuario en los navegadores de Internet. De esta forma, la generación de representaciones SVG se realiza mediante un *buffer*, el cual de igual forma se administra de forma independiente. En la Figura 5.14 se muestra la utilización de este *buffer* para generar representaciones SVG.

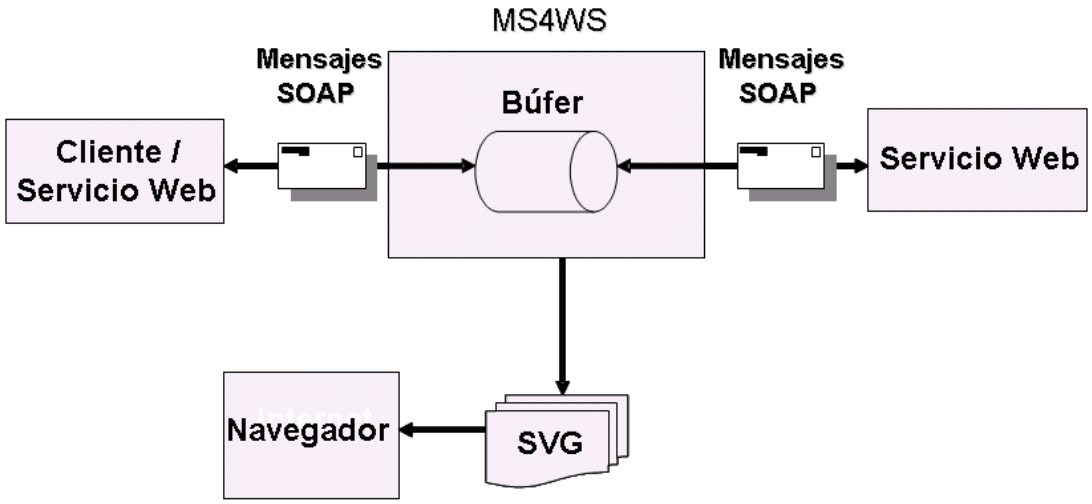


Figura 5.14: Utilización de un *buffer* para generar representaciones SVG

5.6.2. Animaciones en los diagramas con SVG

Un aspecto que se tomó en consideración fue la presentación de los mensajes SOAP en los diagramas. Los mensajes se presentan en los diagramas tan pronto como estos se capturan. Sin embargo, como se mencionó antes, el intercambio de mensajes SOAP es muy rápida, por lo cual presentar inmediatamente estos mensajes en los diagramas es igualmente un

proceso muy veloz, lo que no permite ver con claridad la ejecución de servicios Web. Por esta razón, dentro de las representaciones SVG de mensajes y elementos UML que se generan en documentos SVG se incluyen *etiquetas* SVG de animación, las cuales permiten mantener elementos SVG ocultos por un determinado intervalo de tiempo. La utilización de estas etiquetas de animación mejora la presentación de los diagramas SVG permitiendo ver con mejor claridad la ejecución de servicios Web. En la Figura 5.15 se muestra la secuencia de animación en que aparecen mensajes UML en un diagrama.

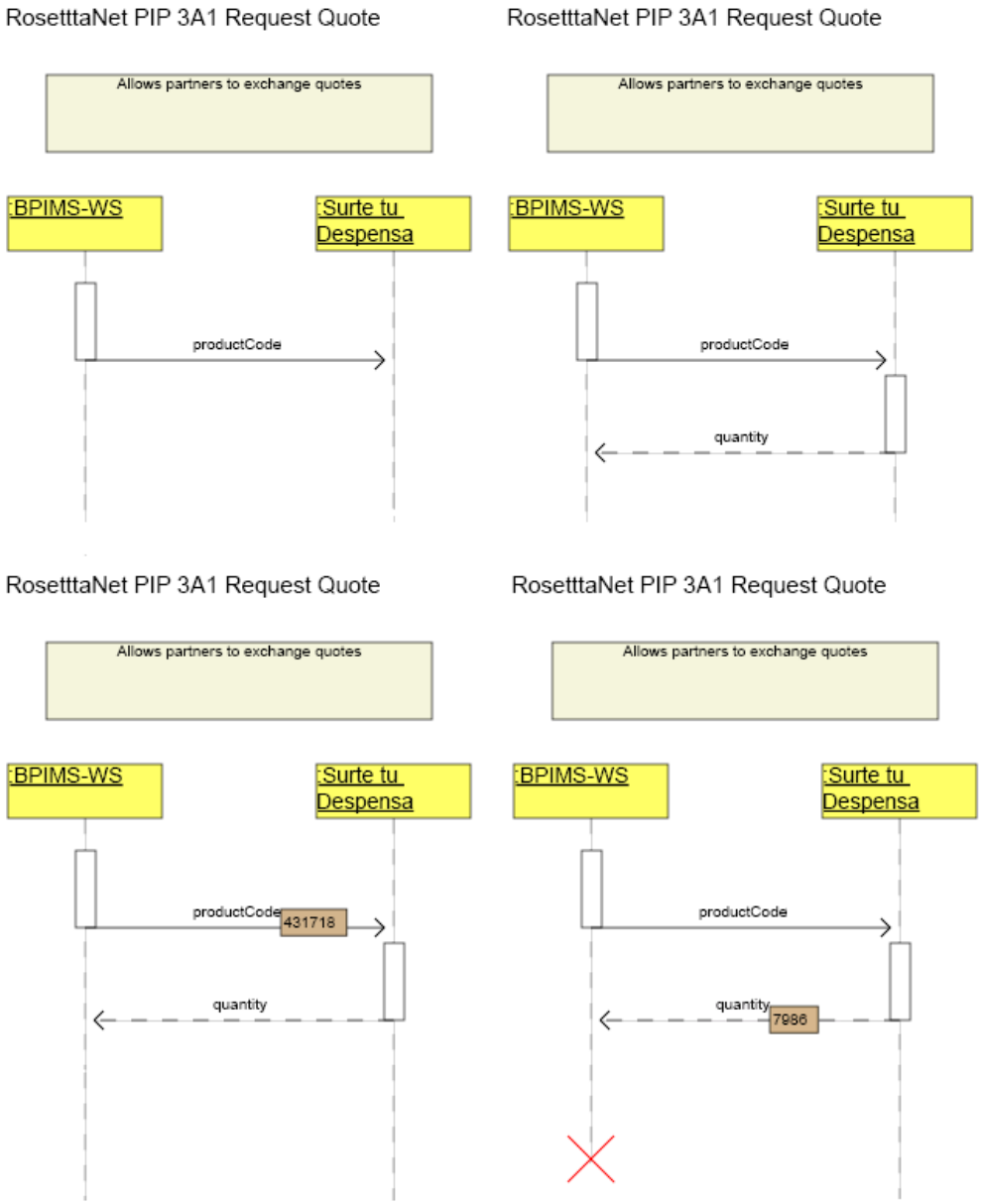


Figura 5.15: Secuencia de mensajes UML en un diagrama

5.7. Conclusiones

En este capitulo se presentaron las implementaciones Java de las librerías de MS4WS. Se presentaron descripciones panorámicas de la utilidad de cada clase. El lenguaje base de desarrollo es Java permitiendo a MS4WS operar en un entorno distribuido como Internet, siendo la portabilidad de MS4WS una característica esencial. Además, otra ventaja de utilizar Java es que se cuenta con bastantes herramientas de soporte para manejar documentos XML. Por otro lado, el uso de documentos SVG para generar y presentar diagramas UML en el navegador de Internet es muy apropiado ya que no maneja los documentos SVG son mejores de manipular. Además el soporte de SVG en varios navegadores de Internet está proliferando. Un aspecto que se considera importante en la implementación fue la utilización del formato propietario de Rational Rose, debido a que es la herramienta de modelado más ampliamente disponible. Se puede decir que MS4WS extiende la funcionalidad de Rational Rose presentándose como un componente externo único de servicios Web para esta herramienta. Finalmente se describieron en este capítulo los problemas que surgen por la diferencia de velocidades en la captura de mensajes SOAP y la producción de los elementos SVG gráficos correspondientes y las medidas utilizadas para disminuir esta diferencia de velocidad en la presentación de los mensajes SOAP en los diagramas SVG.

En el siguiente capítulo se presentan las conclusiones finales de este trabajo de investigación.

Capítulo 6

Conclusiones

Esta tesis presentó el diseño e implementación de MS4WS, un sistema de monitoreo de servicios Web con UML. MS4WS es útil para el diseño y depuración de estos servicios los cuales actualmente son una tecnología muy utilizada en el comercio electrónico. La principal importancia de los servicios Web es la facilidad que ofrecen para la integración entre aplicaciones Web. Otras ventajas de utilizar servicios Web son: la infraestructura con la que operan, la cual se basa en la utilización de lenguajes y protocolos establecidos como estándares de la Web y la propiedad de operar independiente de la plataforma que se utilice. Además, los servicios Web cuentan con el respaldo de importantes empresas y organizaciones de la Web lo cual motivó de gran manera a la realización de esta tesis.

Actualmente se tiene un gran interés por los servicios Web, las expectativas son muy alentadoras, lo cual ha originado en un periodo relativamente corto de tiempo que un gran número de empresas de comercio electrónico y compañías desarrolladoras de *software* adopten esta tecnología. Esta es la principal motivación por la que se decidió realizar este trabajo de investigación. Desarrollar una herramienta que permita monitorear la ejecución de servicios Web la cual tuviera un amplio espacio de implementación.

Por otro lado actualmente existen herramientas que permiten monitorear servicios Web, lo cual ha generado una gran competencia en este ámbito. Sin embargo, los enfoques de estas herramientas son muy especializados lo cual limita su uso únicamente para diseñadores y programadores de estos servicios. En esta tesis se desarrolló una herramienta de servicios Web la cual pueda ser utilizada tanto por diseñadores y programadores,

como para los mismos usuarios de servicios Web. Los resultados de monitorear la ejecución de servicios Web se presentan utilizando diagramas UML de secuencia, los cuales ejemplifican la ejecución de servicios Web y son fáciles de interpretar.

6.1. Contribuciones

El enfoque de este trabajo es el de monitorear y poder interpretar la ejecución de servicios Web con UML. Como principal contribución se tiene a MS4WS la cual es una herramienta capaz de monitorear servicios Web con diagramas UML de secuencia. Actualmente se cuenta con herramientas capaces de realizar esta tarea, sin embargo, todas estas tienen desventajas en comparación con MS4WS.

Las características principales de la herramienta MS4WS son:

- Permite monitorear la ejecución de servicios Web.
- Presenta los resultados de monitorear servicios Web utilizando diagramas UML de secuencia.
- Utiliza lenguajes y protocolos estándares de la Web.
- Opera independiente de la plataforma que se utilice.
- Genera diagramas UML que se pueden visualizar en navegadores de Internet.
- Permite visualizar los diagramas UML en diferentes herramientas que modelan con UML.

Varias de las herramientas existentes permiten monitorear servicios Web y utilizan lenguajes estándares, sin embargo, ninguna de estas utiliza lenguajes estándares de diagramación como UML para presentar sus resultados. Esta es una característica única de MS4WS.

6.2. Trabajo a futuro

Los objetivos que se han alcanzado con MS4WS son muy valiosos, lo que hace de MS4WS una herramienta muy poderosa. Sin embargo, la funcionalidad de MS4WS se puede extender. A continuación se presenta el trabajo a futuro contemplado:

- Manejar más diagramas UML, por ejemplo diagramas de colaboración, de actividades, de estados, etc.
- Crear una interfaz gráfica de modelado con UML.
- Desarrollar un módulo de compilación y visualización de reglas ECA (*Event - Condition - Action*).
- Visualizar el protocolo y las descripciones de servicios Web con diagramas UML de objetos.
- Mantener actualizado el soporte de los lenguajes que se utilizan.
- Desarrollar un módulo de verificación de la ejecución de servicios Web.
- Monitoreo de otros protocolos de comunicación.

Sin duda alguna manejar más diagramas UML sería una mejora muy buena para MS4WS. Por otra parte, además de monitorear servicios Web, una excelente mejora sería incluir un módulo de compilación de reglas ECA que permita a MS4WS administrar la ejecución de servicios Web. Actualmente MS4WS utiliza diagramas UML generados con Rational Rose, lo cual hace necesario contar con un módulo propio de MS4WS que permita crear diagramas UML. También se tiene contemplado presentar el protocolo de comunicación y los servicios Web utilizando diagramas UML de objetos. Actualizar a MS4WS es una necesidad esencial: se necesita mantener actualizado el soporte de los lenguajes que se utilizan debido a que estos sufren actualizaciones en un período de tiempo relativamente corto. Desarrollar un módulo de verificación de servicios Web se refiere, haciendo referencia a transacciones de Bases de Datos, no realizar modificaciones de datos hasta estar segura la perfecta ejecución (*commit* de bases de datos) de los servicios.

Este módulo aportaría una importante funcionalidad para MS4WS. Finalmente, se tiene contemplado desarrollar otros módulos de monitoreo de protocolos de comunicación, por ejemplo HTTP, HTTPS, SMTP, etc., los cuales permitirán, por ejemplo, monitorear comunicaciones remotas y presentar resultados estadísticos del tráfico ocurrente en los protocolos al igual que algunas de las herramientas que se presentaron en el estado del arte.

Apéndice A

Representación de diagramas UML en XMI

A.1. Representación de un diagrama UML de Rational Rose en XMI

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<XMI xmi.version = '1.1' xmlns:UML='href://org.omg/UML/1.3'
timestamp = 'Mon Nov 22 23:04:51 2004' >
 <XMI.header>
  <XMI.documentation>
   <XMI.exporter>Unisys.JCR.2</XMI.exporter>
   <XMI.exporterVersion>1.3.4</XMI.exporterVersion>
  </XMI.documentation>
  <XMI.metamodel xmi.name = 'UML' xmi.version = '1.3'/>
 </XMI.header>
<XMI.content>
<!-- ==================== DiagramaSecuencia    [Model] =====
=============== -->
<UML:Model xmi.id = 'G.0' xmi.uuid = '41A2C1FA0364'
  name = 'DiagramaSecuencia' visibility = 'public' isSpecification
  = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false' >
  <UML:Namespace.ownedElement>
    <!-- ==================== DiagramaSecuencia::(DummiClass)
    [Class] ==================== -->
    <UML:Class xmi.id = 'S.326.2304.39.1' xmi.uuid = '41A2C3E10210'
      name = '(DummiClass)' visibility = 'public' isSpecification
      = 'false' isRoot = 'true' isLeaf = 'true' isAbstract = 'false'
      isActive = 'false' namespace = 'G.0' />
```

```
<!-- ==================== DiagramaSecuencia::Collaboration
[Collaboration] ==================== -->
<UML:Collaboration xmi.id = 'S.326.2304.39.2' name = 'Logical
View-Collaboration' visibility = 'public' isSpecification = 'false'
  isRoot = 'false' isLeaf = 'false' isAbstract = 'false' >
  <UML:Namespace.ownedElement>
    <UML:ClassifierRole xmi.id = 'G.1' xmi.uuid = '41A2C24F03CA'
      name = ':Sales Facilitator' visibility = 'public'
      isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
      isAbstract = 'false'
      base = 'S.326.2304.39.1' message2 = 'G.4' message1 = 'G.5' >
      <UML:ClassifierRole.multiplicity>
        <UML:Multiplicity >
          <UML:Multiplicity.range>
            <UML:MultiplicityRange
              lower = '1' upper = '1' />
          </UML:Multiplicity.range>
        </UML:Multiplicity>
      </UML:ClassifierRole.multiplicity>
    </UML:ClassifierRole>
    <UML:ClassifierRole xmi.id = 'G.3' xmi.uuid = '41A2C26203DC'
      name = ':Supplier' visibility = 'public' isSpecification =
      'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'
      base = 'S.326.2304.39.1' message2 = 'G.5' message1 = 'G.4' >
      <UML:ClassifierRole.multiplicity>
        <UML:Multiplicity >
          <UML:Multiplicity.range>
            <UML:MultiplicityRange
              lower = '1' upper = '1' />
          </UML:Multiplicity.range>
        </UML:Multiplicity>
      </UML:ClassifierRole.multiplicity>
    </UML:ClassifierRole>
    <UML:AssociationRole xmi.id = 'G.2' xmi.uuid = '41A2C26B02D0'
      name = '' visibility = 'public' isSpecification = 'false'
      isRoot = 'false' isLeaf = 'false' isAbstract = 'false' >
      <UML:AssociationRole.multiplicity>
        <UML:Multiplicity >
          <UML:Multiplicity.range>
            <UML:MultiplicityRange
              lower = '1' upper = '1' />
          </UML:Multiplicity.range>
        </UML:Multiplicity>
      </UML:AssociationRole.multiplicity>
      <UML:Association.connection>
        <UML:AssociationEndRole xmi.id = 'XX.22.234.39.5'
          name = '' visibility = 'public' isSpecification
          = 'false' isNavigable = 'false' ordering =
          'unordered' aggregation = 'none' targetScope =
          'instance' changeability = 'changeable'
          type = 'G.3' >
          <UML:AssociationEnd.multiplicity>
            <UML:Multiplicity />
          </UML:AssociationEnd.multiplicity>
```

```
              <UML:AssociationEndRole.collaborationMultiplicity>
                <UML:Multiplicity />
              </UML:AssociationEndRole.collaborationMultiplicity>
            </UML:AssociationEndRole>
            <UML:AssociationEndRole xmi.id = 'XX.22.234.39.6'
              name = '' visibility = 'public' isSpecification =
              'false' isNavigable = 'false' ordering =
              'unordered' aggregation = 'none' targetScope =
              'instance' changeability = 'changeable' type = 'G.1' >
              <UML:AssociationEnd.multiplicity>
                <UML:Multiplicity />
              </UML:AssociationEnd.multiplicity>
              <UML:AssociationEndRole.collaborationMultiplicity>
                <UML:Multiplicity />
              </UML:AssociationEndRole.collaborationMultiplicity>
            </UML:AssociationEndRole>
          </UML:Association.connection>
        </UML:AssociationRole>
      </UML:Namespace.ownedElement>
      <UML:Collaboration.interaction>
        <UML:Interaction xmi.id = 'G.6'
          name = '{Logical View}NewDiagram' visibility = 'public'
          isSpecification = 'false' >
          <UML:Interaction.message>
            <UML:Message xmi.id = 'G.4'
              name = 'Shopping Car' visibility = 'public'
              isSpecification = 'false'
              sender = 'G.1' receiver = 'G.3' message3 = 'G.5'
communicationConnection = 'G.2' action =
'XX.22.234.39.7' />
            <UML:Message xmi.id = 'G.5'
              name = 'Accept/Reject' visibility = 'public'
              isSpecification = 'false' sender = 'G.3' receiver
              = 'G.1' predecessor = 'G.4'
              communicationConnection = 'G.2' action
              = 'XX.22.234.39.8' />
          </UML:Interaction.message>
        </UML:Interaction>
      </UML:Collaboration.interaction>
    </UML:Collaboration>
    <UML:Stereotype xmi.id = 'S.326.2304.39.6'
      name = 'return' visibility = 'public' isSpecification
      = 'false' isRoot = 'false' isLeaf = 'false' isAbstract
      = 'false' icon = '' baseClass = 'Message' extendedElement
      = 'G.5' />
    <UML:UninterpretedAction xmi.id = 'XX.22.234.39.7'
      name = 'Shopping Car' visibility = 'public'
      isSpecification = 'false'
      isAsynchronous = 'false' >
      <UML:Action.recurrence>
        <UML:IterationExpression
          language = '' body = '' />
      </UML:Action.recurrence>
      <UML:Action.target>
```

```
        <UML:ObjectSetExpression
          language = '' body = '' />
      </UML:Action.target>
      <UML:Action.script>
        <UML:ActionExpression
          language = '' body = '' />
      </UML:Action.script>
    </UML:UninterpretedAction>
    <UML:UninterpretedAction xmi.id = 'XX.22.234.39.8'
      name = 'Accept/Reject' visibility = 'public'
      isSpecification = 'false' isAsynchronous = 'false' >
      <UML:Action.recurrence>
        <UML:IterationExpression
          language = '' body = '' />
      </UML:Action.recurrence>
      <UML:Action.target>
        <UML:ObjectSetExpression
          language = '' body = '' />
      </UML:Action.target>
      <UML:Action.script>
        <UML:ActionExpression
          language = '' body = '' />
      </UML:Action.script>
    </UML:UninterpretedAction>
    <UML:Comment xmi.id = 'XX.22.234.39.14'
      name = 'A shopping car contains product items with
      quantities and prices. Shopping cars are often
      assosiated with sales catalogs from wich customers
      select products when "filling car"' visibility = 'public'
      isSpecification = 'false' />
  </UML:Namespace.ownedElement>
</UML:Model>
<UML:TaggedValue xmi.id = 'XX.22.234.39.2'
  tag = 'persistence' value = 'transient'
  modelElement = 'S.326.2304.39.1' />
<UML:TaggedValue xmi.id = 'XX.22.234.39.3'
  tag = 'persistence' value = 'transient'
  modelElement = 'G.1' />
<UML:TaggedValue xmi.id = 'XX.22.234.39.4'
  tag = 'persistence' value = 'transient'
  modelElement = 'G.3' />
<UML:Diagram xmi.id = 'S.326.2304.39.7'
  name = 'Main' toolName = 'Rational Rose 98' diagramType
  = 'ClassDiagram' style = '' owner = 'G.0' />
<UML:Diagram xmi.id = 'S.326.2304.39.8'
  name = 'NewDiagram' toolName = 'Rational Rose 98'
  diagramType = 'SequenceDiagram' style = '' owner = 'G.6' >
  <UML:Diagram.element>
    <UML:DiagramElement xmi.id = 'XX.22.234.39.9'
      geometry = ' 735, 729, 479, 130,' style =
      'FillColor.Blue= 204,FillColor.Green= 255,FillColor.Red
      = 255,FillColor.Transparent=1,Font.Blue= 0,Font.Green=
      0,Font.Red= 0,Font.FaceName=Arial,Font.Size= 12,Font.Bold
      =0,Font.Italic=0,Font.Strikethrough=0,Font.Underline=1,
```

```
      LineColor.Blue= 51,LineColor.Green= 0,LineColor.Red= 153
      ,' subject = 'G.1' />
    <UML:DiagramElement xmi.id = 'XX.22.234.39.10' geometry =
    ' 1404, 717, 352, 130,' style = 'FillColor.Blue= 204,
      FillColor.Green= 255,FillColor.Red= 255,
      FillColor.Transparent=1,Font.Blue= 0,Font.Green= 0,
      Font.Red= 0,Font.FaceName=Arial,Font.Size=12,Font.Bold=0,Font.Italic=0,
      Font.Strikethrough=0,Font.Underline=1,LineColor.Blue= 51,
      LineColor.Green= 0,LineColor.Red= 153,' subject = 'G.3' />
    <UML:DiagramElement xmi.id = 'XX.22.234.39.11' geometry = ' 1069,
    1018, 12, 58,' style = 'Message, SQN= 1,' subject = 'G.4' />
    <UML:DiagramElement xmi.id = 'XX.22.234.39.12' geometry = ' 1070,
    1297, 12, 58,' style = 'Message, SQN= 2,' subject = 'G.5' />
    <UML:DiagramElement xmi.id = 'XX.22.234.39.13' geometry = ' 1059,
    390, 1024, 218,' style = 'Note:Font.Blue = 0, Font.Green = 0,
    Font.Red= 0, Font.FaceName=Arial, Font.Size= 10, Font.Bold=0,
    Font.Italic=0, Font.Strikethrough=0, Font.Underline=0,
    LineColor.Blue= 51,LineColor.Green = 0, LineColor.Red = 153,
    FillColor.Blue= 204,FillColor.Green= 255, FillColor.Red = 255,
    FillColor.Transparent=1,' subject = 'XX.22.234.39.14' />
  </UML:Diagram.element>
</UML:Diagram>
</XMI.content>
</XMI>
```

Apéndice B

Descripciones de las librerías de MS4WS

B.1. Librería SVGDiagram

Clase	`com.cinvestav.www.SVGItem`	
Descripción	Esta clase representa en forma abstracta a cualquier objeto de un diagrama UML de secuencia, estos objetos pueden ser objetos UML, notas UML, etc. Otras clases heredan de `SVGItem`. `SVGItem` únicamente contiene las variables básicas para definir un objeto gráficamente.	
Superclases	No tiene superclases	
Subclases	`SVGDestructor, SVGLabel, SVGFocusOfControl, SVGMsg, SVGNote, SVGObject`	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`private`	`int`	`x`
`private`	`int`	`y`
`private`	`int`	`Okind`
`private`	`String`	`popup`
Los atributos `x` y `y` especifican las coordenadas del objeto en el diagrama. El atributo `Okind` se utiliza para identificar el tipo de objeto UML que representa. El atributo `popup` contiene la leyenda que se muestra junto al cursor del ratón cuando se encuentra encima del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`int`	`getX`
	parámetros	No recibe parámetros
	descripción	Este método regresa la coordenada `x` del objeto.
`public`	`int`	`getY`
	parámetros	No recibe parámetros
	descripción	Este método regresa la coordenada `y` del objeto.

Tabla B.1: Atributos y métodos de la clase SVGItem

Clase	`com.cinvestav.www.SVGObject`	
Descripción	Esta clase representa en forma abstracta a un objeto UML. Contiene las variables necesarias para definir un objeto UML. Cada instancia de esta clase se añade en la lista de objetos UML para posteriormente construir su representación gráfica en un diagrama UML.	
Superclases	`SVGItem`	
Subclases	No tiene subclases	
Asociaciones	`SVGDiagram y SVGCreator`	
Atributos:		
visible	**tipo**	**nombre**
`private`	`int`	`w`
`private`	`int`	`h`
`private`	`int`	`long`
`private`	`String`	`name`
Los atributos de esta clase `w`, `h` y `long` especifican el ancho, alto y largo de un objeto UML respectivamente. El atributo `name` contiene la leyenda que muestra este objeto UML.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`String`	`getName`
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para pedir el nombre del objeto UML. No recibe parámetros. Regresa el nombre del objeto UML como cadena.
`public`	`int`	`SVGObject`

	parámetros	`String` `name`
		`int` `x`
		`int` `y`
		`int` `h`
		`int` `w`
		`cb.petal.Font` `font`
		`String` `fill`
		`String` `popup`

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
`public`	`int`	`SVGObject`
	descripción	Este constructor recibe como parámetros a `name`, el cual contiene el nombre y leyenda del objeto UML. Los parámetros `x` y `y` representan las coordenadas del objeto UML. Los parámetros `h` y `w` representan el alto y ancho del objeto UML. El atributo `font` contiene el tipo de letra y tamaño que se utilizan para presentar el nombre del objeto UML. El parámetro `popup` contiene la leyenda que aparece cuando el cursor del ratón se encuentra encima del `objeto` UML. Este constructor se utiliza cuando no se cuenta con ningún archivo de configuración, por lo que es necesario recibir como parámetros las características necesarias para representar visualmente un objeto UML.
`public`	`String`	`SVGObject`
	parámetros	`String` `name` `String` `popup`
	descripción	Este constructor se utiliza cuando se cuenta con un archivo de configuración. En este caso solo es necesario recibir como parámetros el nombre del objeto UML y la leyenda que se muestra cuando se encuentra el cursor del ratón encima del objeto UML.

Tabla B.2: Atributos y métodos de la clase SVGObject

Clase	`com.cinvestav.www.SVGNote`	
Descripción	Esta clase representa en forma abstracta a una nota UML. Contiene las variables necesarias para definir una nota UML. Cada instancia de esta clase se añade a una lista de notas UML para posteriormente construir su representación gráfica.	
Superclases	`SVGItem`	
Subclases	No tiene subclases	
Asociaciones	`SVGDiagram y SVGCreator`	
Atributos:		
visible	**tipo**	**nombre**
`private`	`int`	`h`
`private`	`int`	`w`
`private`	`text`	`text`
Una nota UML tiene una región que delimita el texto que se presenta en el diagrama, esta región es de forma rectangular. Los atributos `h` y `w` representan el alto y ancho de esta región respectivamente. El atributo `text` almacena el texto que presenta la nota UML.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`int`	`SVGNote`

	parámetros	`String` `text`
		`int` `x`
		`int` `y`
		`int` `h`
		`int` `w`

	descripción	El parámetro `text` contiene el texto que presenta la nota UML. Los parámetros `x` y `y` especifican las coordenadas en el diagrama. Los parámetros `h` y `w` especifican la dimensión rectangular de la región en que se presenta la nota UML.

Tabla B.3: Atributos y métodos de la clase SVGNote

Clase	`com.cinvestav.www.SVGLabel`
Descripción	Esta clase representa en forma abstracta a una etiqueta UML. Contiene las variables necesarias para definir una etiqueta UML. Cada instancia de esta clase se añade en una lista de etiquetas UML para posteriormente construir su representación gráfica en los diagramas.
Superclases	`SVGItem`
Subclases	No tiene subclases
Asociaciones	`cb.petal.StringLiteral y cb.petal.Font`

Atributos:

visible	tipo	nombre
`private`	`int`	`h`
`private`	`int`	`w`
`private`	`String`	`align`
`private`	`StringLiteral`	`text`
`private`	`cb.petal.Font`	`font`

Los atributos `h` y `w` especifican el alto y ancho de la etiqueta UML respectivamente. El atributo `align` contiene la alineación del texto que se usa (centrado, justificado, etc.). El atributo `text` contiene el texto que muestra la etiqueta UML. El atributo `font` especifica el tipo y tamaño de letra que se emplea para presentar el texto de la etiqueta UML.

Constructores y métodos

visible	salida	nombre	
`public`	`void`	`SVGLabel`	
	parámetros	`cb.petal.StringLiteral`	`text`
		`int`	`x`
		`int`	`y`
		`int`	`h`
		`int`	`w`
		`String`	`popup`

Continúa en la página siguiente

<table>
<tr><td colspan="3" align="right">Continúa de la página anterior</td></tr>
<tr><td colspan="3">Constructores y métodos</td></tr>
<tr><td>visible</td><td>salida</td><td>nombre</td></tr>
<tr><td>public</td><td>void</td><td>SVGLabel</td></tr>
<tr><td></td><td>descripción</td><td>Este constructor se utiliza cuando se cuenta con un archivo de configuración. Este archivo contiene el valor de los atributos <code>align</code> y <code>font</code>. Los parámetros <code>x</code> e <code>y</code> contienen las coordenadas del objeto. Los parámetros <code>h</code> y <code>w</code> especifican las dimensiones de la región en que se presenta el objeto. El parámetro <code>popup</code> contiene la leyenda que aparece junto al cursor del ratón cuando se encuentra encima del objeto.</td></tr>
<tr><td>public</td><td>void</td><td>SVGLabel</td></tr>
<tr><td></td><td>parámetros

descripción</td><td><code>cb.petal.StringLiteral text</code>
<code>int x</code>
<code>int y</code>
<code>int h</code>
<code>int w</code>
<code>String popup</code>
<code>String align</code>
Este constructor se utiliza cuando no se cuenta con un archivo de configuración, por lo que es necesario inicializar todos los atributos en este constructor. Los parámetros <code>x</code> e <code>y</code> contienen las coordenadas del objeto. Los parámetros <code>h</code> y <code>w</code> especifican las dimensiones de la región en que se presenta el objeto. El parámetro <code>align</code> contiene la alineación del texto que se presenta y <code>popup</code> contiene la leyenda que aparece junto al cursor del ratón cuando se encuentra encima del objeto.</td></tr>
</table>

Tabla B.4: Atributos y métodos de la clase SVGLabel

Clase	`com.cinvestav.www.SVGFocusOfControl`	
Descripción	Esta clase representa en forma abstracta a un foco de control de un diagrama UML de secuencia. Contiene las variables necesarias para definir un objeto gráficamente. Cada instancia de esta clase se añade a una lista de `FocusOfControls` para posteriormente construir su representación gráfica en los diagramas UML.	
Superclases	`SVGItem`	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`private`	`int`	`h`
`private`	`String`	`parent`
El atributo `parent` contiene el nombre del objeto UML del que forma parte visualmente una instancia de esta clase. El atributo `h` especifica el alto del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`		`FocusOfControl`
	parámetros	`String parent`
	descripción	Este constructor se utiliza cuando se cuenta con un archivo de configuración. En este caso el valor del atributo `h` se encuentra en el archivo de configuración. El parámetro `parent` contiene el nombre del objeto UML del que forma parte visualmente.
`public`		`FocusOfControl`

	parámetros	`int x1`
		`int y`
		`int h`

	descripción	Este constructor se utiliza cuando no se cuenta con un archivo de configuración ni con el objeto UML del que forma parte visualmente. Este constructor recibe las coordenadas `x`, `y` y la altura `h` del objeto.

Tabla B.5: Atributos y métodos de la clase SVGFocusOfControl

Clase	`com.cinvestav.www.SVGMsg`	
Descripción	Esta clase representa en forma abstracta a un mensaje UML. Estos mensajes se forman por una leyenda y una flecha. Contiene las variables necesarias para definir un objeto gráficamente. Cada instancia de esta clase se añade a una lista de mensajes UML para posteriormente construir su representación gráfica.	
Superclases	`SVGItem`	
Subclases	No tiene subclases	
Asociaciones	No tiene Asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`private`	`int`	`x1`
`private`	`String`	`msg`
`private`	`String`	`type`
`private`	`String`	`sender`
`private`	`String`	`receiver`
El atributo `sender` contiene el nombre del objeto UML el cual marca el inicio de la flecha. El atributo `receiver` contiene el nombre del objeto UML el cual marca la terminación de la flecha. El atributo `type` especifica el tipo de mensaje UML, el cual puede ser de petición, de respuesta, etc. El atributo `msg` contiene el texto del mensaje UML. El atributo `x1` contiene la coordenada de terminación de la flecha.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`		`SVGMsg`

	parámetros	`String` `msg`
		`String` `sender`
		`String` `receiver`
		`String` `popup`
		`String` `type`

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
`public`		`SVGMsg`
	descripción	Este constructor se utiliza cuando se cuenta con los nombres de los objetos UML que especifican el inicio y final de la flecha. Los atributos `sender` y `receiver` almacenan estos nombres. El atributo `msg` contiene el texto del mensaje UML. El atributo `popup` contiene la leyenda que aparece junto con el cursor del ratón cuando se encuentra encima del objeto. El atributo `type` especifica el tipo de mensaje UML.
`public`		`SVGMsg`

	parámetros	`int` `x`
		`int` `y`
		`int` `x1`
		`String` `type`
		`String` `msg`

	descripción	Este constructor se utiliza cuando no se cuenta con los objetos de inicio y terminación de la flecha del mensaje. Se especifican las coordenadas `x` y `y` de inicio de la flecha y la coordenada `x1` que especifica en donde termina la flecha. El parámetro `type` especifica el tipo de mensaje UML. El parámetro `msg` contiene el texto del mensaje.

Tabla B.6: Atributos y métodos de la clase SVGMsg

Clase	`com.cinvestav.www.SVGDestructor`	
Descripción	Esta clase representa en forma abstracta a un destructor UML. Contiene las variables necesarias para definir un objeto gráficamente. Cada instancia de esta clase se añade a una lista de destructores UML para posteriormente construir su representación gráfica.	
Superclases	`SVGItem`	
Subclases	No tiene subclases	
Asociaciones	`SVGDiagram y SVGCreator`	
Atributos:		
visible	**tipo**	**nombre**
`private`	`int`	`h`
`private`	`String`	`parent`
El atributo `parent` se utiliza para identificar el objeto al cual hace referencia gráficamente una instancia de esta clase, se almacena el nombre del objeto referencia para poder extraer las coordenadas necesarias para la visualización de una instancia. El atributo `h` define el alto y ancho del destructor UML.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`		`SVGDestructor`
	parámetros	`String parent`
	descripción	Este constructor se utiliza cuando no es necesario especificar coordenadas del objeto. Las coordenadas de un destructor UML se obtienen a partir de un objeto UML de referencia `parent`.
`public`		`SVGDestructor`
	parámetros	`int x` `int y`
	descripción	Este constructor se utiliza cuando no se cuenta con un objeto de referencia. Es necesario contar con las coordenadas del destructor UML, los parámetros `x` y `y` especifican las coordenadas del objeto.

Tabla B.7: Atributos y métodos de la clase SVGDestructor

Clase	`com.cinvestav.www.SVGDiagram`	
Descripción	Esta clase representa en forma abstracta a un diagrama UML de secuencia. Cuenta con las variables necesarias para representar visualmente un diagrama y para almacenar objetos UML abstractos. `SVGDiagram` junto con `SVGCreator` son las clases principales de la librería SVGDiagram.	
Superclases	No tiene superclases	
Subclases	No tiene subclases	
Asociaciones	`cb.petal.Font y SVGCreator`	
Atributos:		
visible	**tipo**	**nombre**
private	Vector	objs
private	Vector	msgs
private	Vector	lbs
private	Vector	nts
private	Vector	dtr
private	int	o_cnt
private	int	m_cnt
private	int	l_cnt
private	int	f_cnt
private	int	n_cnt
private	int	dd_cnt
private	int	d_h
private	int	d_w
private	String	d_background
private	int	o_x
private	int	o_y
private	int	o_height
private	int	d_dist_obj
private	int	d_dist_msg
private	int	d_msg
private	int	d_time
private	float	msg_time
private	int	o_h
private	int	o_w
private	int	o_long
private	String	o_fill

Continúa en la página siguiente

Continúa de la página anterior

Atributos:

visible	tipo	nombre
private	int	m_width
private	int	m_arrow
private	int	d_width
private	String	d_color
private	String	n_background
private	String	n_align
private	int	f_height
private	int	f_width
private	String	f_background
private	String	e_background
private	String	e_font
private	int	p_width
private	String	p_background

Esta clase contiene varios atributos, sin embargo, todos son necesarios. Los atributos `objs`, `msgs`, `lbs`, `nts` y `dtr` son listas las cuales almacenan diferentes elementos de diagramas UML para posteriormente generar su representación SVG de cada uno de ellos. Los atributos `f_cnt`, `l_cnt`, `m_cnt`, `n_cnt`, `o_cnt` y `dd_cnt` son contadores que se utilizan para recorrer las listas de elementos UML. Los atributos `d_h`, `d_w` y `d_background` se utilizan para definir las dimensiones de un diagrama. Los atributos `o_x`, `o_y`, `o_height`, `d_dist_obj`, `d_dist_msg`, `d_msg`, `d_time` y `msg_time` se utilizan para ubicar los elementos UML conforme estos se añaden al diagrama. Los atributos `o_h`, `o_w`, `o_long` y `o_fill` se utilizan para almacenar valores de archivos de configuración de objetos UML. Los atributos `m_width` y `m_arrow` se utilizan para almacenar valores de archivos de configuración de mensajes UML. Los atributos `d_width` y `d_color` se utilizan para almacenar valores de archivos de configuración de destructores UML. Los atributos `n_background` y `n_align` se utilizan para almacenar valores de archivos de configuración de notas UML. Los atributos `f_height` y `f_background` se utilizan para almacenar valores de archivos de configuración de focos de control UML. Los atributos `e_font` y `e_background` se utilizan para almacenar valores de archivo de configuración de etiquetas UML. Los atributos `p_width` y `p_background` se utilizan para almacenar valores de archivos de configuración para definir el ancho y fondo de las leyendas que aparecen junto con el cursor del ratón cuando se encuentra encima de un elemento UML.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
public	int	getWidth
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para obtener el ancho del diagrama.
public	int	getDiagramDefs
	parámetros	org.w3c.dom.Element elm
	descripción	Este método se utiliza para inicializar los atributos d_h, d_w, d_background, o_x, o_y, o_height, d_dist_obj, d_dist_msg, d_msg, d_time y msg_time con valores de un archivo de configuración. El atributo elm es el nodo XML que contiene los valores para estos atributos.
public	void	getObjectDefs
	parámetros	org.w3c.dom.Element elm
	descripción	Este método se utiliza para inicializar los atributos o_h, o_w, o_long y o_fill con valores de un archivo de configuración. El atributo elm es el nodo XML que contiene los valores para estos atributos.
public	void	getPopupDefs
	parámetros	org.w3c.dom.Element elm
	descripción	Este método se utiliza para inicializar los atributos p_width y p_background con valores de un archivo de configuración. El atributo elm es el nodo XML que contiene los valores para estos atributos.
public	void	getMessageDefs
	parámetros	org.w3c.dom.Element elm
	descripción	Este método se utiliza para inicializar los atributos m_width y m_arrow con valores de un archivo de configuración. El atributo elm es el nodo XML que contiene los valores para estos atributos.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
`public`	`void`	`getLabelDefs`
	parámetros	`org.w3c.dom.Element` `elm`
	descripción	Este método se utiliza para inicializar los atributos `e_font` y `e_background` con valores de un archivo de configuración. El atributo `elm` es el nodo XML que contiene los valores para estos atributos.
`public`	`void`	`getNoteDefs`
	parámetros	`org.w3c.dom.Element` `elm`
	descripción	Este método se utiliza para inicializar los atributos `n_background` y `n_align` con valores de un archivo de configuración. El atributo `elm` es el nodo XML que contiene los valores para estos atributos.
`public`	`void`	`getDestructorDefs`
	parámetros	`org.w3c.dom.Element` `elm`
	descripción	Este método se utiliza para inicializar los atributos `d_width` y `d_color` con valores de un archivo de configuración. El atributo `elm` es el nodo XML que contiene los valores para estos atributos.
`public`	`void`	`getFocusOfControlDefs`
	parámetros	`org.w3c.dom.Element` `elm`
	descripción	Este método se utiliza para inicializar los atributos `f_height` y `f_background` con valores de un archivo de configuración. El atributo `elm` es el nodo XML contiene los valores para estos atributos.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos

visible	salida	nombre
`public`	`void`	`assignValues`
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para asignar valores a todos lo elementos que se encuentran en las listas de objetos UML. Para cada objeto se inicializan sus coordenadas las cuales cambian de acuerdo al orden de aparición en las listas de objetos UML.
`public`	`void`	`getElements`
	parámetros	`int` `obj`
	descripción	Este método se utiliza para compartir los objetos que se han creados de acuerdo a un identificador. Cada elemento UML se le asigna un identificador único. El parámetro `obj` regresa el o los elementos con identificadores mayores a `o_obj`.
`public`		`SVGDiagram`

	parámetros	`String` `filename`
		`int` `h`
		`int` `w`

	descripción	El parámetro `filename` se utiliza para especificar el nombre del archivo que se desea crear para almacenar el diagrama. Los atributos `h` y `w` especifican las dimensiones del diagrama.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos

visible	salida	nombre
public	void	addObjElement
	parámetros	SVGObject e
	descripción	Este método se utiliza para añadir objetos UML a la lista objs. El parámetro e es el objeto UML que se desea añadir.
public	void	addMsg_FcsElement
	parámetros	SVGItem e
	descripción	Este método se utiliza para añadir mensajes UML o focos de control UML a la lista msgs. El parámetro e es el elemento que se desea añadir.
public	void	addLbElement
	parámetros	SVGLabel e
	descripción	Este método se utiliza para añadir etiquetas UML a la lista lbs. El parámetro e es el elemento que se desea añadir.
public	void	addDesElement
	parámetros	SVGDestructor e
	descripción	Este método se utiliza para añadir destructores UML a la lista dts. El parámetro e es el elemento que se desea añadir.
public	void	addNtElement
	parámetros	SVGNote e
	descripción	Este método se utiliza para añadir notas UML a la lista nts. El parámetro e es el objeto que se desea añadir.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
`public`	`void`	`load`
	parámetros	`String file`
	descripción	Este método se utiliza para inicializar gran parte de los atributos de esta clase. Se inicializan los atributos `o_h`, `o_w`, `o_long`, `o_font`, `o_fill`, `m_width`, `m_arrow`, `m_font`, `d_width`, `d_color`, `n_background`, `n_align`, `n_font`, `f_width`, `f_background`, `e_background`, `e_font`, `p_width`, `p_background` y `p_font`. El parámetro `file` especifica el nombre del archivo de configuración a cargar.
`public`	`void`	`SVGCreation`
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para crear las representaciones gráficas de todos los elementos que se tienen en las listas de elementos UML. Se crea un documento SVG con las dimensiones `d_h` y `d_w`. Después, para cada elemento se crea su representación SVG y se añade al documento SVG. Una vez que se añaden todos los elementos UML el diagrama se guarda en disco duro.
`public`	`void`	`setBackground`
	parámetros	`String background`
	descripción	Este método se utiliza para establecer el fondo del diagrama. El parámetro `background` contiene el nuevo fondo que se desea aplicar al diagrama.

Tabla B.8: Atributos y métodos de la clase SVGDiagram.

Clase	`com.cinvestav.www.SVGCreator`	
Descripción	Esta clase se encarga de crear cada una de las representaciones gráficas de los elementos UML de los diagramas UML de secuencia. También almacena documentos SVG en disco duro.	
Superclases	No tiene superclases	
Subclases	No tiene subclases	
Asociaciones	`SVGDiagram y DOM`	
Atributos:		
visible	**tipo**	**nombre**
`private`	`int`	`popup_width`
`private`	`int`	`svg_h`
`private`	`String`	`d_background`
`private`	`double`	`zoom`
`private`	`int`	`id`
`private`	`String`	`filename`
`private`	`boolean`	`b_reset`
El atributo `popup_width` se utiliza para limitar el ancho que se utiliza para presentar las leyendas que aparecen sobre los objetos. El atributo `svg_h` se utiliza para establecer la altura del diagrama. El atributo `d_background` se utiliza para establecer el fondo del diagrama. El atributo `zoom` se utiliza para definir la escala en que se desean crear los elementos UML en el diagrama. El atributo `id` se utiliza como identificador de elementos UML. El atributo `filename` contiene el nombre del archivo en el que se desea almacenar el documento SVG. El atributo `b_reset` en una bandera que se utiliza para iniciar un nuevo diagrama.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`void`	`setDimensions`
	parámetros	`int h` `int w`
	descripción	Este método se utiliza para establecer el ancho y alto del diagrama.
`public`	`void`	`getFilename`
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para pedir el nombre del archivo en que se almacenará el diagrama.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
public	int	getWDimension
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para pedir el ancho del diagrama.
public	int	getHDimension
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para pedir el alto del diagrama.
public	Element	save2Disk
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para almacenar el diagrama en disco duro. Regresa una etiqueta la cual contiene el nombre del archivo que se desea crear para almacenar el diagrama.
public	void	addElement
	parámetros	org.w3c.dom.Element elm
	descripción	Este método se utiliza para añadir elementos al documento SVG. El parámetro elm es el elemento que se desea añadir.
public	Element	getElements
	parámetros	int n_obj
	descripción	Este método se utiliza para obtener elementos de acuerdo a su identificador. Regresa todos los elementos con identificadores mayores o iguales a n_obj.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
public	void	FocusOfControl
	parámetros	int x1 int y int h double t String background
	descripción	Este método se utiliza para crear y añadir elementos focos de control UML en el documento SVG. Se recibe el fondo, coordenadas, alto y tiempo de presentación del objeto.
public	Element	timeLine
	parámetros	int x int y int h
	descripción	Este método se utiliza para crear y añadir líneas de tiempo en el documento SVG. Se reciben las coordenadas y alto del elemento.
public	void	destructorMarker
	parámetros	int x int y double t String color
	descripción	Este método se utiliza para crear y añadir destructores UML en el documento SVG. Se reciben las coordenadas, color y tiempo de presentación del elemento.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
public	void	setBackground
	parámetros	String background
	descripción	Este método se utiliza para establecer el fondo del diagrama. El parámetro background contiene el nuevo fondo para el diagrama.
public		SVGCreator
	parámetros	String file double zoom
	descripción	Este constructor crea el documento SVG que se utiliza para almacenar los elementos UML. El parámetro file contiene el nombre del archivo en que se desea almacenar el documento SVG. El atributo zoom especifica la escala de trabajo para crear los elementos UML.
public	Element	getRoot
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para pedir la etiqueta principal del documento SVG.
public	void	timeLine2
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para alargar las líneas de tiempo de objetos UML tomando en cuenta las dimensiones del diagrama.
public	void	clearAnimations
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para eliminar todas las etiquetas de animación cuando estas ya no se requieren.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
public	String	SVG2String
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para convertir el documento SVG en cadena.
public	String	DOM2String
	parámetros	org.w3c.dom.Document doc
	descripción	Este método se utiliza para convertir objetos de tipo Element en cadenas.
public	void	Message
	parámetros	int x int y int x1 String text String msgkind cb.petal.Font font double t String popup
	descripción	Este método se utiliza para crear y añadir representaciones SVG de mensajes UML. Se reciben las coordenadas iniciales y finales del mensaje, la fuente, el tipo de mensaje, el tiempo de aparición y la leyenda de cada mensaje UML.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
public	void	Note
	parámetros	int x int y int h int w String text String fill String align cb.petal.Font font
	descripción	Este método se utiliza para crear representaciones de notas UML. Se reciben las coordenadas, el texto, la fuente, las dimensiones, la alineación y el color de fondo de cada nota UML.
public	void	Label
	parámetros	cb.petal.StringLiteral lb int x int y int h int w String popup String align cb.petal.Font font
	descripción	Este método se utiliza para crear etiquetas UML. Se reciben las coordenadas, la fuente, la alineación del texto, las dimensiones y la leyenda de las etiquetas.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
public	void	SVGObject
	parámetros	int x1 int y1 int h int w int long String popup String name String fill cb.petal.Font font
	descripción	Este método se utiliza para crear objetos UML. Se reciben las coordenadas, el nombre del objeto, la fuente, el alto, las dimensiones, el fondo y la leyenda de cada objeto UML.
public	void	setPopup
	parámetros	int width String background cb.petal.Font font
	descripción	Este método se utiliza para establecer el ancho, fondo y fuente de las leyendas de todos los elementos UML.

Tabla B.9: Atributos y métodos de la clase SVGCreator

B.2. Librería MDL2SVG

Clase	`com.cinvestav.www.MDL2SVG`	
Descripción	Esta clase se utiliza para convertir diagramas de archivos `.mdl` en diagramas UML en documentos SVG. Únicamente se convierten los diagramas UML de secuencia.	
Superclases	No tiene superclases	
Subclases	No tiene subclases	
Asociaciones	`SVGDiagram, CrazyBeans` y `DOM`	
Atributos:		
visible	**tipo**	**nombre**
`private`	`int`	`i`
`private`	`int`	`j`
`private`	`int`	`nDiagram`
`private`	`int`	`t`
Los atributos `i` y `j` se utilizan como contadores. El atributo `nDiagram` se utiliza como contador de diagramas UML. El atributo `t` especifica los tiempos de animación con que se presentan los elementos UML.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`		`MDL2SVG`
	parámetros	`String filename`
	descripción	Este constructor recibe como parámetro el nombre del archivo `.mdl` que se desea analizar. Dentro de este mismo constructor se realizan las conversiones de elementos UML a representaciones SVG de estos mismos
`public`	`Mechanism`	`getMechanism`
	parámetros	`InteractionDiagram d`
	descripción	Este método obtiene el mecanismo del diagrama `d`
`public`	`Message`	`getMessage`
	parámetros	`String name` `List messages`
	descripción	Este método obtiene un mensaje UML que contiene a `name` dentro de una lista de mensajes UML

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
`public`	`List`	`getMessages`
	parámetros	`InteractionDiagram d`
	descripción	Este método obtiene todos los mensajes UML del diagrama de interacción `d`
`public`	`void`	`getDiagrams`
	parámetros	`cb.petal.List diagram`
	descripción	Este método encuentra diagramas UML de secuencia contenidos en la lista `diagram`. También genera las representaciones SVG de cada diagrama encontrado. Cada diagrama se almacena en documentos SVG diferentes

Tabla B.10: Atributos y métodos de la clase MDL2SVG

B.3. Librería SVG2XMI

Clase	`com.cinvestav.www.SVG2XMI`	
Descripción	Esta clase se utiliza para serializar diagramas SVG generados por MS4WS en documentos escritos en XMI.	
Superclases	No tiene superclases	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`private`	`String`	`data`
`private`	`String`	`fxmi`
`private`	`Document`	`svg`
`private`	`Document`	`xmi`
`private`	`Element`	`obj`
`private`	`Element`	`msg`
`private`	`Element`	`note`
`private`	`Element`	`geometric`
`private`	`int`	`i`
`private`	`int`	`j`
El atributo `data` contiene los elementos XML del diagrama SVG en cadena. El atributo `fxmi` especifica el nombre del archivo en el que se almacena el documento XMI creado. El atributo `svg` se utiliza para representar con DOM los elementos XML de `data`. El atributo `xmi` se utiliza para crear elementos XML de documentos escritos en XMI. Los atributos `obj`, `msg` y `note` se utilizan para serializar elementos UML en documentos XMI. El atributo `geometric` se utiliza para especificar los valores geométricos de elementos UML. Los atributos `i` y `j` se utilizan como contadores.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`		`SVG2XMI`
	parámetros	`String` `data`
	descripción	Este constructor inicializa instancias de esta clase. El parámetro `data` contiene los elementos XML del diagrama SVG en cadena.
`public`	`void`	`setHeader`
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para añadir la etiqueta `<XMI.header>` de un documento XMI.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
public	void	setAttrs
	parámetros	No recibe parámetros
	descripción	Este método se establece los atributos de la etiqueta XMI principal, por ejemplo la fecha de creación.
public	void	setContent
	parámetros	String name
	descripción	Este método se utiliza para añadir elementos XMI que describen diagramas UML de secuencia. Estos elementos describen elementos UML, los cuales se utilizan como plantillas para crear nuevos elementos. El parámetro name establece el nombre del diagrama UML a serializar.
public	void	createXMI
	parámetros	String xmifile
	descripción	Este método se utiliza para inicializar un documento escrito en XMI. En este documento se añaden las serializaciones de los elementos UML del diagrama SVG.
public	String	getObjectId
	parámetros	int x
	descripción	Este método obtiene el identificador XMI de un elemento XMI de acuerdo al valor geométrico x.
public	String	getSVGObjectDimensions
	parámetros	Element elm
	descripción	Este método obtiene las dimensiones geométricas (ancho, alto, largo, etc.) de un objeto UML. El parámetro elm es la representación SVG del objeto UML. Estas dimensiones se obtienen en una cadena separadas por comas. XMI utiliza este formato para especificar valores geométricos.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
`public`	`String`	`getSVGNoteDimensions`
	parámetros	`Element elm`
	descripción	Este método obtiene las dimensiones geométricas (ancho, alto, largo, etc.) de una nota UML. El parámetro `elm` es la representación SVG de la nota UML. Estas dimensiones se obtienen en una cadena separadas por comas. XMI utiliza este formato para especificar valores geométricos.
`public`	`String`	`getSVGMsgDimensions`
	parámetros	`Element elm`
	descripción	Este método obtiene las dimensiones geométricas (ancho, alto, largo, etc.) de un mensaje UML. El parámetro `elm` es la representación SVG del mensaje UML. Estas dimensiones se obtienen en una cadena separadas por comas. XMI utiliza este formato para especificar valores geométricos.
`public`	`void`	`convert2XMI`
	parámetros	No recibe parámetros
	descripción	Este método genera todas las serializaciones de los elementos UML de diagramas SVG. Estas serializaciones se almacenan en un nuevo documento XMI.
`public`	`void`	`SVGCreation`
	parámetros	No recibe parámetros
	descripción	Este método guarda en disco duro todas las serializaciones creadas en el documento escrito en XMI.

Tabla B.11: Atributos y métodos de la clase SVG2XMI

B.4. Librería XMI2SVG

Clase	`com.cinvestav.www.XMI2SVG`	
Descripción	Esta clase se utiliza para generar diagramas UML con SVG a partir de documentos XMI.	
Superclases	No tiene superclases	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`private`	`String`	`fxmi`
`private`	`Document`	`xmi`
`private`	`Element`	`obj`
`private`	`int`	`i`
`private`	`int`	`j`
El atributo `fxmi` especifica el nombre del documento XMI. El atributo `xmi` se utiliza para manipular los elementos XML del documento XMI. El atributo `obj` se utiliza para crear nuevos elementos SVG. Los atributos `i` y `j` se utilizan como contadores.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`		`XMI2SVG`
	parámetros	`String filename`
	descripción	Este constructor inicializa instancias de esta clase. El parámetro `filename` especifica el nombre del documento XMI.
`public`	`void`	`convert2SVG`
	parámetros	No recibe parámetros
	descripción	Este método genera las representaciones SVG de las serializaciones de elementos UML contenidas en documentos XMI. Estas representaciones SVG se añaden a los diagramas SVG.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
`public`	`String`	`getXMIObjectDimensions`
	parámetros	`Element elm`
	descripción	Este método se utiliza para obtener las dimensiones geométricas de un objeto UML serializado. El parámetro `elm` contiene la etiqueta XML del objeto UML. Con estas dimensiones se generan representaciones SVG que se añaden a los diagramas SVG.
`public`	`String`	`getXMIMsgDimensions`
	parámetros	`Element elm`
	descripción	Este método se utiliza para obtener las dimensiones geométricas de un mensaje UML serializado. El parámetro `elm` contiene la etiqueta XML del mensaje UML. Con estas dimensiones se generan representaciones SVG que se añaden en los diagramas SVG.
`public`	`String`	`getXMINoteDimensions`
	parámetros	`Element elm`
	descripción	Este método se utiliza para obtener las dimensiones geométricas de una nota UML serializada. El parámetro `elm` contiene la etiqueta XML de la nota UML. Con estas dimensiones se generan representaciones SVG que se añaden en los diagramas SVG.
`public`	`String`	`getText`
	parámetros	`String id`
	descripción	Este método se utiliza para obtener el texto de un mensaje UML serializado. El parámetro `id` especifica el identificador del mensaje UML del cual se desea obtener su texto.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
public	String	getObjectName
	parámetros	String id
	descripción	Este método se utiliza para obtener el nombre de un objeto UML serializado. El parámetro id especifica el identificador del objeto UML del cual se desea obtener su nombre.
public	String	getNoteText
	parámetros	String id
	descripción	Este método se utiliza para obtener el texto de una nota UML. El parámetro id especifica el identificador de la nota UML de la cual se desea obtener su texto.
public	String	getAction
	parámetros	String id
	descripción	Este método se utiliza para obtener el tipo de mensaje UML serializado, por ejemplo, mensaje simple, de respuesta, etc. La representación SVG del mensaje UML se genera igualmente con el mismo tipo. El parámetro id especifica el identificador del mensaje UML de la cual se desea obtener su tipo.
public	String	getSender
	parámetros	String id
	descripción	Este método se utiliza para obtener el nombre del objeto UML emisor del mensaje UML con identificador id.
public	String	getReceiver
	parámetros	String id
	descripción	Este método se utiliza para obtener el nombre del objeto UML receptor del mensaje UML con identificador id.

Tabla B.12: Atributos y métodos de la clase XMI2SVG

B.5. Librería SOAP2HTML

Clase	`com.cinvestav.www.Stylizer`	
Descripción	Esta clase crea hojas XML de estilo diseñadas para presentar mensajes SOAP con un formato de tablas HTML.	
Superclases	No tiene superclases	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`private`	`Document`	`soapdoc`
`private`	`Document`	`xsldoc`
`private`	`Element`	`aux`
`private`	`Element`	`cursor`
`private`	`Element`	`body`
`private`	`Element`	`envelope`
`private`	`Element`	`tables`
`private`	`Element`	`tnomv`
`private`	`Element`	`tvalv`
`private`	`Element`	`template`
`private`	`int`	`i`
Los atributos `soapdoc` y `xsldoc` se utilizan para manejar mensajes SOAP y de una tabla XSL respectivamente. Los atributos `aux` y `cursor` se utilizan para modificar elementos XML de plantillas XSL. Los atributos `envelope` y `body` se utilizan para manipular elementos XML de mensajes SOAP. Los atributos `tables`, `tnomv` y `tvalv` se utilizan para crear o modificar tablas de plantillas XSL. El atributo `template` se utiliza para añadir etiquetas `<template>` a las plantillas XSL. El atributo `i` se utiliza como contador.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`Element`	`firstChild`
	parámetros	`Element elm`
	descripción	Este método se utiliza para obtener el primer nodo hijo de `elm` sin contar nodos texto.
`public`	`Element`	`nextSibling`
	parámetros	`Element elm`
	descripción	Este método se utiliza para obtener el siguiente nodo hermano de `elm` sin contar nodos texto.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos

visible	salida	nombre
public		Stylizer
	parámetros	String type
	descripción	Este constructor inicializa objetos de esta clase. El parámetro `type` sirve para identificar el tipo de hoja XML de estilo que se desea (petición o respuesta).
public	void	createXSL
	parámetros	String data String xslFile String path
	descripción	Este método se utiliza para generar hojas de estilo. El parámetro `data` contiene el mensaje SOAP en cadena. El parámetro `xslFile` contiene el nombre de la plantilla XSL a utilizar. El atributo `path` especifica la ruta en la que se desea guardar la hoja de estilo generada.
public	boolean	hasChildren
	parámetros	Element elm
	descripción	Este método se utiliza para verificar si el elemento `elm` tiene nodos hijo sin contar nodos texto. Si `elm` tiene hijos que no son de tipo nodo texto este método regresa *true*, *false* en otro caso.
public	void	writeOut
	parámetros	String filename
	descripción	Este método se utiliza para guardar en disco duro las hojas XML de estilo generadas. El parámetro `filename` especifica el nombre del archivo en el que se desea guardar.

Continúa en la página siguiente

Continúa de la página anterior

Constructores y métodos		
visible	**salida**	**nombre**
`public`	`void`	`buildTables`
	parámetros	`Element elm` `String name`
	descripción	Este método se utiliza para analizar la estructura de mensajes SOAP y para modificar plantillas XSL de acuerdo a estas estructuras. El atributo `elm` contiene la estructura del mensaje SOAP. El atributo `name` se utiliza para relacionar los elementos de mensajes SOAP con los elementos XML de hojas XML de estilo.

Tabla B.13: Atributos y métodos de la clase Stylizer

Apéndice C

Paquete CrazyBeans

C.1. Implementaciones de las clases de CrazyBeans

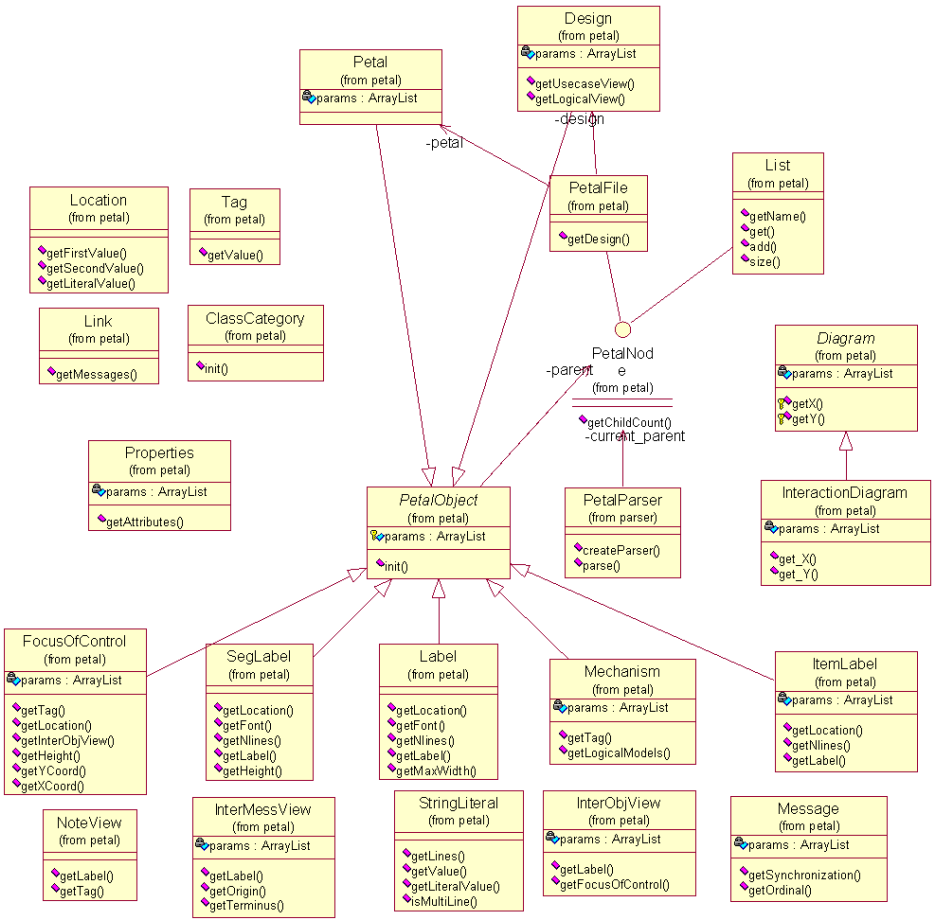


Figura C.1: Diagrama de clases del paquete CrazyBeans

En la Figura C.1 se muestra el diagrama de clases del paquete CrazyBeans. En este diagrama se muestran las relaciones que existen entre clases. Debido a que CrazyBeans es un paquete externo y muy extenso, no se describen ni de presentan todas las clases ni los parámetros ni métodos de estas mismas, únicamente las que se utilizan con MS4WS.

En este paquete las clases `PetalFile`, `Petal` y `PetalParser` junto con la interfaz `PetalNode` se utilizan para manejar archivos `.mdl`. Las clases `Design`, `Diagram`, `InteractionDiagram` y `ClassCategory` se utilizan para acceder a las secciones en las que se encuentran los diagramas UML. La clase `List` se utiliza para listar distintos objetos. Las clase `PetalObject` se utiliza para representar distintos tipos de objetos. Las clases `Properties`, `Tag`, `Location`, `StringLiteral`, `SegLabel`, `Label` y `Mechanism` describen secciones de otros objetos. Las clases `FocusOfControl`, `ItemLabel`, `NoteView`, `InterObjView` e `InterMessView` representan objetos gráficos de un diagrama UML. Las clases `Message` y `Link` contienen información del mecanismo de un diagrama UML de secuencia.

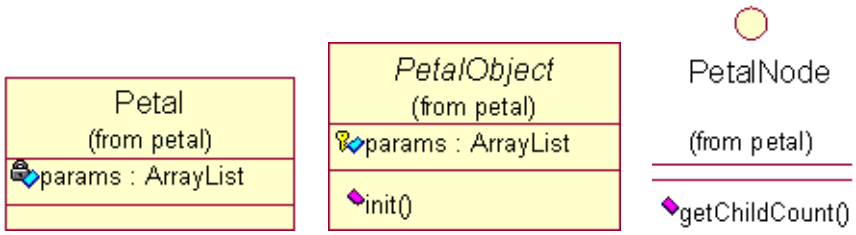


Figura C.2: Clases Petal, PetalObject y PetalNode

C.1.1. La clase Petal

A la izquierda de la Figura C.2 se muestra la estructura de la clase `Petal`. Esta clase se utiliza para representar elementos de archivos `.mdl`, por ejemplo, diagramas UML, objetos UML, mensajes UML, etc. En la Tabla C.1 se presentan los métodos y atributos de esta clase.

C.1.2. La clase PetalObject

En el centro de la Figura C.2 se muestra la estructura de la clase `PetalObject`. Esta clase se utiliza para representar cualquier objeto gráfico de cualquier diagrama UML. En la Tabla C.2 se presentan los métodos y atributos de esta clase.

C.1.3. La interfase PetalNode

A la derecha de la Figura C.2 se muestra la estructura de la interfase `PetalNode`. Esta interfase se utiliza para analizar archivos `.mdl`. En la Tabla C.3 se presentan los métodos y atributos de esta interfase.

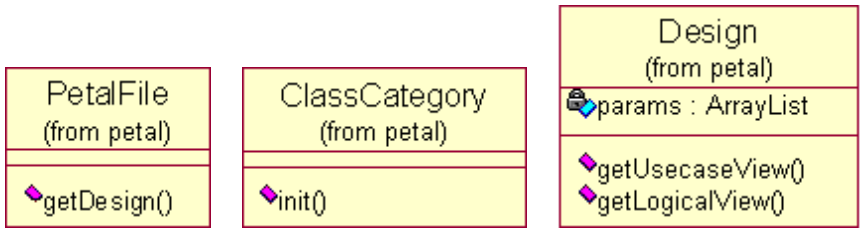


Figura C.3: Clases PetalFile, ClassCategory y Design

C.1.4. La clase PetalFile

A la izquierda de la Figura C.3 se muestra la estructura de la clase `PetalFile`. Esta clase se utiliza para abrir y analizar archivos `.mdl`. La clase `PetalFile` genera la representación en memoria de un archivo `.mdl`. En la Tabla C.4 se presentan los métodos y atributos de esta clase.

C.1.5. La clase ClassCategory

En el centro de la Figura C.3 se muestra la estructura de la clase `ClassCategory`. Esta clase se utiliza para clasificar a los diagramas UML en dos categorías, la categoría de vista lógica y la de casos de uso. En la Tabla C.5 se presentan los métodos y atributos de esta clase.

C.1.6. La clase Design

A la derecha de la Figura C.3 se muestra la estructura de la clase `Design`. Esta clase se utiliza para acceder al diseño de un archivo `.mdl`. Este diseño se divide en tres categorías, la categoría de casos de uso, la de vista lógica y la de componentes. Únicamente se almacenan diagramas UML en las primeras dos categorías, por lo que los métodos `getUseCaseCategory` y `getLogicalView` se utilizan para acceder a ellas. En la Tabla C.6 se presentan los métodos y atributos de esta clase.

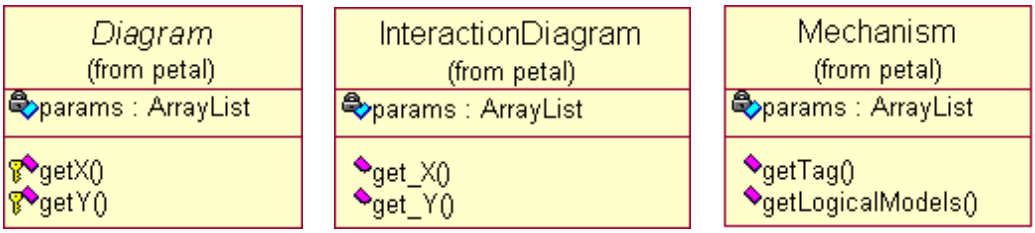


Figura C.4: Clases Diagram, InteractionDiagram y Mechanism

C.1.7. La clase Diagram

A la izquierda de la Figura C.4 se muestra la estructura de la clase `Diagram`. Esta clase se utiliza para representar a cualquier tipo de diagrama UML. En esta clase se definen métodos que permiten dar formato a diagramas UML. En la Tabla C.7 se presentan los métodos y atributos de esta clase.

C.1.8. La clase InteractionDiagram

En el centro de la Figura C.4 se muestra la estructura de la clase `InteractionDiagram`. Esta clase se utiliza para generar diagramas UML de secuencia en documentos SVG. Se utilizan los métodos para acceder a todos los elementos UML del objeto y generar sus representaciones SVG. En la Tabla C.8 se presentan los métodos y atributos de esta clase.

C.1.9. La clase Mechanism

A la derecha de la Figura C.4 se muestra la estructura de la clase `Mechanism`. Esta clase se utiliza para acceder al mecanismo de un diagrama UML de secuencia. Dentro del

mecanismo de un diagrama se encuentran propiedades necesarias para generar un mensaje UML, por ejemplo, el tipo de mensaje, es decir, mensajes simples, mensajes de respuesta, etc. En la Tabla C.9 se presentan los métodos y atributos de esta clase.

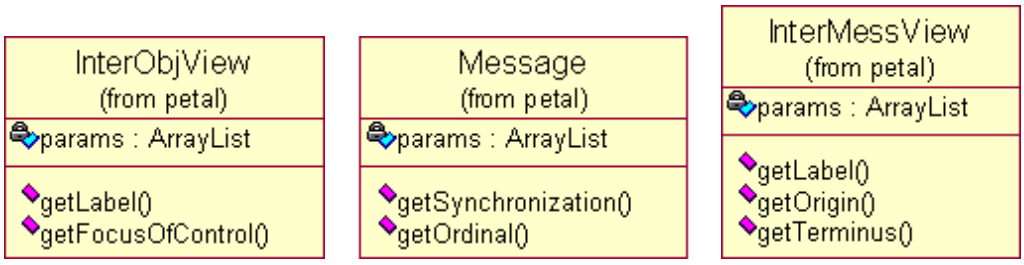


Figura C.5: Clases InterObjView, Message e InterMessView

C.1.10. La clase InterObjView

A la izquierda de la Figura C.5 se muestra la estructura de la clase `InterObjView`. Esta clase se utiliza para generar elementos SVG que representan a objetos UML. Se utilizan los métodos para acceder a las propiedades del objeto y generar su representación SVG. En la Tabla C.10 se presentan los métodos y atributos de esta clase.

C.1.11. La clase Message

En el centro de la Figura C.5 se muestra la estructura de la clase `Message`. Esta clase no representa por si sola a un mensaje UML, únicamente contiene información del mecanismo individual de un mensaje UML. Dentro de este mecanismo se tienen propiedades necesarias para generar mensajes UML en documentos SVG, por ejemplo, esta clase contiene el tipo de mensaje UML que se representa. En la Tabla C.11 se presentan los métodos y atributos de esta clase.

C.1.12. La clase InterMessView

A la derecha de la Figura C.5 se muestra la estructura de la clase `InterMessView`. Esta clase se utiliza para generar un elemento SVG que representa a un mensaje UML. Se utilizan los métodos para acceder a las propiedades del objeto y generar su representación

en un documento SVG. En la Tabla C.12 se presentan los métodos y atributos de esta clase.

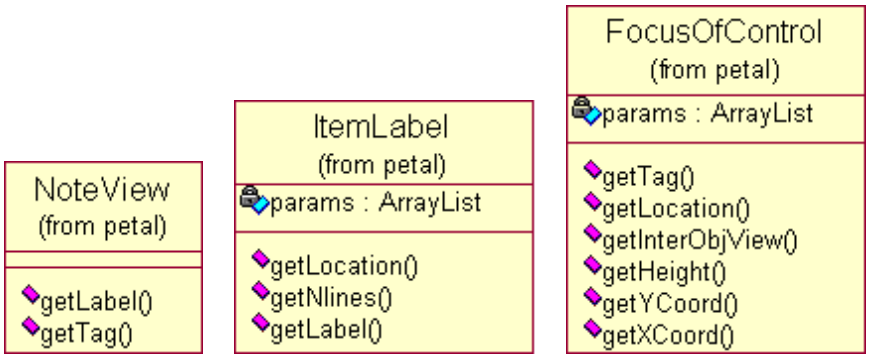


Figura C.6: Clases NoteView, ItemLabel y FocusOfControl

C.1.13. La clase NoteView

A la izquierda de la Figura C.6 se muestra la estructura de la clase `NoteView`. Esta clase se utiliza para generar un elemento SVG que representa a una nota UML. Se utilizan los métodos para acceder a las propiedades del objeto y generar su representación SVG. En la Tabla C.13 se presentan los métodos y atributos de esta clase.

C.1.14. La clase ItemLabel

En el centro de la Figura C.6 se muestra la estructura de la clase `ItemLabel`. Esta clase se utiliza para generar un elemento SVG que representa a una etiqueta UML. Se utilizan los métodos para acceder a las propiedades del objeto y generar su representación en un documento SVG. En la Tabla C.14 se presentan los métodos y atributos de esta clase.

C.1.15. La clase FocusOfControl

A la derecha de la Figura C.6 se muestra la estructura de la clase `FocusOfControl`. Esta clase se utiliza para generar un elemento SVG que representa a un foco de control UML. Se utilizan los métodos para acceder a las propiedades del objeto y generar su representación SVG. En la Tabla C.15 se presentan los métodos y atributos de esta clase.

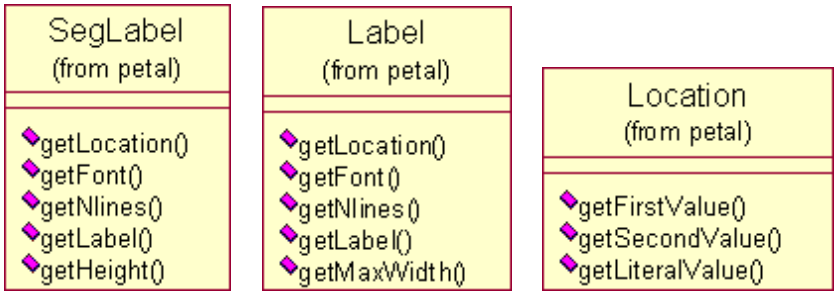


Figura C.7: Clases SegLabel, Label y Location

C.1.16. La clase SegLabel

A la izquierda de la Figura C.7 se muestra la estructura de la clase `SegLabel`. Esta clase se utiliza para extraer cadenas segmentadas de notas UML y de etiquetas UML. Los textos de estos elementos UML generalmente se presentan en más de una línea de texto, por lo que se crean instancias de esta clase para almacenar este tipo de textos. Para generar representaciones SVG de notas UML y etiquetas UML se utiliza esta clase para extraer los textos de estos elementos UML. En la Tabla C.16 se presentan los métodos y atributos de esta clase.

C.1.17. La clase Label

En el centro de la Figura C.7 se muestra la estructura de la clase `Label`. Esta clase se utiliza para extraer cadenas de elementos UML que no presentan textos en más de una línea, como por ejemplo los textos de mensajes UML. En la Tabla C.17 se presentan los métodos y atributos de esta clase.

C.1.18. La clase Location

A la derecha de la Figura C.7 se muestra la estructura de la clase `Location`. Esta clase se utiliza para obtener las coordenadas de elementos UML dentro de un diagrama UML. Para generar las representaciones SVG de elementos UML se necesita conocer sus posiciones dentro del diagrama, por lo que se utiliza a `Location`, ya que permite extraer las coordenadas de elementos UML. En la Tabla C.18 se presentan los métodos y atributos de esta clase.

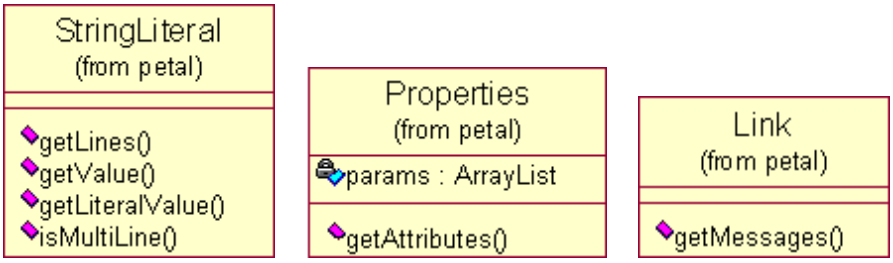


Figura C.8: Clases StringLiteral, Properties y Link

C.1.19. La clase StringLiteral

A la izquierda de la Figura C.8 se muestra la estructura de la clase `StringLiteral`. Al igual que `SegLabel`, `String Literal` permite obtener textos que tengan más de una línea. La diferencia entre estas dos clases reside en que `StringLiteral` se utiliza para obtener textos de etiquetas UML, mientras que `SegLabel` se utiliza para textos de notas UML. En la Tabla C.19 se presentan los métodos y atributos de esta clase.

C.1.20. La clase Properties

En el centro de la Figura C.8 se muestra la estructura de la clase `Properties`. Esta clase se utiliza para obtener las propiedades de los elementos UML. En la Tabla C.20 se presentan los métodos y atributos de esta clase.

C.1.21. La clase Link

A la derecha de la Figura C.8 se muestra la estructura de la clase `Link`. Esta clase se utiliza para obtener las descripciones de mensajes UML dentro del mecanismo de un diagrama UML de secuencia. Esta clase contiene las relaciones entre objetos UML a través de mensajes UML. En la Tabla C.21 se presentan los métodos y atributos de esta clase.

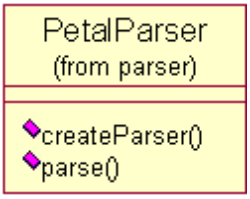


Figura C.9: La clase PetalParser

C.1.22. La clase PetalParser

En la Figura C.9 se muestra la estructura de la clase `PetalParser`. Esta clase se utiliza para abrir y analizar gramaticalmente archivos `.mdl`. En la Tabla C.22 se presentan los métodos y atributos de esta clase.

C.2. Descripción de las clases de CrazyBeans

Este paquete se utiliza para manipular elementos UML de archivos `.mdl` de Rational Rose. Genera las representaciones en memoria (objetos Java) de todos los elementos UML de un archivo `.mdl`. CrazyBeans es un paquete externo, por lo que únicamente se presentan las descripciones de las clases que se utilizan. Para saber más acerca de este paquete, en [10] se puede encontrar el sitio Web principal.

Las clases que se utilizan de este paquete son: `cb.petal.Petal`, `cb.petal. PetalObject`, `cb.petal.PetalNode`, `cb.petal.PetalFile`, `cb.petal.ClassCategory`, `cb.petal.Design`, `cb.petal.Diagram`, `cb.petal.InteractionDiagram`, `cb.petal.Mechanism`, `cb.petal. InterObjView`, `cb.petal.Message`, `cb.petal.InterMessView`, `cb.petal.NoteView`, `cb.petal.ItemLabel`, `cb.petal.FocusOfControl`, `cb.petal.SegLabel`, `cb.petal.Label`, `cb.petal.Location`, `cb.petal.StringLiteral`, `cb.petal.Properties`, `cb.petal.Tag`, `cb.petal.Link`, `cb.petal.List` y `cb.parser.PetalParser`. En las Tablas C.1 - C.22 se presentan las descripciones de estas clases.

Clase	`cb.petal.Petal`	
Descripción	Esta clase representa en alto nivel objetos de este paquete.	
Superclases	PetalObject	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**

Tabla C.1: Atributos y métodos de la clase Petal

Clase	`cb.petal.PetalObject`	
Descripción	Esta clase es la súper clase de todos los objetos que contienen una lista de propiedades.	
Superclases	`PetalNode`	
Subclases	`ActionTime, Attribute, Compartment, Defaults, Design, Event, ExternalDoc, FocusOfControl, Font, ItemLabel, Label, Mechanism, Parameter, Petal, QuidObject, SegLabel, SemanticInfo y View`	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`void`	`init`
	parámetros	No recibe parámetros
	descripción	Este método realiza cualquier acción necesaria después de haber establecido todas las propiedades del objeto, por ejemplo, asignar un identificador al objeto.

Tabla C.2: Atributos y métodos de la clase PetalObject

Interfase	`cb.petal.PetalNode`	
Descripción	Esta interfase funciona como súper clase para todos los demás nodos, por ejemplo objetos, listas, etc.	
Superclases	No tiene superclases	
Subclases	`List, Literal, PetalFile y PetalObject`	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`int`	`getChildCount`
	parámetros	No recibe parámetros
	descripción	Este método regresa el número de elementos hijo que se tienen.

Tabla C.3: Atributos y métodos de la interfase PetalNode

Clase	`cb.petal.PetalFile`	
Descripción	Esta clase representa el nodo de más alto nivel. Se utiliza para crear y guardar en disco duro los elementos UML generados.	
Superclases	No tiene superclases	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`cb.petal.Design`	`getDesign`
	parámetros	No recibe parámetros
	descripción	Este método regresa el diseño de un archivo `.mdl`

Tabla C.4: Atributos y métodos de la clase PetalFile

Clase	cb.petal.ClassCategory	
Descripción	Esta clase se usa para estructurar en sub-categorías o vistas un archivo `.mdl`	
Superclases	No tiene superclases	
Subclases	`LogicalCategory y UseCaseCategory`	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`void`	`init`
	parámetros	No recibe parámetros
	descripción	Este método realiza cualquier acción necesaria después de haber establecido todas las propiedades del objeto, por ejemplo, obtener un identificador para el objeto

Tabla C.5: Atributos y métodos de la clase ClassCategory

Clase	`cb.petal.Design`	
Descripción	Esta clase representa en alto nivel de diseño objetos. Instancias de esta clase contienen la estructura de diseño de archivos `.mdl`.	
Superclases	No tiene superclases	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`ClassCategory`	`getLogicalView`
	parámetros	No recibe parámetros
	descripción	Este método regresa la vista lógica que tiene un archivo `.mdl`
`public`	`ClassCategory`	`getUseCaseView`
	parámetros	No recibe parámetros
	descripción	Este método regresa la vista de casos de uso que tiene un archivo `.mdl`

Tabla C.6: Atributos y métodos de la clase Design

Clase	`cb.petal.Diagram`	
Descripción	Esta clase es la súper clase de todas las clases de diagramas, por ejemplo `ClassDiagram`, `InteractionDiagram`, etc.	
Superclases	`PetalObject`	
Subclases	`ActivityDiagram, ClassDiagram, InteractionDiagram, ObjectDiagram, UseCaseDiagram`	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`int`	`get_x`
	parámetros **descripción**	No recibe parámetros Este método regresa la siguiente coordenada `x` para un nuevo objeto.
`public`	`int`	`get_y`
	parámetros **descripción**	No recibe parámetros Este método regresa la siguiente coordenada `y` para un nuevo objeto.

Tabla C.7: Atributos y métodos de la clase Diagram

Clase	`cb.petal.InteractionDiagram`	
Descripción	Esta clase representa un diagrama UML de interacción.	
Superclases	`Diagram`	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`int`	`get_x`
	parámetros **descripción**	No recibe parámetros Este método regresa la coordenada `x` del objeto.
`public`	`int`	`get_y`
	parámetros **descripción**	No recibe parámetros Este método regresa la coordenada `y` del objeto.

Tabla C.8: Atributos y métodos de la clase InteractionDiagram

Clase	`cb.petal.Mechanism`	
Descripción	Esta clase representa el mecanismo que siguen los diagramas UML de secuencia.	
Superclases	`PetalObject`	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`List`	`getLogicalModels`
	parámetros **descripción**	No recibe parámetros Este método obtiene el mecanismo de un diagrama de interacción.
`public`	`int`	`getTag`
	parámetros **descripción**	No recibe parámetros Este método obtiene el identificador del objeto

Tabla C.9: Atributos y métodos de la clase Mechanism

Clase	`cb.petal.InterObjView`	
Descripción	Esta clase representa un objeto UML de un diagrama UML de secuencia	
Superclases	`PetalObject`	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`FocusOfControl`	`getFocusOfCOntrol`
	parámetros **descripción**	No recibe parámetros Este método regresa los focos de control del objeto UML
`public`	`ItemLabel`	`getLabel`
	parámetros **descripción**	No recibe parámetros Este método regresa el nombre o leyenda del objeto UML

Tabla C.10: Atributos y métodos de la clase InterObjView

Clase	`cb.petal.Message`	
Descripción	Esta clase representa un mensaje UML de un diagrama UML	
Superclases	`PetalObject`	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`int`	`getOrdinal`
	parámetros **descripción**	No recibe parámetros Este método regresa el número de secuencia que le corresponde al objeto UML en un diagrama de secuencia.
`public`	`String`	`getSynchronization`
	parámetros **descripción**	No recibe parámetros Este método regresa el tipo de mensaje, por ejemplo, mensaje simple, mensaje de respuesta, etc.

Tabla C.11: Atributos y métodos de la clase Message

Clase	`cb.petal.InterMessView`	
Descripción	Esta clase representa un mensaje UML de un diagrama UML de secuencia.	
Superclases	`PetalObject`	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`Location`	`getOrigin`
	parámetros	No recibe parámetros
	descripción	Este método regresa la coordenada de origen del mensaje UML
`public`	`SegLabel`	`getLabel`
	parámetros **descripción**	No recibe parámetros Este método regresa un objeto de tipo `SegLabel` el cual contiene el texto del mensaje UML
`public`	`Location`	`getTerminus`
	parámetros **descripción**	No recibe parámetros Este método regresa la coordenada del mensaje UML

Tabla C.12: Atributos y métodos de la clase InterMessView

Clase	`cb.petal.NoteView`	
Descripción	Esta clase representa una nota UML de un diagrama UML de secuencia	
Superclases	`PetalObject`	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`ItemLabel`	`getLabel`
	parámetros **descripción**	No recibe parámetros Este método regresa un objeto de tipo `ItemLabel` el cual contiene el texto de la nota UML
`public`	`int`	`getTag`
	parámetros **descripción**	No recibe parámetros Este método regresa el identificador de la nota UML dentro del diagrama UML

Tabla C.13: Atributos y métodos de la clase NoteView

Clase	`cb.petal.ItemLabel`	
Descripción	Esta clase representa una nota UML dentro de un archivo `.mdl`	
Superclases	`PetalObject`	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`String`	`getLabel`
	parámetros	No recibe parámetros
	descripción	Este método regresa el texto en cadena del objeto `ItemLabel`
`public`	`Location`	`getLocation`
	parámetros	No recibe parámetros
	descripción	Este método regresa las coordenadas de un elemento UML.
`public`	`int`	`getNlines`
	parámetros	No recibe parámetros
	descripción	Este método regresa el número de líneas que utiliza el texto de la etiqueta UML.

Tabla C.14: Atributos y métodos de la clase ItemLabel

Clase	`cb.petal.FocusOfControl`	
Descripción	Esta clase representa objetos de control UML	
Superclases	`PetalObject`	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`Location`	`getLocation`
	parámetros	No recibe parámetros
	descripción	Este método regresa las coordenadas del elemento UML.
`public`	`Tag`	`getInterObjView`
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para obtener identificadores de objetos.
`public`	`int`	`getHeight`
	parámetros	No recibe parámetros
	descripción	Este método obtiene el alto del elemento UML.
`public`	`int`	`getYCoord`
	parámetros	No recibe parámetros
	descripción	Este método obtiene la coordenada `y` del elemento UML
`public`	`int`	`getXCoord`
	parámetros	No recibe parámetros
	descripción	Este método obtiene la coordenada `x` del elemento UML en el diagrama.

Tabla C.15: Atributos y métodos de la clase FocusOfControl

Clase	`cb.petal.SegLabel`	
Descripción	Esta clase representa objetos de cadenas segmentadas	
Superclases	`PetalObject`	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto.		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`Location`	`getLocation`
	parámetros	No recibe parámetros
	descripción	Este método obtiene las coordenadas del elemento UML
`public`	`String`	`getLabel`
	parámetros	No recibe parámetros
	descripción	Este método regresa el texto en cadena del elemento UML
`public`	`int`	`getNLines`
	parámetros	No recibe parámetros
	descripción	Este método regresa el número de líneas que se utilizan para almacenar una cadena.
`public`	`Font`	`getFont`
	parámetros	No recibe parámetros
	descripción	Este método obtiene tipos de letra y tamaños de letra que utiliza el objeto.
`public`	`int`	`getHeight`
	parámetros	No recibe parámetros
	descripción	Este método regresa la dimensión de altura, la cual limita la región en que se presenta el elemento UML.

Tabla C.16: Atributos y métodos de la clase SegLabel

Clase	`cb.petal.Label`	
Descripción	Esta clase representa una etiqueta UML de un diagrama UML de secuencia	
Superclases	`PetalObject`	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades de la etiqueta UML		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`Location`	`getLocation`
	parámetros	No recibe parámetros
	descripción	Este método obtiene las coordenadas del elemento dentro del diagrama UML
`public`	`String`	`getLabel`
	parámetros	No recibe parámetros
	descripción	Este método regresa el texto en cadena del elemento UML
`public`	`int`	`getNLines`
	parámetros	No recibe parámetros
	descripción	Este método regresa el número de líneas que utiliza el elemento UML
`public`	`Font`	`getFont`
	parámetros	No recibe parámetros
	descripción	Este método obtiene el tipo y tamaño de letra
`public`	`int`	`getMaxWidth`
	parámetros	No recibe parámetros
	descripción	Este método obtiene el límite de anchura, la cual limita la región del elemento UML

Tabla C.17: Atributos y métodos de la clase Label

Clase	`cb.petal.Location`	
Descripción	Esta clase contiene una tupla de enteros. Se usa para almacenar coordenadas.	
Superclases	No tiene superclases	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`int`	`getFirstValue`
	parámetros	No recibe parámetros
	descripción	Este método regresa el primer valor de la tupla de enteros
`public`	`int`	`getSecondValue`
	parámetros	No recibe parámetros
	descripción	Este método regresa el segundo valor de la tupla de enteros.
`public`	`java.lang.Object`	`getLiteralValue`
	parámetros	No recibe parámetros
	descripción	Este método regresa ambas coordenadas del elemento UML

Tabla C.18: Atributos y métodos de la clase Location

Clase	`cb.petal.StringLiteral`	
Descripción	Esta clase representa dos tipos de cadenas. El primer tipo es una cadena sencilla y el segundo es una cadena de múltiples líneas	
Superclases	`Literal`	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`Collection`	`getLines`
	parámetros	No recibe parámetros
	descripción	Este método regresa todas las cadenas del objeto
`public`	`Object`	`getLiteralValue`
	parámetros	No recibe parámetros
	descripción	Este método regresa todas las cadenas del objeto incluyendo el carácter |
`public`	`String`	`getValue`
	parámetros	No recibe parámetros
	descripción	Este método regresa únicamente la primera cadena del objeto.
`public`	`boolean`	`isMultiLine`
	parámetros	No recibe parámetros
	descripción	Este método regresa *true* si el objeto contiene dos o mas cadenas, en otro caso regresa *false*

Tabla C.19: Atributos y métodos de la clase StringLiteral

Clase	`cb.petal.Properties`	
Descripción	Esta clase representa el conjunto de propiedades de cada objeto en un archivo `.mdl`	
Superclases	`PetalObject`	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`List`	`getAttributes`
	parámetros	No recibe parámetros
	descripción	Este método regresa todos los atributos en forma de lista que contiene el objeto.

Tabla C.20: Atributos y métodos de la clase Properties

Clase	`cb.petal.Link`	
Descripción	Esta clase representa conexiones entre elementos UML de diagramas UML	
Superclases	`PetalObject`	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
`public`	`ArrayList`	`params`
El atributo `params` se utiliza para almacenar todas las propiedades del objeto		
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`List`	`getMessages`
	parámetros	No recibe parámetros
	descripción	Este método se utiliza para obtener todos los mensajes UML que tienen relación con un objeto UML

Tabla C.21: Atributos y métodos de la clase Link

Clase	cb.parser.PetalParser	
Descripción	Esta clase se utiliza para dividir un archivo `.mdl` en *tokens*. Se utiliza como *parser*, un analizador gramatical de archivos `.mdl`	
Superclases	No tiene superclases	
Subclases	No tiene subclases	
Asociaciones	No tiene asociaciones	
Atributos:		
visible	**tipo**	**nombre**
Constructores y métodos		
visible	**salida**	**nombre**
`public`	`PetalParser`	`createParser`
	parámetros	`String filename`
	descripción	Este método regresa un objeto de tipo `PetalParser`
`public`	`PetalFile`	`Parse`
	parámetros	No recibe parámetros
	descripción	Este método construye los objetos principales de un objeto `PetalFile`, `Petal` y `Design`

Tabla C.22: Atributos y métodos de la clase PetalParser

Bibliografía

[1] Rogue Wave, http://www.roguewave.com/products/amf/.

[2] Rogue Wave, http://www.roguewave.com/products/leif/.

[3] Cape Clear, http://www.capeclear.com/.

[4] Oracle, http://www.oracle.com/technology/bpel/.

[5] Infravio, http://www.infravio.com/.

[6] Infravio, http://www.infravio.com/.

[7] *Adobe*. http://www.adobe.com.

[8] *Analyzer*. StrikeIron, http://www.strikeiron.com/.

[9] *ArgoUML*. Tigris Open Source Engineering, http://argouml.tigris.org/.

[10] *CrazyBeans*. https://sourceforge.net/projects/crazybeans/.

[11] *Describe*. Embarcadero, http://www.embarcadero.com/products/describe/.

[12] *Ideogramic*. http://www.ideogramic.com/.

[13] *MagicDraw*. http://www.magicdraw.com/.

[14] *Microsoft*. Microsoft, http://www.microsoft.com/.

[15] *NAICS, North American Industry Classification System*. http://www.naics.com.

[16] *ParaSoft SOAPtest*. ParaSoft, http://www.parasoft.com/.

[17] *Poseidon*. GentleWare, http://www.gentleware.com/.

[18] *Rational Rose*. IBM Rational Software, http://www.rational.com/.

[19] *RosettaNet, eBusiness Standards for the Global Supply Chain*. http://www.rosettanet.org.

[20] *UML2BPEL*. IBM ETTK.

[21] *Unisys*. www.unisys.com/.

[22] *UNSPCS, United Nations Standard Products and Services Code.* http://www.unspsc.com.

[23] *Visio.* Microsoft, http://www.microsoft.com/.

[24] *W3c, World Wide Web Consortium.* http://www.w3.org.

[25] *WebServiceTester.* Optimyz, http://www.optimyz.com/index-ie.html.

[26] *XML-RPC, XML Remote Procedure Calls.* http://www.xml-rpc.org, 2003.

[27] *DHTML, Dynamic HyperText Markup Language.* http://www.webreference.com/dhtml/, 2004.

[28] *HTML, HyperText Markup Language Specification 4.01.* W3C, World Wide Web Consortium, http://www.w3.org/TR/html4/, 2004.

[29] *SGML, Standard Generalized Markup Language.* W3C, World Wide Web Consortium, http://www.w3.org/MarkUp/SGML/, 2004.

[30] *XHTML, Extensible HyperText Markup Language (Second Edition).* W3C, World Wide Web Consortium, http://www.w3.org/TR/xhtml1/, 2004.

[31] Giner Alor-Hernández , César Sandoval-Hernández. Descubrimiento dinámico de servicios web en nodos uddi utilizando usml. *CORE*, 2004.

[32] Giner Alor-Hernández , César Sandoval-Hernández. Generación dinámica de guis para la invocación de servicios web publicados en nodos uddi. *CORE*, 2004.

[33] Giner Alor-Hernández , José-Oscar Olmedo-Aguirre. Sistema de intermediación para el comercio electrónico b2b basado en servicios web. *XII Congreso Internacional en Computación*, Vol. 3:330.

[34] Giner Alor-Hernández , José-Oscar Olmedo-Aguirre. Automatización de la cadena de suministro en comercio electrónico b2b. *CIE*, 2003.

[35] Giner Alor-Hernández , José-Oscar Olmedo-Aguirre. Automatización de la cadena de suministro usando uddi. *CNCIIC-ANIEI*, 2003.

[36] Giner Alor-Hernández , Lucio-Daniel Castelán-Vega. Bpims-ws: Business process integration and monitoring system. *CIICC*, 2004.

[37] Lucio-Daniel Castelán-Vega , Giner Alor-Hernández. Ms4ws: Sistema de monitoreo para servicios web con uml. *XVII Congreso Nacional y III Congreso Internacional de Informática y Computación*, 20-23 Octubre 2004.

[38] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[39] Second Edition Benjamin/Cummings. *Object-Oriented Design.* 1994.

[40] Bellwood T. et al. *UDDI, Universal Description, Discovery and Integration Version 3.0*. OASIS, Organization for Advancement of Structured information System, http://www.uddi.org/specification.html, July 2002.

[41] Chinnini et al. *WSDL, Web Services Description Language Specification*. W3C, World Wide Web Consortium, http://www.w3c.org/TR/wsdl20, 2004.

[42] Dean Jackson et al. *SVG, Scalable Vector Graphics. XML Graphics For the Web Version 1.2*. W3C, World Wide Web Consortium, http://www.w3c.org/TR/SVG12, 2004.

[43] E. Amstrong et al. *The Java Web Services Tutorial*. Sun Microsystems, 2003.

[44] Francisco Curbera et al. Unraveling the web services. an introduction to soap, wsdl and uddi. *IEEE Internet Computing*, 2002.

[45] Gudgin M. et al. *SOAP, Simple Object Access Protocol Recommendation Version 1.1*. W3C, World Wide Web Consortium, http://www.w3c.org/TR/soap12-part1/, 2003.

[46] Satish Thatte et al. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*. IBM Software Group, 5 May 2003.

[47] Vidur Apparao et al. *DOM, Document Object Model W3C Recommendation*. W3C, World Wide Web Consortium, http://www.w3.org/DOM/, October 1998.

[48] Yergeau F. et al. *XML, Extensible Markup Language Recommendation Version 1.1*. W3C, World Wide Web Consortium, http://www.w3c.org/XML/, February 2004.

[49] Linda Heaton. *UML, Unified Modeling Language*. OMG, Object Management Group, http://www.omg.org/technology/documents/formal/uml.htm, 2004.

[50] Linda Heaton. *XMI, XML Metadata Interchange Versions 1.2 and 2.0*. OMG, Object Management Group, http://www.omg.org/technology/documents/formal/xmi.htm, 2004.

[51] P. Jonsson I. Jacobson, M. Christerson and G. Övergaard. *Object-Oriented Software Engineering. A Use Case Driven Approach.* ACM Press, 1992.

[52] W. Premerlani-F. Eddy J. Rumbaugh, M. Blaha and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[53] Christerson M. Jonsson P. Jacobson, I. and G. Overgaard. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addisson-Wesley, 1992.