UNIVERSIDAD NACIONAL DE SAN LUIS

FACULTAD DE CIENCIAS FÍSICO, MATEMÁTICAS Y NATURALES

Tesis
para optar al título de postgrado correspondiente a la
Maestría en Ingeniería de Software

Definición de Reglas de Transformación de Grafos y QoS para la Interfaz entre Aplicaciones Externas a un Flujo de Trabajo con Servicios Web

Lic. Paola Martellotto

Director: Dr. Daniel Riesco

Co-directora: Mg. Marcela E. Daniele

San Luis
Febrero de 2010

Agradecimientos

Quiero agradecer a todas las personas que de una u otra forma han hecho posible la realización de esta Tesis. De forma muy especial a mi director de Tesis Daniel Riesco, que ha permitido llevar a un final exitoso este trabajo, por su asesoría, dedicación y sus valiosos aportes. A mi co-directora Marcela Daniele, por su tiempo y desvelo, y por su interés en aportar ideas y ayudar a la concreción del trabajo. Agradezco especialmente sus constantes palabras de aliento y sus consejos precisos.

A mis padres y a mi esposo, por su incondicional apoyo durante este largo camino.

Además, quiero agradecer a la Universidad Nacional de Río Cuarto, y en particular, a la Facultad de Ciencias Exactas, Físico-Químicas y Naturales, por brindarme el apoyo económico necesario para realizar esta maestría. Al Departamento de Computación por permitirme continuar mi formación individual, y en especial a mis compañeros de área, quienes debieron cubrir más tareas por mis ausencias temporales.

Finalmente, a la Universidad Nacional de San Luis, por brindarme la posibilidad de optar a un título de posgraduación de muy alto nivel, y a todos los profesores por trasmitirme sus apreciados conocimientos.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

CAPÍTULO 1.

Introducción

La presente Tesis de maestría tiene como objetivo principal permitir un manejo transparente de la ubicación de las aplicaciones externas a un Sistema de Gestión de Flujos de Trabajo (SGFT) al momento de invocarlas, favoreciendo su distribución en la red. Para lograrlo, se propone especificar la interfaz que permite dicha comunicación con servicios web, y valerse de la técnica de Transformación de Grafos para encontrar el servicio web más adecuado según los requerimientos del SGFT.

El resto del capítulo se estructura como sigue. En la sección 1.1 se presenta la motivación que dio lugar a la realización de esta Tesis. En la sección 1.2 se plantean los objetivos de la misma, y finalmente en la sección 1.3 se describe la estructura del resto del documento, presentando el contenido de todos los capítulos desarrollados.

1.1. Motivación y Propuesta

La globalización y los cambios de paradigmas empresariales, la evolución de las tecnologías de la información y la política liberal imperante hoy en el mundo, ubican a las organizaciones en el juego de la competitividad internacional. Las iniciativas sobre calidad y la mejora continua de los procesos ya no son suficientes. Se deben considerar nuevos paradigmas empresariales que permitan a las organizaciones obtener mejoras radicales mediante el uso de nuevas y potentes herramientas que faciliten el diseño del trabajo. Las organizaciones se orientan a ser más horizontales hacia el enfoque de redes de procesos, las que deben ser diseñadas de principio a fin, empleando nuevas tecnologías. El cambio radical de procesos comprende la visualización de nuevas estrategias de trabajo, el desarrollo de la propia actividad de diseño del proceso con tecnologías de la información innovadoras y la implementación del cambio en todas sus dimensiones: la tecnológica, la humana y la organizativa, a fin de establecer los mecanismos para un constante crecimiento del valor de las organizaciones. Para lograr un adecuado cambio integral de los procesos, es conveniente confeccionar un mapa de los mismos a fin de determinar los niveles de cambios a realizar. Por lo tanto, la identificación y el modelado de negocio necesario para generar un producto o ejecutar un servicio, es de gran importancia en el desarrollo de cualquier industria. Un conocimiento claro y ordenado de los procesos de negocio facilita su optimización y adaptación. Estas características proporcionan a la organización una gran capacidad de reacción cuando el proceso es automático.

Un proceso de negocio es un conjunto de tareas relacionadas lógicamente que se ejecuta con la intención de obtener un resultado de negocio particular, el cual incluye recursos humanos así como los recursos materiales con el objetivo de producir un beneficio para la organización. El modelado de procesos de negocio permite visualizar las tareas, actividades y flujos, así como las diferentes unidades organizacionales que son afectadas por el proceso. La ingeniería de procesos de negocio se ha definido como el planteo fundamental y diseño del proceso de negocio para lograr un mejoramiento en las medidas de rendimientos tales como costo, calidad de servicio y velocidad. Estos procesos de negocio se describen y automatizan a través de los Sistemas de Gestión de Flujos de Trabajo, más conocidos como Sistemas de Gestión de Workflow (en la literatura el término flujo de trabajo se denota por la palabra en inglés Workflow). Los Sistemas de Gestión de Flujos de Trabajo (SGFT) proporcionan herramientas integradas para automatizar los procesos de negocio, permitiendo a las organizaciones estandarizar y racionalizar las tareas repetitivas, y supervisar el progreso de las mismas, eliminando rutinas en las que es fácil cometer errores. La eliminación de los procesos que consumen tiempo y las costosas rutinas manuales,

brinda la posibilidad de conectar al personal con la información y los procesos que necesitan para mejorar los ingresos y recortar los gastos.

Por otra parte, el ambiente donde las organizaciones operan es cada vez más competitivo y agresivo. En estos días, debido a la globalización de los mercados, las compañías tienen que operar "globalmente", lo cual implica la posibilidad de interoperar unas con otras. En la actualidad existe la Coalición de Administración de Workflow (Workflow Management Coalition - WfMC), fundada en 1993 y con más de 180 miembros en 25 países, la cual está abocada al avance en la tecnología de flujo de trabajo y su uso en la industria con el propósito de estandarizar en esta materia. El Modelo de Referencia de Workflow, desarrollado por la WfMC, define un marco genérico para la construcción de Sistemas de Gestión de Flujos de Trabajo, permitiendo la interoperabilidad entre ellos y con otras aplicaciones involucradas. Dicho modelo define cinco interfaces, que permiten a las aplicaciones del Flujo de Trabajo la comunicación a distintos niveles. En particular, para permitir la interacción de los usuarios con el motor del Flujo de Trabajo utiliza una lista de trabajo, que es manejada por un administrador. La Interfaz de las Aplicaciones de Clientes es la encargada de manejar la interacción entre el motor Del Flujo de Trabajo y el administrador de la lista de trabajo. Por otro lado, la Interfaz de las Aplicaciones Invocadas se define para la invocación de las aplicaciones externas. La WfMC ha especificado un conjunto de WAPIs (Workflow Aplication Programming Interfaces) para la administración de los Flujos de Trabajos. Estas WAPIs definen las funciones de las interfaces como llamadas a APIs en un lenguaje de tercera generación, obligando a conocer la información acerca de la aplicación y su invocación en tiempo de desarrollo. Si bien en muchos Sistemas de Gestión de Flujos de Trabajo se conocen a priori las aplicaciones que se desean utilizar, también existen sistemas donde diferentes aplicaciones que brindan un mismo servicio podrían ser requeridas por el motor del SGFT.

Los servicios web proveen esencialmente un medio estándar de comunicación entre diferentes aplicaciones de software, ofreciendo la capacidad de acceder a servicios heterogéneos de forma unificada e interoperable a través de Internet. Los servicios web son registrados a través del protocolo denominado UDDI (Universal Description, Discovery and Integration) para ser localizados y usados por las aplicaciones. Este protocolo presenta algunos inconvenientes al momento de seleccionar un servicio. Uno de estos inconvenientes está dado por el hecho de que no tiene en cuenta las características de calidad de los servicios web, las cuales se refieren a la habilidad del servicio de responder a las invocaciones y llevarlas a cabo en consonancia con las expectativas del proveedor y de

los clientes. Diversos factores de calidad que reflejan las expectativas del cliente, como la disponibilidad, conectividad y alta respuesta, se vuelven clave para mantener un negocio competitivo y viable. Otro de los inconvenientes que presenta el protocolo UDDI es la dificultad de establecer una correspondencia entre los requerimientos del solicitante y las múltiples especificaciones de servicios web existentes en la actualidad que proveen similares funcionalidades, aunque tienen diferente descripción sintáctica. Esta Tesis apunta a mejorar la comunicación del SGFT con las aplicaciones de software externas a través del uso de servicios web, y optimizar la selección de los mismos usando Técnicas de Transformación de Grafos. A continuación se detallan los objetivos de la Tesis.

1.2. Objetivos

El objetivo principal de esta Tesis es favorecer la distribución en la red de las aplicaciones externas a un Flujo de Trabajo, y que esto resulte transparente al motor del SGFT, a la hora de invocarlas.

Este objetivo general se desglosa en los siguientes objetivos específicos:

1. Proponer una solución eficiente para invocar las aplicaciones externas que resultan más adecuadas de acuerdo a los requerimientos del SGFT, a través de la especificación de la Interfaz de las Aplicaciones Invocadas del Modelo de Referencia de Workflow con servicios web.
2. Incorporar a la especificación del servicio web las características de calidad (QoS) del mismo, es decir, sus requerimientos no funcionales.
3. Proporcionar una especificación precisa del servicio web a través de la aplicación de Reglas de Transformación de Grafos, para establecer la correspondencia entre los requerimientos del solicitante y las especificaciones de todos los servicios web existentes que semánticamente se comportan igual.

La especificación de la Interfaz de las Aplicaciones Invocadas con servicios web permite la distribución transparente de las aplicaciones externas, y así el motor del SGFT puede invocarlas sin la necesidad de conocer su ubicación exacta, con el importante beneficio de que las aplicaciones pueden cambiar su ubicación en la red sin que esto implique ningún cambio en su invocación.

Asimismo, con la incorporación de los atributos de calidad (Quality of Service - QoS) a la descripción de los servicios web se hace posible evaluar la calidad de los mismos, filtrar la lista de todos los servicios descubiertos y obtener los más adecuados.

Con respecto al protocolo UDDI, actualmente utilizado para registrar y localizar los servicios web en la red, dado que el mismo requiere una descripción específica de la aplicación o servicio concreto que se quiere invocar (la cual intenta describir de alguna forma la semántica del mismo), los resultados de las consultas estén limitados por las palabras clave de la búsqueda, y no son del todo fiables. La utilización de reglas de transformación de grafos permite lograr una especificación semántica más precisa del servicio web, y así establecer una correspondencia entre los requerimientos del usuario y los servicios web disponibles, y que el motor del Flujo de Trabajo pueda elegir en el momento de la invocación. La formalización de la especificación como un grafo con atributos, permite realizar una correspondencia con los servicios web disponibles en la red que cumplen con el morfismo establecido, logrando que el Flujo de Trabajo se comporte internamente de forma distribuida. Además, facilita la incorporación de servicios nuevos, y la invocación no está limitada a una aplicación específica, definida en tiempo de desarrollo.

1.3. Estructura del documento

El documento se ha estructurado como se muestra a continuación:

- El capítulo 1 comprende la introducción.
- El capítulo 2 analiza el estado del arte en cuanto a los Sistemas de Gestión de Flujos de Trabajo y las distintas posibilidades de implementar la comunicación del motor del SGFT con la Interfaz de las Aplicaciones Invocadas u otros SGFT, la incorporación del significado semántico de los servicios web en la selección automática de los mismos y de las características de calidad.
- El capítulo 3 describe los flujos de trabajo y los sistemas de gestión de flujos de trabajo, detallando la definición oficial provista por la Workflow Management Coalition (WfMC), la clasificación de SGFT, los conceptos básicos y esfuerzos actuales de estandarización.
- El capítulo 4 contiene una introducción a los conceptos básicos de los servicios web y el lenguaje WSDL utilizado para especificarlos. Detalla los objetivos de los servicios web, las tecnologías subyacentes, algunos ejemplos de aplicación, y la arquitectura básica y

dinámica de selección. Finalmente, presenta la especificación funcional de la Interfaz de la Aplicaciones Invocadas, con servicios web.

- El capítulo 5 se refiere a la especificación no funcional de la Interfaz de la Aplicaciones Invocadas y comprende la descripción de las características de calidad deseables en los servicios web, la especificación no funcional incorporando la descripción de dichas características y una exposición de la especificación completa.
- El capítulo 6 se refiere a la utilización de las reglas de transformación de grafos en la invocación de aplicaciones externas al SGFT e incluye una descripción breve de la técnica de transformación de grafos y su relación con el modelado orientado a objetos, el esquema propuesto para la selección automática de servicios web con reglas de transformación de grafos y su utilización en la invocación de una aplicación externa.
- El capítulo 7 presenta el caso de estudio, relacionado al proceso de desarrollo de software OpenUP/Basic y la invocación de aplicaciones externas durante la ejecución de las tareas propuesta en este proceso.
- El capítulo 8 presenta las conclusiones de la Tesis.
- El Anexo A presenta los Servicios Web y detalla las tecnologías básicas de los servicios web.
- El Anexo B presenta las características de calidad de los servicios web.
- El Anexo C detalla la tecnología de transformación de grafos, ampliamente utilizada para especificar cómo deben ser construidos y cómo deben evolucionar los modelos de software.
- El Anexo D detalla las APIs (Application Programming Interfaces) definidas en el estándar de la WfMC, que soportan los Sistemas de Gestión de Flujos de Trabajo (SGFT).

CAPÍTULO 2.

Estado del Arte

En este capítulo se hace una revisión del estado del arte en lo que respecta a tres puntos: las implementaciones alternativas de la comunicación del Sistemas de Gestión de Flujo de Trabajo (SGFT) con las aplicaciones externas, lo cual se detalla en la sección 2.1, la incorporación de las características de calidad de los servicios web a la descripción, publicación y descubrimiento de los mismos, lo cual se detalla en la sección 2.2, y, finalmente, las diferentes implementaciones alternativas para la búsqueda y selección dinámica de servicios web, lo cual se presenta en la sección 2.3. También, en la sección 2.4, se presentan algunas conclusiones respecto de los aportes y desventajas de los trabajos presentados, en cada una de los aspectos analizados en este capítulo.

2.1. Implementaciones alternativas de la comunicación del SGFT con las aplicaciones externas

Existen diversos trabajos que presentan alternativas para implementar la comunicación del SGFT con las aplicaciones de clientes y externas al mismo. A continuación se mencionan y describen diferentes propuestas exploradas para el desarrollo de esta Tesis.

En [WfMC09] la Workflow Management Coalition (WfMC) presenta una propuesta de especificación de las Interfaces 2 y 4 basada en el uso de IDL y enlaces con OLE como alternativas a las especificaciones existentes C y MIME. Los primeros trabajos de la WfMC (sobre especificación de las interfaces del Modelo de Referencia de Workflow) se concentran en la definición de las funciones de las interfaces y su especificación como llamadas a interfaces de programación de aplicaciones (Aplication Programming Interfaces - APIs) en el lenguaje "C". Inicialmente, la estandarización de los SGFT se enfocó en especificar el *servicio de ejecución del flujo de trabajo (workflow enactment service)* como una "caja negra", un objeto de alta granularidad con varias interfaces a través de las cuales otros objetos de software pueden requerir servicios. Este enfoque oculta la complejidad interna del servicio de ejecución del flujo de trabajo, lo cual hace muy difícil su estandarización. Para la comunicación entre las aplicaciones de software y el *servicio de ejecución del flujo de trabajo* se provee un modelo estandarizado de APIs (Workflow Aplication Programming Interfaces - WAPIs). Las WAPIs de la WfMC asumen que lo proveedores proporcionarán el soporte adecuado para que las aplicaciones de clientes puedan acceder al servicio de ejecución del flujo de trabajo desde plataformas distribuidas. En escenarios de interoperabilidad más complejos existen requerimientos de datos y servicios subyacentes comunes. Estos escenarios complejos se pueden soportar a través de la adopción incremental de tecnologías de componentes comunes, que soporten el reuso. Para ello se especifica un *Modelo de Objetos Común* el cual se puede implementar en las distintas tecnologías subyacentes. Tal enfoque se vuelve más importante en aquellos escenarios donde se requiere soportar una interoperabilidad compleja. En esos casos, es interesante construir un servicio de ejecución del flujo de trabajo unificado a partir de una arquitectura de componentes común, abarcando servicios distribuidos y métodos comunes. En este modelo se representan los elementos principales del flujo de trabajo como objetos relacionados. Se trata en realidad de un meta-modelo, que identifica las entidades principales para la definición de los procesos, sus relaciones y atributos, y agrega una representación de la definición de procesos. En esta propuesta el manejo de las consultas WAPI se reemplaza por el uso de una colección de objetos OLE y se define un filtro para reemplazar el filtro WAPI. Se define un modelo de objetos que combina las interfaces 2, 4 y 5, y también direcciona el área de

aplicaciones invocadas, donde las aplicaciones se asumen como objetos de negocio. El *servicio de ejecución del flujo de trabajo* puede construirse de manera totalmente distribuida, para explotar la tecnología de componentes objetos y soportar la interoperabilidad a través de diferentes plataformas.

En [Schmidt98] se presenta la *especificación JointFlow*, adoptada por diecinueve compañías en respuesta a las RFP (Request For Proposals) de la *Object Management Group* (OMG). Esta especificación se basa en el trabajo de la Workflow Management Coalition (WfMC) y define las interfaces que soportan la interacción en tiempo de ejecución entre los componentes del SGFT. La especificación JointFlow focaliza en definir un conjunto de interfaces que permitan la interoperabilidad de los componentes de un flujo de trabajo, y soporten el monitoreo y la asignación de recursos. La especificación JointFlow esta orientada a la construcción de SGFT distribuidos. Un proceso se representa como un conjunto de componentes, manteniendo la estructura esencial del modelo subyacente y permitiendo además el monitoreo de la ejecución de los procesos. Un flujo de trabajo complejo puede componerse de componentes de flujos de trabajo desarrollados independientemente lo cual permite la distribución del SGFT en ambientes heterogéneos. El control de los procesos está separado de las operaciones de negocio actuales, los componentes de negocio pueden integrarse con los procesos del flujo de trabajo permitiendo el reuso de lo componentes en los múltiples SGFT.

En [YLX05] los autores presentan una especificación de sistemas de gestión de flujos de trabajo utilizando computación grid (grid computing). La computación grid es una tecnología que permite compartir recursos distribuidos a gran escala e integrar los sistemas. En los SGFT construidos en un ambiente grid los componentes más funcionales del flujo de trabajo se abstraen y encapsulan en servicio grid de alto nivel, lo cual permite mayor interoperabilidad. En el Modelo de Referencia de Workflow tradicional, propuesto por la WfMC, un SGFT consiste de herramientas de administración de procesos, herramientas de gestión y monitoreo, aplicaciones de clientes, aplicaciones invocadas y motores del SGFT. Los procesos son diseñados por herramientas de definición de procesos y ejecutados por el motor, en el mismo WFMS. Las tareas son desarrolladas por los usuarios finales o las aplicaciones. Los procesos solamente pueden ser ejecutados por motores que son accedidos por usuarios específicos y aplicaciones específicas. Por lo tanto, utilizar SGFT es restrictivo de las localizaciones. En esta propuesta las tareas son desarrolladas por los servicios Grid (los cuales se basan en un conjunto de interfaces estándar). Los servicios pueden ser accedidos por cualquier aplicación de acuerdo a

estos estándares. Así, la ejecución de un proceso no esta limitada a la ubicación del motor del SGFT, los procesos se pueden diseñar, almacenar y ejecutar en cualquier lugar. Para comenzar a ejecutar un proceso los usuarios pueden elegir cualquier motor disponible distribuido en el ambiente grid. Durante la ejecución el proceso puede transferirse entre los diferentes motores de el o los SGFT del ambiente grid con el fin de conseguir un balance de carga, disminuir el costo de la comunicación entre las aplicaciones y los motores, recuperarse fácilmente de una falla, etc. Inclusive, un proceso puede ejecutarse en diferentes motores al mismo tiempo para obtener mayor eficiencia. De acuerdo a este modelo, las funcionalidades principales del SGFT se dividen en componentes independientes y servicios grid, incluyendo las herramientas de definición de procesos, el servicio de repositorio, el servicio de motor del SGFT y el servicio de repositorio de instancias del flujo de trabajo. Esta implementación permite construir SGFT basados en servicios que pueden ayudar a las empresas a manejar sus procesos de una forma más flexible y eficiente, y con un mínimo costo de desarrollo.

En [CS05] se propone encapsular la funcionalidad de las organizaciones en interfaces apropiadas y publicarlas como servicios web. Se espera que los servicios web puedan integrarse como parte de los procesos web. Para lograr esta integración propone el uso de ontologías. También es esencial para los servicios web soportar todas las fases del ciclo de vida de un proceso web. Se describe cómo aplicar semántica a cada paso del ciclo de vida de un proceso web semántico puede ayudar al rehúso, la integración y escalabilidad.

En [MY05] se presenta un lenguaje de flujo de trabajo que usa los servicios web como componentes, una arquitectura para un ambiente en tiempo de ejecución para este lenguaje, y aprovecha las ventajas de la utilización de esta clase de tecnología. La idea es desarrollar procesos de negocio formados por una composición de servicios que están disponibles en una red de computadoras. La tecnología de flujo de trabajo se usa para coordinar las interacciones entre los servicios web. Los servicios web representan los pasos lógicos que componen un flujo de trabajo. El lenguaje propuesto AELCWS no soporta la interacción directa con las personas durante la ejecución de un proceso, de modo que la Interfaz de Aplicaciones de Cliente no esta definida ni implementada. La Interfaz de Invocación de las Aplicaciones sí es soportada por este lenguaje, a través de la invocación de los servicios que conforman la definición del proceso.

En [LH03] se introduce un agente de composición de flujos de trabajo, el cual es capaz de componer flujos de trabajo de servicios web. Los flujos de trabajo de servicios web son

un conjunto de servicios web que se ejecutan de forma estructurada. Este agente utiliza las descripciones semánticas de los servicios web con el fin de encontrar servicios para un flujo de trabajo. En la composición de servicios web, puede suceder que un servicio del no esté disponible en tiempo de ejecución, y se requiera la búsqueda y reemplazo de ese servicio web por otro similar, para completar el flujo de trabajo. Este servicio web similar debería implementar la misma funcionalidad que el servicio web original. Pero, como los servicios web son desarrollados por diferentes organizaciones en el mundo, es improbable que se encuentre otro servicio web con exactamente la misma funcionalidad que el original (número diferente de parámetros de entrada y salida, significado diferente de los parámetros de entrada y salida, etc.). Se busca entonces, encontrar servicios web semánticamente similares teniendo en cuenta sus pre y pos-condiciones, es decir, el efecto que el servicio web produce sobre el *estado del sistema*. Se introduce entonces el concepto de ontologías de servicios web semánticos. Un servicio web se puede componer utilizando estas ontologías de servicios web semánticos, lo cual se refiere a definir la semántica de un servicio web, es decir, su significado, más que sus parámetros de entrada y salida. Se presenta la ontología de servicios web DAML-S, basada en el lenguaje DAML (DARPA Agent Markup Language), la cual especifica cómo reemplazar un servicio web con otro de similar descripción semántica. DAML-S está dividido en tres partes. Un *profile de servicios*, utilizado para publicar y descubrir los servicios web, el cual describe los servicios en términos de entradas, salidas, precondiciones y efectos de su ejecución. Un *modelo de procesos*, que provee una descripción detallada de la operación del servicio, es decir, cómo debería usarse el servicio web. Un *grounding*, que provee los detalles concretos sobre cómo interoperar con el servicio. Con esta tecnología se presenta un modelo para componer flujos de trabajo de servicios web, con el objetivo de automatizar el proceso de encontrar los servicios web accedidos por un flujo de trabajo. El modelo presentado aún requiere interacción humana, pero sólo en la fase inicial de composición del flujo de trabajo.

2.2. Características de calidad de los servicios web

Existen diversos trabajos que focalizan en describir, publicar y descubrir servicios web teniendo en cuenta los atributos de calidad de los mismos.

En [Ran03], el autor propone una solución para mejorar el descubrimiento de los servicios web incorporando los atributos de calidad del servicio (Quality of Service - QoS). En esta solución se utiliza un nuevo modelo de descubrimiento de servicios web que incorpora un Certificador QoS, el cual es responsable de verificar los atributos QoS ofrecidos por los proveedores del servicio. Para incorporar el certificado QoS en el modelo de descubrimiento de servicios web existente, se introduce una ligera modificación a la corriente arquitectura, ya que el registro en UDDI se realiza después que el proveedor verifica los requerimientos QoS con el Certificador. Bajo este modelo, el proveedor prepara un conjunto de medidas QoS y las comunica al Certificador, el cual las chequea y le devuelve una notificación al proveedor del servicio. Esta notificación puede ser un aviso de que el servicio web falló en los requerimientos QoS, o una certificación ID de que los requerimientos QoS han sido satisfactoriamente verificados. Una vez que el proveedor obtiene su certificación ID comienza el proceso de registración en UDDI, proveyendo la información funcional y no funcional, junto con la certificación ID. Una desventaja de esta propuesta es la redundancia en el cálculo de las medidas QoS. Las medidas QoS deben ser provistas por el proveedor del servicio al momento de registrarse, mientras que el Certificador QoS eventualmente desarrollará otras medidas para verificar los requerimientos QoS del proveedor.

En [HSD08], los autores proponen perfeccionar el proceso de descubrimiento de servicios web a través de la elaboración de un framework que permita mejorar la recuperación de los mismos, mediante la combinación de su correspondencia sintáctica y semántica, y que soporte también la especificación de la información QoS. Este framework se basa en los componentes básicos de un servicio web (Proveedor del servicio, Consumidor del servicio, Registro UDDI) con la capacidad adicional de almacenar las características QoS usando la estructura de datos *tModel*. El modelo es ampliado con tres agentes: *Discovery Agent*, *Reputation Manager* y *Service Mediator*. El *Discovery Agent* recibe los requerimientos de los clientes y luego busca en el registro UDDI la lista de todos los servicios web que se corresponden con los requerimientos. Luego consulta al *Reputation Manager* por la reputación de cada servicio (respaldo) y organiza la lista de servicios de acuerdo a los criterios QoS. El *Discovery Agent* comenzará entonces un test inicial y enviará el resultado al consumidor. El *Discovery Agent* pude usar la historia de los servicios web almacenada en el *Reputation Manager* para comunicarse con otros agentes que usaron el servicio. El consumidor compara los tres resultados: la descripción propia del proveedor del servicio web de la puntuación de

sus atributos de calidad, la recomendación del *Reputation Manager* y los resultados del test inicial, y finalmente selecciona el servicio web más apropiado de acuerdo a su análisis. Este modelo no requiere modificar el modelo de servicios web actual y permite compartir el conocimiento entre el proveedor y el consumidor del servicio a través de los agentes que participan.

En [BSSB08] se presenta ConQo, un framework de descubrimiento que considera los atributos de calidad QoS, así como también la información del contexto de ejecución del servicio. Aunque existen muchas propuestas para incorporar las características QoS al descubrimiento y selección de servicios web, y otras para considerar la información del contexto de ejecución, no hay ninguna que tome en cuenta ambas categorías. Este trabajo utiliza WSMO para describir las propiedades de un servicio web. WSMO provee los conceptos para definir los aspectos funcionales y los requerimientos del servicio. Para describir las propiedades QoS se propone reusar la ontología del proyecto DIP. Y dada la imposibilidad de describir el contexto en WSMO, se define una ontología propia denominada *WSMO-context*. En adición a lo requerimientos del usuario, los proveedores pueden definir los requerimientos del ambiente en sus descripciones de servicios web. Los solicitantes de servicios web interactúan con la *Web Service Requester Interface* para enviar una descripción global al *Discovery Component* del proyecto DIP, para buscar en el repositorio de servicios web. El resultado de la correspondencia es una lista de todos los servicios web que satisfacen los requerimientos funcionales y no funcionales descriptos en el objetivo. Con el objetivo de incorporar la información del contexto, este componente de descubrimiento soporta el matching contra las características QoS y el contexto. Luego, el *Ranking Component* realiza un ranking sobre este conjunto de servicios web. Este componente verifica todos los parámetros obligatorios. Los servicios que no cumplen estos requerimientos esenciales son removidos del conjunto de servicios web aplicables. Finalmente, sobre el conjunto resultante el *Ranking Component* crea un puntaje final para cada servicio web. Al final, la respuesta al usuario que hizo el requerimiento es una lista ordenada de servicios web que satisfacen los requerimientos del usuario y los requerimientos del contexto. Ahora, el usuario puede invocar el mejor servicio web disponible.

En [ZHYJ07], los autores proponen extender la ontología OWL-S con un modelo QoS flexible que facilita el proceso de selección de servicios web. Basado en un conjunto de diferentes restricciones globales del usuario, el problema de selección de servicios basados en QoS se divide en tres categorías. Para cada categoría se presentan

separadamente un modelo matemático y su correspondiente algoritmo de solución. El modelo propuesto como extensión a OWL-S, OWL-S* presenta una solución más flexible, expresiva y aplicable para la representación de los distintos atributos de calidad QoS de una manera interpretable por la maquina.

Otro trabajo basado en WSMO es el de [VHA06], donde los autores introducen un enfoque para el descubrimiento semántico de los servicios web basado en registro P2P, teniendo en cuenta las características QoS. Las publicaciones de los servicios web semánticos se distribuyen entre los registros, de forma tal que es posible identificar rápidamente los repositorios que contienen el mejor servicio web disponible. Adicionalmente, se presenta información relevante para el proceso de descubrimiento usando *filtros Bloom* e información de correspondencia pre-computada, para que los esfuerzos de búsqueda sean minimizados cuando se consulta por servicios con ciertas características QoS y funcionales. El resultado de la consulta puede ser rankeado y los usuarios pueden proveer feedbacks a las actuales características provistas por un servicio web. Para evaluar la credibilidad de estos reportes de los usuarios cuando se predice la calidad de un servicio, se incluye un sólido mecanismo de administración de la reputación y confianza. La estructura de la descripción semántica de un servicio incluye:

- una especificación WSDL del servicio web
- una descripción funcional del servicio en términos de entradas, salidas, pre-condiciones, pos-condiciones y efectos, descripta con la ontología WSMO
- información opcional de las características QoS

Durante la operación del sistema esta información es cruzada con las consultas semánticas, que consisten de:

- los requerimientos funcionales del usuario en términos de entradas, salidas, pre-condiciones, pos-condiciones y efectos, también descripta con la ontología WSMO
- requerimientos QoS opcionales del usuario provistos como una tupla $\{q_i,n_i,v_i\}$, donde q_i es el parámetro QoS requerido, n_i es el orden de importancia de q_i en la consulta y v_i es el valor mínimo requerido por el usuario para este atributo.

Como se usa un sistema P2P como la infraestructura subyacente, este sistema escala en términos de número de registros, búsqueda eficiente, numero de propiedades en la descripción de un servicio y numero de usuarios.

En [D'Ambrogio06] los autores introducen una ligera extensión a WSDL para incorporar la descripción de las características de calidad QoS de un servicio web. Esta extensión es llevada a cabo como un metamodelo de transformación, de acuerdo a los

principios y estándares recomendados por la MDA (Model Driven Architecture). Se introduce un metamodelo WSDL que luego es transformado en un metamdelo Q-WSDL. Le extensión propuesta puede ser usada efectivamente para especificar los requerimientos QoS, establecer los SLA (service level agreements) y agregar características QoS cuando se consultan los registros de servicios web y soportar el mapeo automático desde documentos WSDL a Q-WSDL y desde modelos UML a servicios web Q-WSDL.

En [Kokash05], la autora presenta un framework para mejorar el descubrimiento de los servicios web. En este modelo, cada cliente o grupo de clientes locales con preferencias similares tiene asignado un *Personal Agent* (PA), cuya tarea es procesar todas las actividades pertenecientes a los servicios web (comunicación con los registros, bindings, requerimientos, respuestas). El PA acepta los requerimientos de sus clientes, y envía la descripción del objetivo al *Matching Agent*, el cual administra el registro de los servicios web disponibles. El *Matching Agent* busca todos los servicios que pueden satisfacer el objetivo del cliente. Este mantiene una historia de los requerimientos y le provee al solicitante una lista de de los servicios web relevantes y los agentes que han buscado anteriormente esos servicios web. El PA pide información a estos agentes y, si los servicios web usan los *Service Mediators* (SMs) para recopilar estadísticas sobre sus invocaciones, el PA puede obtener datos más fiables mediante la comparación de los parámetros provistos por los agentes y el *Service Mediator.* El *Matching Agent* provee al PA la puntuación correspondiente a sus requerimientos y la lista de servicios web retornados. Si la puntuación es baja, la relevancia de los servicios web es dudosa y se necesitan enfoques alternativos para chequear si ellos se corresponden con los requerimientos del cliente. El registro de las entradas y salidas de las invocaciones puede ayudar a evaluar el servicio antes de que sea usado por clientes nuevos.

El PA establece un ranking de servicios web combinando las puntuaciones del matching, las recomendaciones y las evaluaciones. Finalmente, se invoca el mejor servicio web, se miden los parámetros QoS, y se graban los resultados.

2.3. Búsqueda y selección dinámica de servicios web

Los trabajos antes mencionados proponen alternativas para implementar la comunicación de los flujos de trabajo con las aplicaciones de clientes y las aplicaciones externas. Las

propuestas que se basan en la utilización de servicios web presentan algunos inconvenientes al momento de elegir el servicio web que satisface los requerimientos del cliente, dados por las características de funcionamiento del protocolo UDDI (Universal Description, Discovery and Integration), actualmente utilizado para dicha selección. Estos inconvenientes están dados por el hecho de que el registro del servicio web no tiene en cuenta su semántica. El protocolo UDDI publica las descripciones de los servicios permitiendo a los usuarios registrar sus archivos WSDL con la información del servicio en forma de texto explicativo y palabras clave. Las palabras clave son una forma de describir la semántica del servicio web pero no aseguran que el resultado de una consulta no esté fuertemente influenciado por los términos de búsqueda elegidos.

Actualmente, el desarrollador es el encargado de resolver estos problemas semánticos. Cuando esta desarrollando un programa que utiliza un servicio web, el desarrollador consulta manualmente un servidor UDDI y selecciona el servicio web adecuado de acuerdo a su descripción en lenguaje natural. La información del servicio web y su invocación es codificada en el cliente y el descubrimiento en tiempo de ejecución se restringe, a lo sumo, a descubrir proveedores alternativos que ofrezcan exactamente el mismo servicio. Se han propuesto distintas alternativas para solucionar estos inconvenientes, las cuales se presentan a continuación.

En [HCH04] se presenta un enfoque formal para la selección automática de servicios web, basado en la correspondencia de los requerimientos del solicitante del servicio con la descripción de los servicios ofrecidos por el proveedor. Focaliza en el chequeo del comportamiento funcional de los contratos de las operaciones, especificando las precondiciones y los efectos de las operaciones requeridas y las ofrecidas. Utiliza reglas de transformación de grafos con condiciones positivas de aplicación como una notación visual y formal para definir los contratos, y poder determinar la compatibilidad semántica y sintáctica entre los requerimientos del solicitante y las ofertas el proveedor.

En [CHH04] los mismos autores del trabajo anterior avanzan en su propuesta de formalizar la selección automática de servicios web, usando reglas de transformación de grafos con condiciones positivas y negativas, y se concentran en chequear la compatibilidad del comportamiento. A partir de la relación semántica encontrada entre el solicitante y el proveedor, avanzan en probar la correctitud y completitud de esta relación.

En [HHL03] se propone solucionar el problema de la selección dinámica y en tiempo de ejecución de los servicios web usando reglas de transformación de grafos. El proceso de descubrimiento depende del entendimiento semántico de los servicios ofrecidos, pero

actualmente, este descubrimiento se desarrolla manualmente y en tiempo de desarrollo. Las reglas de transformación de grafos proveen una especificación semántica precisa, necesaria para automatizar el descubrimiento de los servicios de una manera visual e intuitiva.

En [HHL04] los autores presentan un enfoque visual basado en el uso de modelos de software y transformación de grafos, que permite la descripción de los servicios y además provee un concepto preciso de correspondencia. Presentan también una implementación usando estándares y herramientas disponibles en la actualidad.

En [HHL05] se propone modelar la semántica de los servicios web combinando ontologías y reglas de transformación de grafos. Las ontologías se usan para especificar la semántica del servicio web, representándolas de manera visual a través de diagramas de clases UML. Las reglas proveen una especificación semántica precisa, necesaria para automatizar el descubrimiento de los servicios de una manera visual e intuitiva.

En [HM05] los autores plantean la dificultad de incorporar la prueba automática para validar los servicios web antes de permitir su registro. Primero, el descubrimiento automático de un servicio web genera casos de prueba de conformidad a partir de la descripción del servicio a proveer, luego se corren estos casos de prueba sobre el servicio web considerado, y solamente si la prueba es pasada satisfactoriamente, el nuevo servicio web se registra. De esta manera, los clientes obligan a lo servicios web ofrecidos a proveer una signatura compatible, un comportamiento adecuado, y una implementación de alta calidad.

2.4. Algunas conclusiones sobre los trabajos analizados

Con respecto a las implementaciones alternativas de la comunicación del SGFT con las aplicaciones externas, las propuestas presentadas en la sección 2.1 requieren, en general, basarse en un modelo subyacente o una forma unificada de definir los componentes principales del sistema, para conseguir la interoperabilidad entre los SGFT. Pero esta integración es muy dificultosa dada la alta heterogeneidad y autonomía de los SGFT. Existen gran cantidad de plataformas y lenguajes de implementación diferentes, lo cual dificulta enormemente la integración.

Como otra alternativa se plantea el uso de la tecnología grid, que, si bien resulta muy útil para aplicaciones que requieren cantidades extraordinarias de recursos

computacionales, no está claro que esta solución sea aplicable en el campo de las aplicaciones de empresas de negocio en general, donde no se requiere una súper potencia de cálculo. La computación grid parece adecuada sólo en usos muy específicos y limitados, y sus beneficios no pueden ser fácilmente transferidos a cualquier tipo de empresa. Además, al igual que con cualquier otra tecnología, los beneficios deben sopesarse con el costo de su implementación, que conlleva el aumento en la complejidad de la gestión de los recursos, y el significativo costo de la implementación de esta tecnología en sí. Todo esto conspira para limitar el mercado de esta arquitectura a las grandes empresas con enormes necesidades computacionales.

Otros trabajos proponen el uso de ontologías para describir la semántica de los servicios web. Pero, mientras que algunos tipos de información son eminentemente adecuados para ser representados por medio de ontologías, otros puede que no lo sean tanto. Hay quienes han argumentado que las ontologías no son apropiadas para muchas aplicaciones del mundo real, una vez que ellas adquieren un cierto nivel de complejidad. En otras palabras, existe una brecha entre la expresividad y la usabilidad de las ontologías. Por otra parte, resultan difíciles de mantener y materializan un punto de vista particular del dominio de conocimiento.

Finalmente, el lenguaje AELCWS de la última propuesta, tal cual lo indican los autores del trabajo, no soporta la interacción directa con las personas durante la ejecución de un proceso, con lo cual la Interfaz de Aplicaciones de Cliente no esta definida ni implementada, y surgen algunas dudas acerca de cómo se manejaría la invocación dinámica de aplicaciones externas al SGFT, desde la Interfaz de las Aplicaciones Invocadas.

Con respecto a la incorporación de las características de calidad en la descripción de los servicios web, el principal inconveniente que se observa en las propuestas analizadas en la sección 2.2 es que las mismas requieren modificar el modelo subyacente de descubrimiento de servicios web. Por ejemplo, en la propuesta de [Ran03], la incorporación de un *Certificador QoS* en el modelo de descubrimiento de servicios web existente requiere introducir una modificación a la arquitectura corriente, de manera tal que el registro del servicio en UDDI ocurra únicamente después de que el proveedor del servicio acuerde con el Certificador. Otra desventaja es la redundancia en el desarrollo de medidas QoS, las cuales deben ser provistas por el proveedor del servicio al momento de registrarlo, mientras que el *Certificador QoS* eventualmente desarrollará otras medidas QoS para verificar lo expuesto por el proveedor, y así, las medidas se realizan al menos

dos veces. También, aunque este modelo incorpora los atributos de calidad en el registro UDDI, no provee los procedimientos o algoritmos necesarios para realizar la correspondencia de servicios y evalúo de atributos, ni tampoco incorpora al proceso de descubrimiento los conocimientos del consumidor.

Finalmente, las propuestas analizadas para la búsqueda y selección automática de servicios web teniendo en cuenta su significado semántico, presentadas en la sección 2.3, resultan muy interesentes. De hecho, las propuestas de [CH04] y [HHL03], se han tomado como base en el presente trabajo. Sin embargo, su uso no esta planteado en el marco de los SGFT, y es aquí donde el presente trabajo realiza un aporte.

Otros trabajos proponen el uso de ontologías predefinidas, pero otra vez, estas soluciones no son un mecanismo flexible y contradicen la naturaleza dinámica de la interacción basada en la web.

CAPÍTULO 3.

Sistemas de Gestión de Flujos de Trabajo

En este capítulo se presenta una visión general del contexto tecnológico en el que se sitúa el trabajo desarrollado en esta Tesis, respecto de las tecnologías de flujos de trabajo (comúnmente denominados workflows) y los Sistemas de Gestión de Flujos de Trabajo. Se realiza una revisión que abarca su definición, clasificación, conceptos básicos y esfuerzos actuales de estandarización.

El capítulo se estructura como sigue: en la sección 3.1 se introducen los flujos de trabajo y los Sistemas de Gestión de Flujo de Trabajo. En la sección 3.2 se presentan las definiciones oficiales de flujo de trabajo y Sistemas de Gestión de Flujo de Trabajo, según la Workflow Management Coalition (WfMC). La sección 3.3 presenta los diferentes tipos de flujos de trabajo, describiendo sus categorías según el tipo de proceso y su naturaleza. La sección 3.4 introduce los términos básicos de los flujos de trabajo y sus relaciones, de acuerdo a la WfMC. Finalmente, la sección 3.5 describe la estandarización propuesta por

la WfMC, denominada Modelo de Referencia de Workflow, y detalla las cinco interfaces de dicho Modelo.

3.1. Introducción

Las tecnologías de flujos de trabajo o workflows han tenido un crecimiento notable en gran cantidad de industrias. Hay cada vez más productos que usan flujos de trabajo para planear y ejecutar los procesos de negocio.
La característica principal que tienen los sistemas de flujo de trabajo es la automatización, total o parcial, de los procesos de negocio. La Workflow Management Coalition (WfMC) define un proceso de negocio de la siguiente manera:

> ***"Un proceso de negocio es un conjunto de uno o más procedimientos o actividades directamente ligadas, que colectivamente realizan un objetivo del negocio, normalmente dentro del contexto de una estructura organizacional que define roles funcionales y relaciones entre los mismos".***
>
> ***Workflow Management Coalition [WfMCa]***

Esta automatización de los procesos de negocio se basa en la interpretación por parte de un motor del flujo de trabajo o motor workflow de un conjunto de reglas de procedimiento. Estas reglas definen el conjunto de tareas a realizarse en un proceso de negocio, cuáles de ellas son automáticas, cómo es la interacción entre los recursos humanos y las máquinas, cómo es la secuencia de las actividades en el flujo de trabajo, etc. Normalmente comprende un cierto número de pasos lógicos, donde cada uno de estos es conocido como una actividad. Una actividad puede involucrar interacción manual con un usuario, o participante del flujo de trabajo, o puede ser ejecutada usando computadoras como recursos. A su vez, existen sistemas que permiten definir, crear y manejar la ejecución del flujo de trabajo, denominados Sistemas de Gestión de Flujo de Trabajo (SGFT).

Todos los Sistemas de Gestión de Flujo de Trabajo exhiben ciertas características en común, las cuales proveen una base para el desarrollo de capacidades de integración e

interoperabilidad entre diferentes productos. En el nivel más alto se caracterizan por proveer tres áreas de funcionalidades:

- Funciones en tiempo de construcción: dedicadas a la definición y modelado de un flujo de trabajo junto con todas sus actividades concernientes.
- Funciones de control en tiempo de ejecución: dedicadas al control del flujo de trabajo en el ambiente de ejecución, llevando a cabo cada tarea o actividad definida como parte del proceso.
- Funciones de interacción en tiempo de ejecución: dedicadas a la interacción con los usuarios y/o aplicaciones externas para que los participantes del flujo de trabajo puedan llevar a cabo sus tareas o actividades.

En la ejecución de estas funcionalidades participa uno de los componentes más importantes de un Sistema de Gestión de Flujos de Trabajo, su motor. Este es el responsable de interpretar la definición de procesos, interactuar con los participantes del flujo de trabajo cuando es requerido, e invocar el uso de herramientas y aplicaciones de tecnología de información.

3.2. Definición

La Workflow Management Coalition (WfMC) define un flujo de trabajo de la siguente manera:

> ***"Un flujo de trabajo es la automatización de los procesos de negocio donde los documentos, la información y las tareas se pasan entre los participantes del sistema de acuerdo a un conjunto de reglas previamente establecidas, con el fin de culminar una meta común impuesta por la empresa".***
>
> ***Workflow Management Coalition [WfMCa]***

Se puede ver al *flujo de trabajo* como un conjunto de métodos y tecnologías que ofrecen las facilidades para modelar y gestionar los diversos procesos que ocurren dentro de una empresa. Es un proceso manejado por un programa de computadora que asigna el trabajo, sus pasos, y sigue su progreso. El flujo de trabajo es el último, de una gran

línea de facilidades propuestas en respuesta de las exigencias de las organizaciones, las cuales apuntan a poder reaccionar tan rápido como sea posible ante la frenética demanda de la competición.

Los *Sistemas de Gestión de Flujo de Trabajo* (SGFT) son sistemas que definen, crean y administran la ejecución de los flujos de trabajo a través del uso de software que se ejecuta sobre uno o más motores de ejecución del flujo de trabajo, los que interpretan la definición de los procesos, interactúan con los otros participantes del flujo de trabajo e invocan herramientas y aplicaciones [WfMCa].

Las tendencias imperantes en el mundo de los negocios obligan a las empresas a repensarse a sí mismas y rediseñar sus soportes tecnológicos, con el fin de aumentar la productividad, mejorar la calidad, mejorar el servicio al cliente, reducir los costos y adaptarse a un entorno cambiante. En un SGFT:

- el trabajo no se extravía o detiene,
- los gerentes pueden enfocarse en el personal y problemas de negocio,
- los procedimientos son formalmente documentados,
- la mejor persona (o máquina) es asignada para hacer cada trabajo,
- el procesamiento paralelo es práctico.

El trabajo es realizado por el mejor participante, es decir, se distribuye el trabajo, y existe un supervisor quien puede influir en esta asignación automática. Además, el sistema necesita conocer qué trabajo espera ser asignado. Los SGFT permiten representar los procedimientos de trabajo mediante el flujo de documentos electrónicos, transferir estos documentos de un empleado a otro, de acuerdo a las reglas del negocio, y registrar los datos en una base de datos administrativa, con fines de medición y seguimiento.

Por su parte, los flujos de trabajo permiten ligar las actividades y aplicaciones que pertenecen a un mismo proceso, apoyar la coordinación de las personas, dar seguimiento a las tareas, reconfigurar procesos sin tocar los sistemas, y evaluar la efectividad en el cumplimiento de los compromisos. Los flujos de trabajo tienen como misión apoyar procesos estructurados orientados a la administración caso a caso, en los que intervienen varios actores. Por lo tanto, los flujos de trabajo no son candidatos para procesos altamente automatizados e intensivos en transacciones que requieren escasa intervención

humana, ni para procesos colaborativos no repetitivos, creados para un número limitado de ocurrencias.

Muchos productos desarrollos en el mercado de la Tecnología de la Información (TI) han soportado aspectos de los flujos de trabajo desde hace años, aunque hace relativamente poco ha sido reconocida su importancia. Algunas de las tecnologías de software que contribuyeron a la evolución de los flujos de trabajo son:

- *Procesamiento de imágenes:* Se captura en forma de imagen electrónica la información o documento que se desea (por ejemplo mediante un escáner), para luego ser pasada entre los diferentes participantes con distintos propósitos, durante la realización de un proceso.
- *Administración de documentos:* Esta tecnología esta relacionada con la administración del ciclo de vida de los documentos. Incluye facilidades para guardar en un depósito común aquellos documentos que se comparten, y facilidades para el acceso o modificación de los mismos mediante un conjunto predefinido de reglas.
- *Correo Electrónico y Directorios:* El sistema de directorios no sólo provee una forma de identificar a los participantes dentro de un conjunto de direcciones de correo electrónico, sino que ofrece además la potencialidad de registrar la información sobre los participantes, es decir, roles dentro de la empresa u otros atributos.
- *Aplicaciones basadas en transacciones:* Las transacciones de Flujo de trabajo guardan la información, reglas, roles, y otros elementos sobre un servidor de Bases de Datos Relacionales, ejecutando la aplicación de Flujo de trabajo sobre una interfaz gráfica para los usuarios.
- *Procesamiento de Formularios:* El ambiente de los formularios es amigable y familiar para muchos usuarios. Éste es un excelente vehículo para el manejo de la información dentro de una aplicación de Flujo de trabajo. Algunos productos para implementar aplicaciones de Flujos de trabajo proveen constructores de formularios, o se integran a constructores de terceros.

Es evidente que el contar con un SGFT proporciona grandes beneficios a las organizaciones que lo emplean. Estos beneficios no redundan únicamente en el ahorro de tiempo en el manejo de papeles, que en un principio era uno de los grandes problemas a resolver. Son varios los puntos a favor del uso de la tecnología de flujos de trabajo:

- Mejoran la calidad y rapidez del servicio,
- mejoran el uso y la oportunidad de la información,
- mejoran el control de los procesos,
- proveen flexibilidad organizacional,
- proveen diferenciación en el mercado,
- permiten la eliminación de trabajo.

Un sistema para la automatización de los procesos de negocios mejora el control de procesos, con mucha menos intervención de administración, y mucha menos chance de retrasos o trabajos faltantes. Además, mejora la calidad de los servicios, respondiendo más rápidamente y con el mejor personal obtenible, y disminuye el costo de instruir al personal, dado que el trabajo puede ser guiado mediante procedimientos complejos. Esto permite a los gerentes concentrarse en la educación de los empleados y dirigir casos especiales en lugar de rutinas de reportes y distribuciones.

También, un sistema para la automatización de los procesos de negocios mejora la satisfacción de los usuarios. Por lo tanto, *un sistema de gestión automatizado es bueno para la empresa, para el cliente y para los usuarios.*

3.3. Diferentes tipos de flujos de trabajo

La diversidad de procesos de negocio existentes hace necesario dividir los flujos de trabajo en varias categorías según el valor del proceso a manejar y si este proceso es repetitivo o no. Un proceso tiene un valor alto si representa grandes ahorros a una empresa o la relación costo-oportunidad es buena, en definitiva un proceso tiene alto valor si redunda en grandes beneficios para una empresa.

Teniendo en cuenta las consideraciones anteriores, actualmente se encuentran tres tipos de flujos de trabajo:

- *De Producción*: modelan e implementan los procesos de negocio críticos de la organización. Son sistemas complejos que se ejecutan sobre entornos heterogéneos y en los que suelen participar gran variedad de personas y organizaciones. Frecuentemente se requiere la ejecución de transacciones para el acceso a la información. Este tipo de flujos de trabajo es el segmento más grande en el mercado.

En general automatizan procesos de negocio que tienden a ser repetitivos, bien estructurados y con gran manejo de datos. Por ejemplo la integración de software preexistente (*legacy applications*), los préstamos bancarios, préstamos y devoluciones en general, seguros, etc.

- *De Colaboración*: Las aplicaciones de flujos de trabajo que resuelven procesos de negocio donde participan distintas personas para lograr una meta común, son llamadas *Flujos de Trabajo de Colaboración.* Los flujos de trabajo de colaboración estructuran o semi-estructuran procesos de negocio donde la mayor parte de la coordinación es realizada por el hombre, y el sistema se limita a proporcionar una buena interfaz para las interacciones. Típicamente involucran documentos, los cuales son los contenedores de la información, se sigue la ruta de estos paso a paso además de las acciones que se toman sobre ellos. Los documentos son la clave. Es esencial para la solución de flujo de trabajo mantener la integridad de los documentos. Son sistemas muy dinámicos que en muchas ocasiones se definen conforme se avanza en el proceso.
- *Administrativo*: es aquel que involucra procesos de administración en una empresa, tales como órdenes de compra, reportes de ventas, etc. La función básica asociada al mismo es el procesamiento de formularios. El SGFT se encarga de activar la ejecución de las actividades, recoger las respuestas (los datos) y obtener el formulario (una o varias actividades automáticas, que recopilan y procesan la información obtenida). Por ejemplo, la matrícula universitaria o la obtención de certificados. Las actividades involucradas son repetitivas y de baja complejidad. Se emplea Flujo de Trabajo Administrativo si se cumplen ciertas condiciones:

    (1) Hay gran cantidad de procesos de administración dentro de la empresa. Por esto la aplicación de Flujo de Trabajo utilizada debe poder manejar gran cantidad de procesos.

    (2) Una solución de Flujo de Trabajo Administrativo difiere para cada organización, y los cambios son frecuentes. Por esto, la posibilidad de poder hacer cambios de diseño es muy importante.

    (3) Toda persona en la organización es un potencial participante, por lo que es importante tener la posibilidad de distribuir la solución a un gran número de usuarios sin mucho esfuerzo.

- *Ad-Hoc*: estos sistemas son muy similares a los administrativos, pero con la característica de utilizarse para tratar situaciones únicas o excepcionales, en lugar de procesos burocráticos perfectamente establecidos. El SGFT informa sobre el estado de ejecución de cada actividad y la participación humana es fundamental. Las actividades involucradas en el proceso suelen ser únicas, en el sentido de que muchas de ellas sólo se realizan una vez y su complejidad oscila de baja a media. La principal dificultad en este tipo de sistemas está dada en la construcción del grafo de coordinación y cooperación entre actividades.

3.4. Conceptos básicos de un flujo de trabajo

De acuerdo a la definición de la WfMC, en un SGFT existen dos actividades bien diferenciadas. Por una parte está la definición del flujo de trabajo que implementa el proceso de negocio, lo que se denomina modelado del flujo de trabajo. Por otra parte está la ejecución de dicho modelo. Los términos básicos de un flujo de trabajo y sus relaciones, tanto para la fase de modelado como para la fase de ejecución, se ilustran en la Figura 3.1.

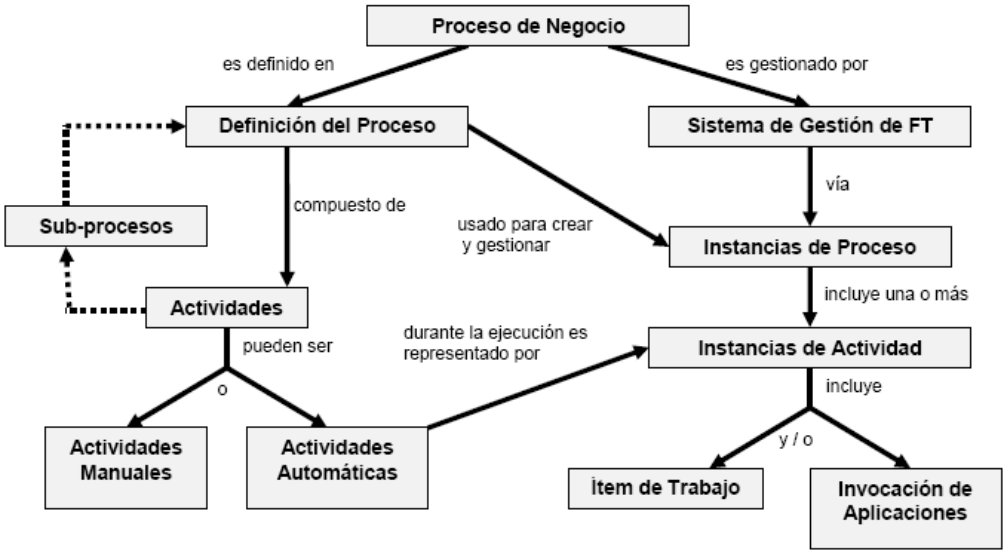


Figura 3.1. Términos principales y sus relaciones.

El modelado del flujo de trabajo consiste en la definición de un conjunto de actividades relacionadas, y un conjunto de criterios que indican el comienzo y la finalización del

proceso. Además, se definen sus componentes, e información adicional sobre cada actividad, tal como participantes, invocación de aplicaciones, datos, etc. En el modelado participan los siguientes elementos:

- *Proceso de Negocio:* un conjunto de uno o más procedimientos o actividades que colectivamente realizan un objetivo o política global de una organización.
- *Definición del Proceso:* la representación de un proceso de negocio en una forma que soporta la manipulación automática. La definición consiste de un conjunto de actividades y sus relaciones.
- *Actividades:* la descripción de un trozo de trabajo que forma un paso lógico dentro de un proceso.
- *Actividades Automáticas:* las que soportan la automatización por computadora.
- *Actividades Manuales:* las que no soportan la automatización por computadora y por lo tanto quedan fuera de alcance del sistema de FT.

La ejecución del flujo de trabajo en un SGFT consiste en la creación, manipulación y destrucción de instancias de proceso, que representan al flujo de trabajo de acuerdo a la especificación realizada en el modelado. Cada instancia de proceso tendrá una identidad visible externamente y un estado interno que representa su progreso hacia la finalización y su estado con respecto a las actividades que lo componen. Cada vez que la ejecución de un proceso implique la invocación de una actividad, se crea una instancia de actividad que la representa. Dicha instancia se encarga de ejecutar las acciones asignadas, accediendo a los datos que sean necesarios e invocando la aplicación externa correspondiente, si así lo requiere la actividad. Los elementos que participan en la ejecución son:

- *Instancias de Proceso:* representan un proceso simple, incluyendo sus datos asociados.
- *Instancias de Actividad:* representan una actividad dentro de una instancia de proceso.
- *Ítem de Trabajo:* es la representación de una tarea a ser procesada por un participante del flujo de trabajo, en el contexto de una actividad dentro de una instancia de proceso. Cada tarea es un conjunto de acciones manejadas como una sola unidad. Generalmente son desempeñadas por una única persona dentro de los roles que

pueden realizar dicha tarea. Las tareas surgen del análisis del flujo del trabajo, donde se define por quienes deben ser ejecutadas.

- *Invocación de Aplicaciones:* es la representación de las aplicaciones del flujo de trabajo que son invocadas por el SGFT para automatizar una actividad.

3.5. El Modelo de Referencia de la WfMC

Debido a la gran diversidad de productos de flujo de trabajo ha sido necesario tener un lenguaje común que permita la interoperabilidad entre estos productos. Pero, a pesar de la gran variedad de productos que se encuentran en el mercado, se puede ver que los conceptos y terminologías utilizadas no varían en gran forma. Esto permite que se tienda a realizar un modelo de implementación general. Actualmente se busca identificar los principales componentes de un sistema de flujo de trabajo, de modo tal de poder volcarlos dentro de un mismo modelo abstracto. Es necesaria la representación formal de un modelo que permita la realización de sistemas sobre diversos escenarios, de forma tal de tener la posibilidad que distintos sistemas de flujo de trabajo puedan interactuar entre sí.

La WfMC ha estandarizado la automatización de los procesos de negocio. Dicho estándar define un marco genérico para la construcción de sistemas de gestión de flujos de trabajo, permitiendo la interoperabilidad entre ellos y con otras aplicaciones involucradas. Dado que el objetivo de la WfMC es definir estándares y guías globales para el desarrollo de sistemas de gestión de flujo de trabajo, su documentación no describe en detalle ningún sistema en particular, sino que brinda un marco genérico a seguir en la construcción de este tipo de sistemas. Este estándar, denominado Modelo de Referencia de Workflow [WfMCa], ha sido desarrollado tomando como partida la estructura genérica de las aplicaciones de flujo de trabajo, e identificando en esta estructura las interfaces que permiten a los productos interoperar a distintos niveles. Todos los sistemas de gestión de flujos de trabajo contienen varios componentes genéricos que interactúan entre sí de forma perfectamente definida. Diferentes productos exhiben diferentes niveles de capacidad dentro de cada uno de estos componentes. Para lograr la interoperabilidad entre los diferentes productos de flujo de trabajo es necesario definir un conjunto de interfaces y formato de datos estándar.

La Figura 3.2 ilustra los principales componentes y sus interfaces en la arquitectura de un Sistema de Gestión de Flujo de Trabajo.

En el modelo adoptado hay una separación entre los procesos y el control de la lógica de las actividades. Esta lógica esta dentro del Servicio de Ejecución del Flujo de Trabajo. Esta separación permite la integración de las diversas herramientas con una aplicación particular.

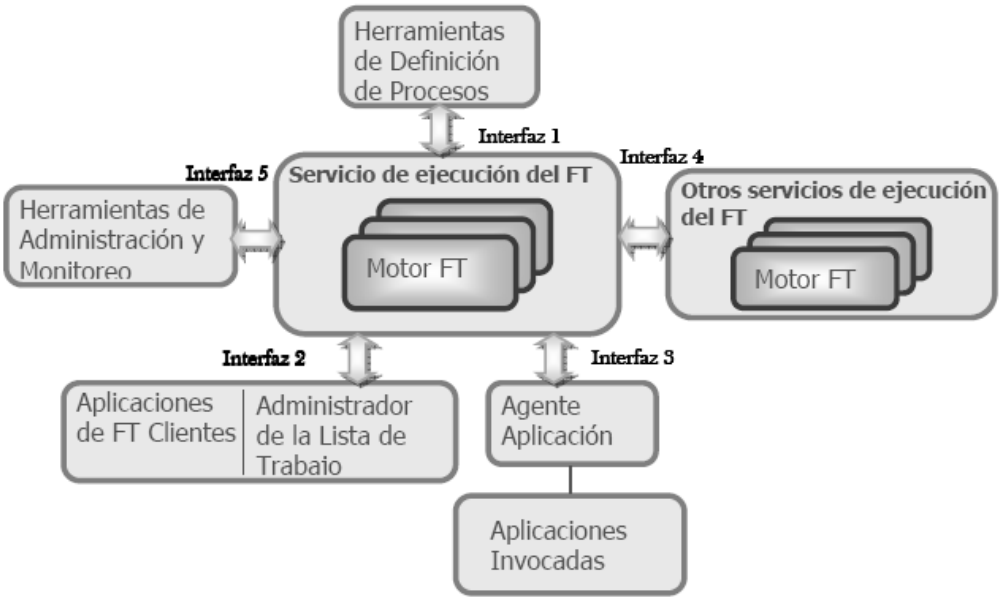


Figura 3.2. Modelo de Referencia de SGFT. Componentes e Interfaces.

Servicio de Ejecución del Flujo de Trabajo (Workflow Enactment Service)

Es el componente central, quien interpreta la descripción de los procesos y controla sus diferentes instancias, secuencia las actividades, adiciona elementos a la lista de trabajo de los usuarios, e invoca las aplicaciones necesarias. En este componente se encuentra el o los motores del flujo de trabajo, los cuales manejan la ejecución propiamente dicha, de cada instancia del proceso. La interacción del Servicio de Ejecución del Flujo de Trabajo con los recursos externos se da por una de las dos siguientes interfaces:

- La Interfaz de las Aplicaciones de Cliente, a través de la cual el motor del flujo de trabajo interactúa con el manejador de la lista de trabajo, responsable de organizar el trabajo por intermedio de un recurso de usuario. Es responsabilidad del manejador de la lista de trabajo elegir y hacer progresar cada elemento de la lista.
- La Interfaz de las Aplicaciones Invocadas, la cual le permite al motor del flujo de trabajo activar una herramienta para realizar una actividad particular. Esta interfaz podría basarse en un servidor, es decir, podría no existir la interacción con el usuario.

Lista de trabajo

La lista de trabajo (worklist) forma parte de los datos del flujo de trabajo, ya que la interacción con los usuarios es necesaria en algunos casos. El motor del flujo de trabajo utiliza una lista de trabajo manipulada por un manejador de la lista para controlar tal interacción. Esta lista es una cola de trabajo asignado a un usuario o a un grupo de usuarios por el motor del flujo de trabajo, quien deposita en la lista de trabajo los ítems de trabajo ha ser ejecutados por los mismos. La lista de trabajo puede ser visible o invisible para los usuarios dependiendo del caso, muchas veces se deja que el usuario seleccione ítems y los procese en forma individual.

Manejador de la Lista de Trabajo

Este componente de software maneja la interacción entre los participantes del flujo de trabajo y el servicio de ejecución del flujo de trabajo, vía la lista de trabajo. El manejador soporta en general un amplio rango de interacción con otras aplicaciones clientes. En algunos sistemas, la interfase con el usuario y el manejador de Worklist, están agrupadas como una única entidad funcional.

Motor del Flujo de Trabajo

El motor del flujo de trabajo (workflow engine) es el software que provee el control del ambiente de ejecución de una instancia de flujo de trabajo. Típicamente dicho software provee facilidades para:

- La interpretación de la definición de procesos,
- el control de las instancias de los procesos: creación, activación, terminación, etc.,
- la navegación entre actividades,
- el soporte de interacción con el usuario,
- el pasaje de datos a usuarios o aplicaciones,
- la invocación de aplicaciones externas.

Con respecto a las posibles implementaciones del servicio de ejecución del flujo de trabajo, el software consiste de uno o más motores de flujo de trabajo, los cuales son responsables del manejo de toda, o parte, de la ejecución de las instancias de los procesos. Este software puede ser implementado como un sistema centralizado con un único motor de flujo de trabajo, responsable del manejo de todas las ejecuciones de

procesos que existen en el sistema. La otra alternativa es una implementación como un sistema distribuido, en la cual varios motores cooperan, la complejidad es mucho mayor pero en general redunda en mayores beneficios. En el escenario distribuido varios motores cooperan en la ejecución de una instancia de un proceso, el control de datos asociado al proceso debe tener la capacidad de dialogar con diferentes motores. Este control de datos podría estar distribuido entre los motores o podría estar en un único motor (Motor maestro). El control de datos mantiene el estado de la información asociada a cada proceso, podría tener también *checkpoints* para ser usados en caso de fallas.

La definición de procesos, es usada para modelar la navegación entre los procesos, y provee información acerca de entradas a procesos y criterios a tomar en cada paso de la navegación, asigna tareas a usuarios, asigna aplicaciones a cada actividad, etc. La definición de procesos también podría realizarse en forma distribuida o centralizada. La implementación de la opción distribuida implica una gran complejidad al establecer la relación entre la definición de procesos y los motores.

3.5.1 Procesos y Transiciones de Estados de Actividad

El *Servicio de Ejecución del Flujo de Trabajo* puede ser considerado como una máquina de estados, donde los procesos cambian de estado según eventos externos, o decisiones de control específicas, tomadas internamente por el motor de flujo de trabajo.

Los procesos están constituidos por diversas actividades. La culminación de las actividades que constituyen un proceso, implica la culminación del proceso mismo.

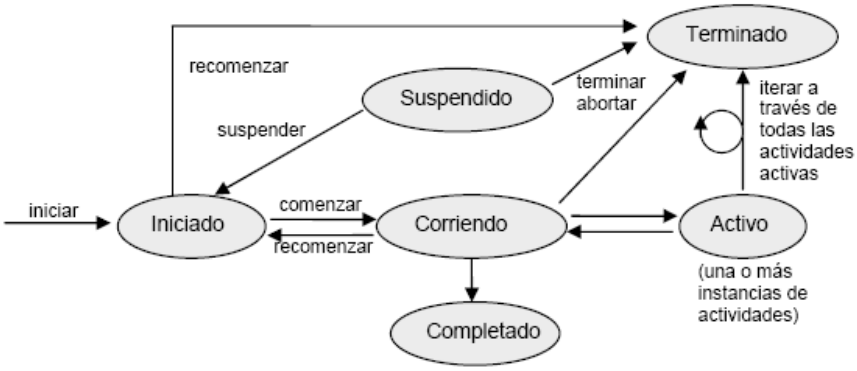


Figura 3.3. Transiciones de Estados de Actividad y Proceso

La Figura 3.3 ilustra los estados básicos dentro de un esquema de transición para la instancia de un proceso. Las transiciones entre los distintos estados están representadas por las flechas. Los estados básicos son:

- *Iniciado*: Ha sido creada una instancia del proceso, pero no se han dado las condiciones para su comienzo.
- *Corriendo*: Se comenzó la ejecución del proceso, y cualquiera de sus actividades podría comenzar.
- *Activo*: Una o más actividades del proceso comenzaron.
- *Suspendido*: La instancia del proceso está inmóvil y ninguna actividad es comenzada hasta que el proceso haya retornado al estado *corriendo*.
- *Completado*: El proceso culminó, se realizan las acciones programadas (auditoría) y luego se elimina la instancia del proceso.
- *Terminado*: La ejecución de la instancia del proceso se ha detenido antes de su realización normal.

Cuando se crea una instancia de un proceso, se crean a su vez instancias para las actividades que forman parte de ese proceso.

Ignorando ciertas complejidades como por ejemplo la atomicidad de las actividades, se puede hacer un diagrama de estados básico para una instancia de una actividad. En este caso los estados básicos son:

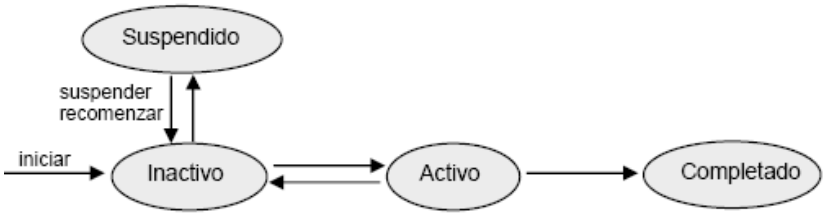


Figura 3.4. Transiciones de Estados de las Actividades

- *Inactivo*: La actividad dentro de la instancia del proceso ha sido creada pero no ha sido activada y no tiene ningún ítem de trabajo para procesar.
- *Activo*: Un ítem de trabajo ha sido creado y asignado a la instancia de actividad para su procesamiento.

- *Suspendido*: Se suspende la ejecución de la instancia de la actividad. A la misma no se le asignan ítems de trabajo hasta que no vuelve al estado *Inactivo*.
- *Completado*: La ejecución de la instancia de la actividad ha sido completada normalmente.

3.5.2 Workflow Application Programming Interface (WAPI)

Las WAPI pueden ser vistas como un conjunto de llamadas APIs (Application Programming Interfaces) y funciones de intercambio soportadas por el Servicio de Ejecución del Flujo de Trabajo, para su interacción con otros recursos y aplicaciones. Son un conjunto de llamadas a funciones de software que permiten a las aplicaciones acceder a funciones de un programa.

Por su parte, las Interfaces de Programación de Aplicaciones del Flujo de Trabajo (Workflow Application Programming Interface o WAPIs) permiten la interacción del servicio de ejecución del flujo de trabajo con otros recursos y aplicaciones. Estas agrupan un conjunto de llamadas APIs y sus formatos de intercambio relacionados, requeridas para soportar las cinco interfaces funcionales del Modelo de Referencia de Workflow. Las WAPIs permiten la interacción del Servicio de Ejecución del Flujo de Trabajo con otros recursos y aplicaciones.

3.5.3 Intercambio de Datos

Los datos son los documentos, archivos, imágenes, registros de la Base de Datos, y otros utilizados como información para llevar a cabo el trabajo. Entre los datos manejados por el sistema de flujo de trabajo encontramos:

- *Datos de Control:* son los datos internos que se manejan por el SGFT y/o un motor del flujo de trabajo.
- *Datos Relevantes:* son aquellos que son usados por el SGFT para determinar la transición de estado de una instancia de proceso, es decir, para determinar el ruteo de los distintos procesos del sistema.
- *Datos de Aplicación:* son los datos que están en una aplicación específica y no son accesibles por el SGFT.

El Servicio de Ejecución del Flujo de Trabajo mantiene el control interno de los datos, identificando el estado de las instancias de procesos o actividades. Estos datos no son accesibles, pero se puede obtener cierta información en respuesta de ciertos comandos (por ejemplo consultar los estados de los procesos, obtener métricas, etc.). Estos datos son los que utiliza el motor del flujo de trabajo para mantener el control de las diversas instancias de los procesos.

La manipulación de datos desde una aplicación puede ser requerida por alguna actividad en la definición de un proceso. El intercambio de datos relevantes y datos de aplicación es requerido a través de las WAPI para soportar la interacción de las tres funcionalidades siguientes en tiempo de ejecución:

- Manejador de la Lista de Trabajo (Interfaz 2)
- Aplicaciones Invocadas (Interfaz 3)
- Intercambio entre los motores del flujo de trabajo (Interfaz 4)

El intercambio directo de los datos de las aplicaciones es tipificado por sistemas de flujos de trabajo orientados al mail electrónico, en donde los datos son físicamente transferidos entre las actividades. En este caso no es necesario definir una relación explícita entre las actividades y la aplicación. En algunos casos es necesario proveer la conversión del formato de los datos entre las actividades. En este caso, la aplicación puede definir, como un atributo, el tipo de datos con el cual esta asociado el formato de los datos. Esto le permite a los sistemas de flujos de trabajo heterogéneos proveer la conversión de los datos sobre la base del tipo de datos de los mismos.

En el caso de un sistema de flujo de trabajo implementado por intermedio de documentos compartidos, no hay transferencia física de datos entre las actividades. En este tipo de sistema, los datos son accedidos in situ por la aplicación, usando la ruta de acceso apropiada. Las direcciones de los documentos deben mantenerse como variables globales en el sistema, de forma tal que los distintos servicios o motores puedan acceder a ellos. La conversión de datos puede ser modelada usando herramientas apropiadas para ello (por ejemplo un procesador de texto). Los sistemas homogéneos pueden usar convenciones privadas para el nombrado de los objetos y para el acceso a permisos; pero en sistemas heterogéneos se requiere de un esquema común.

3.5.4 Interfaz 1: Definición de Procesos

Las herramientas de definición del proceso o flujo de trabajo permiten modelar, describir y documentar un determinado flujo de trabajo, proceso de negocio o proceso general. Tales herramientas pueden variar desde las más informales (lápiz y papel), a las más formales y sofisticadas. La salida de este proceso de modelización y diseño es una *definición del proceso*, el cual será interpretado en tiempo de ejecución por el o los motores del flujo de trabajo. Esta definición especifica la lógica del proceso, las actividades que lo componen, los participantes humanos, aplicaciones invocadas, datos utilizados, etc. La Interfaz 1 permite la comunicación entre este componente y el servicio de ejecución del flujo de trabajo. Existe una gran cantidad de herramientas de definición de procesos (independientes de los productos en muchos casos), las cuales deben comunicarse con los motores de algún producto del flujo de trabajo. Lo deseable es que estas herramientas puedan comunicarse con cualquier motor, esto únicamente sería posible si se establecen ciertas normas de comunicación entre las herramientas de definición de procesos y el motor del flujo de trabajo. Por esto la WfMC propone una interfaz para esta comunicación.

Cuando un producto del flujo de trabajo proporciona su propia herramienta de definición de proceso, la definición de proceso resultante normalmente estará dentro del dominio del producto y puede o no ser accesible vía una interfaz de programación para lectura y escritura de la información. Cuando son usados productos separados para definir y ejecutar el proceso, la definición de proceso debe ser transferida entre los productos (como y cuando sea requerida).

El objetivo de esta interfaz es dar un formato de intercambio y llamadas a APIs, para soportar el intercambio de información en la definición de los procesos. El intercambio podría ser una definición completa de los procesos o un subconjunto del mismo.

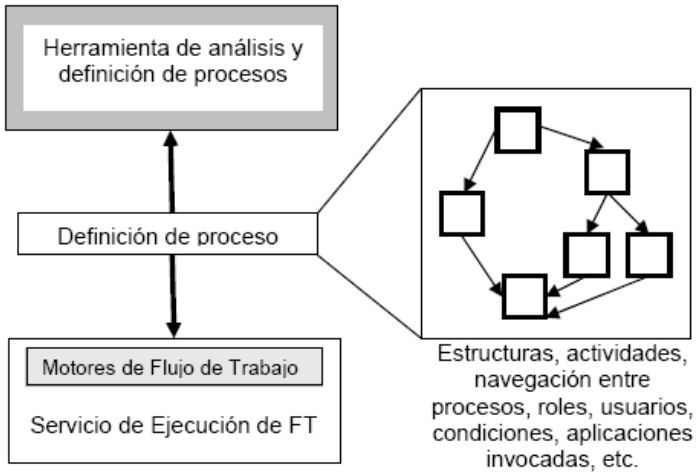


Figura 3.5. Componentes que participan en la Interfaz 1.

Un beneficio de usar una forma estandarizada de definición es que existe una separación entre el entorno de definición y el de ejecución del proceso. De este modo, es posible almacenar en un repositorio la información sobre la definición del proceso, para que puedan consultarla los distintos entornos de ejecución y ejecutar el proceso completamente o de forma distribuida. Además, esta forma estandarizada ofrece el potencial para exportar una definición de proceso a varios SGFT diferentes.

Con respecto a esta interfaz, la WfMC trabaja en dos aspectos fundamentales. Uno es la construcción de un meta-modelo para expresar los objetos, sus relaciones y atributos dentro de una definición de proceso. La especificación en notación UML [BRJ05] de este metamodelo del flujo de trabajo se puede apreciar en la Figura 3.6.

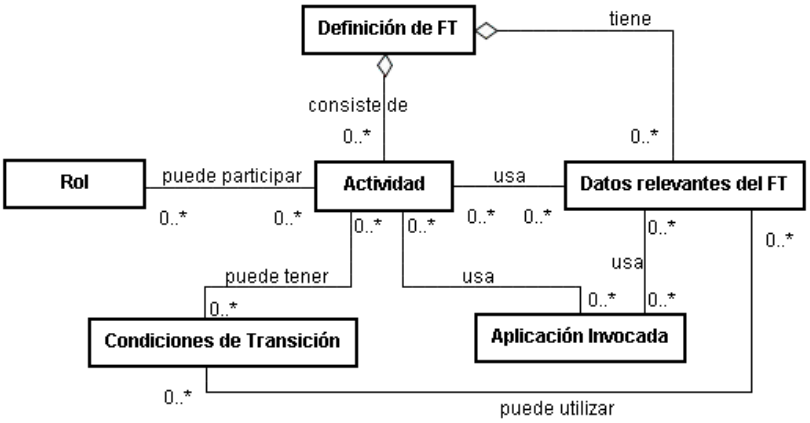


Figura 3.6. Meta-modelo básico para la definición de un flujo de trabajo

El otro aspecto se refiere a las llamadas APIs entre los SGFT, que proporcionan una manera común para acceder a la definición del flujo de trabajo. El enfoque propuesto por la WfMC en esta área consiste en trabajar con otros grupos para la definición de formatos de intercambios adecuados. Se esta desarrollado un conjunto de comandos API que soportan el acceso a datos de la definición del proceso. En la tabla 3.1 se detallan dichas APIs.

Categoría	APIs
Establecimiento de Sesión	– conexión y desconexión de sesiones entre los sistemas participantes
Operaciones de	– recuperación de listas de nombres de definición de procesos de FT desde

definición del FT	un repositorio u otra lista – selección/de-selección de una definición de proceso de FT para proveer un manejo de sesión para posteriores operaciones a nivel de objeto – lectura/escritura de definición de procesos de FT a un nivel superior
Operaciones de definición de Objetos del FT	– creación, recuperación y borrado de objetos dentro de una definición de FT – recuperación, configuración y eliminación de atributos de objeto

Tabla 3.1. APIs para acceder a la definición de procesos

3.5.5 Interfaz 2: Aplicaciones de Cliente

Una aplicación de cliente es un software que el participante del flujo de trabajo utiliza para la interacción con el SGFT, ejecutando las aplicaciones necesarias para llevar a cabo las actividades que tiene asignadas.

Además del propio servicio de ejecución del flujo de trabajo y las herramientas para la definición del proceso, existe el componente denominado *aplicaciones cliente del flujo de trabajo*, que representa estas entidades de software utilizadas por el participante del flujo de trabajo en aquellas actividades que requieren intervención humana para su realización. Si este componente se separa de lo que es el propio componente de ejecución, es necesaria una interfaz que defina y maneje claramente el concepto de lista de trabajo. Esta interfaz es la Interfaz 2. Parte de la información almacenada en la lista de trabajo es utilizada para trasmitirle al manejador de la lista de trabajo qué aplicaciones hay que invocar. La lista de trabajo puede contener ítems relacionados con diferentes instancias de un proceso o ítems de diferentes procesos. El manejador de la lista de trabajo puede interactuar con diferentes motores.

La aplicación de las especificaciones correspondientes a la Interfaz 2 del SGFT, permite que cualquier aplicación de cliente, ya sea que forma parte del mismo sistema, que forma parte de otro diferente, o sea una aplicación independiente, se comunique eficazmente con éste para solicitarle aquellos servicios que sean necesarios para procesar las actividades pertinentes. La interfaz entre una aplicación cliente y el motor del flujo de trabajo debe ser lo suficientemente flexible en los siguientes puntos:

- Identificadores de procesos y actividades,
- estructuras de datos,

- diferentes alternativas de comunicación.

En este escenario se alcanza un nivel simple de interoperabilidad a través del uso de formatos de intercambio de datos. Además, existe un enfoque global para las APIs de las Aplicaciones de Clientes. Las especificaciones API son publicadas en documentos separados de la WfMC.

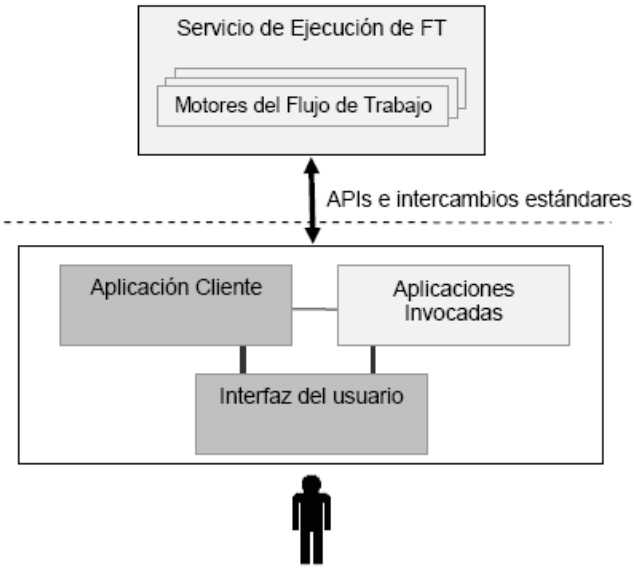


Figura 3.7. Componentes que participan en la Interfaz 2.

En la tabla 3.2 muestra las APIs asociadas a esta interfaz [WfMCb].

Categoría	APIs
Establecimiento de Sesión	– conexión y desconexión de sesiones entre los sistemas participantes
Operaciones de definición del FT	– funciones de recuperación y consulta sobre los nombres o atributos de la definición de procesos del FT
Funciones de Control de Proceso	– creación/comienzo/terminación de instancias de procesos individuales – suspensión/reanudación de una instancia de proceso individual – forzado del cambio de estado dentro de una instancia de actividad o proceso individual – asignación o consulta de un atributo (por ejemplo prioridad) de una instancia de actividad o proceso
Funciones de	– apertura/cierre de consultas sobre una instancia de actividad o proceso,

Estado de Proceso	estableciendo criterios de filtro opciones – búsqueda de detalles de las instancias de actividad o proceso, utilizando filtros – búsqueda de detalles de una instancia de actividad o proceso específica
Funciones de Manipulación de la Lista de Trabajo y los Ítems de Trabajo	– apertura/cierre de una consulta sobre la lista de trabajo, estableciendo criterios de filtro opcionales – búsqueda de ítems de la lista de trabajo utilizando filtros – notificación de selección/reasignación/terminación de un ítem de trabajo específico – asignación o consulta de un atributo de un ítem de trabajo
Funciones de Supervisión de procesos	– modificación del estado operacional de una definición de proceso de FT y/o sus instancias de proceso existentes – modificación del estado de todas las instancias de actividad o proceso de un tipo específico – asignación de atributo(s) a todas las instancias de actividad o proceso de un tipo específico – terminación de todas las instancias de proceso
Funciones de Manipulación de datos	– Recuperar /retornar datos relevantes de la aplicación o FT

Tabla 3.2. APIs para el uso de aplicaciones de clientes

Las funciones descriptas anteriormente proveen un nivel básico de funcionalidad para soportar la invocación de aplicaciones por parte del manejador de la lista de trabajo.

3.5.6 Interfaz 3: Aplicaciones Invocadas

El componente *aplicaciones invocadas* representa software o aplicaciones ya existentes que un SGFT puede utilizar para la realización de ciertas actividades, teniendo en cuenta que, en principio, dichas aplicaciones se pueden encontrar en cualquier plataforma o lugar de la red. La Interfaz 3 permite la comunicación entre este componente y el servicio de ejecución del flujo de trabajo, no sólo a nivel de invocación del mismo, sino de transformación de datos en formatos entendibles por ambos componentes. Una solución se obtiene a través de los *agentes de aplicación*, que permiten que el servicio de

ejecución del flujo de trabajo se comunique con las funciones estándar de dichos agentes de aplicación y éste define interfaces específicas para cada tipo de aplicación invocada.

La aplicación invocada es manejada localmente por un motor del flujo de trabajo, usando la información suministrada en la definición del proceso para identificar la naturaleza de la actividad, el tipo de aplicación a ser invocada y los requerimientos de datos. La aplicación que se invoca puede ser local al motor del flujo de trabajo, o sea, residente en la misma plataforma, o estar en otra plataforma dentro de una red. En este caso, la definición del proceso debe contener la información necesaria para poder encontrar la aplicación que se va a invocar (como ser la dirección dentro de la red).

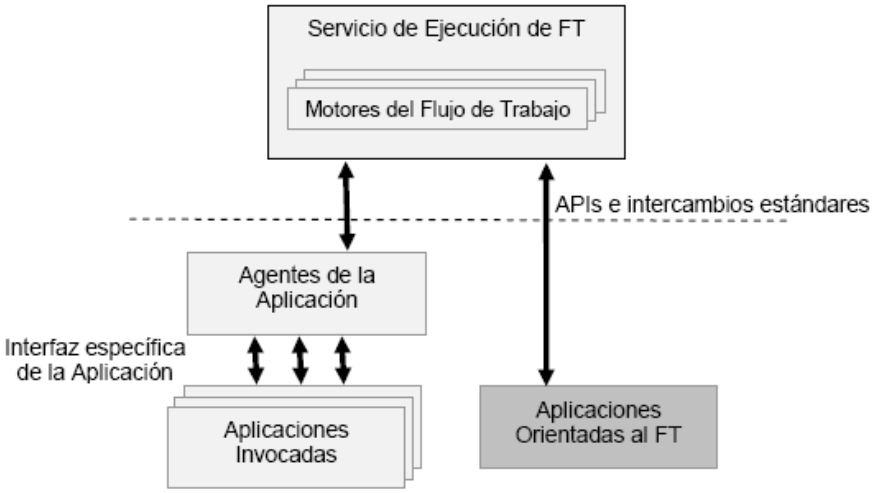


Figura 3.8. Componentes que participan en la Interfaz 3.

Hay SGFT que tratan con un rango más restrictivo de aplicaciones, aquellas donde los datos son fuertemente tipados y pueden estar directamente asociados con una herramienta particular, por ejemplo: Microsoft Word, Microsoft Excel. En otros casos la invocación de una operación puede lograrse a través de mecanismos de intercambio estándar tales como el protocolo OSI TP o X.400. Se pueden desarrollar herramientas que usen un conjunto de APIs estándar para comunicarse con el flujo de trabajo. Estas APIs permiten la comunicación eficaz entre el motor del flujo de trabajo y aquellas aplicaciones que éste ha de ejecutar para que sirvan de soporte a las actividades correspondientes. Son desarrolladas conjuntamente a las especificaciones de la Interfaz 2, pero cada una tiene sus propios objetivos. En la tabla 3.3 se muestran las APIs asociadas a esta interfaz [WfMCb].

Categoría	APIs
Establecimiento de sesión	– Conexión/desconexión de una sesión de aplicación o agente de aplicación
Funciones de Manipulación de Actividades	– Comienzo de una actividad (del motor a la aplicación) – suspender/reanudar/abortar una actividad (del motor a la aplicación) – notificación de actividad completa (de la aplicación al motor) – disparo de evento de señal (de la aplicación al motor) – consulta de atributos de la actividad (de la aplicación al motor)
Funciones de Manipulación de Datos	– provisión de datos relevantes del FT – provisión de datos de la aplicación

Tabla 3.3. APIs para la invocación de aplicaciones

3.5.7 Interfaz 4: Funciones de Interoperabilidad

La interoperabilidad entre los SGFT está representada por el componente denominado *servicio de ejecución del flujo de trabajo*, siendo la Interfaz 4 la que permite dicha comunicación. El trabajo de la WfMC se ha enfocado en desarrollar varios escenarios de interoperabilidad. La idea de estos escenarios es poder operar en diferentes niveles, desde la conexión a nivel de tareas simples hasta aplicaciones del flujo de trabajo con un nivel alto de interoperabilidad, con un intercambio completo de la definición de procesos, datos relevantes y una vista común de esto entre los distintos ambientes del flujo de trabajo. Se han identificado cuatro modelos posibles de interoperabilidad. A continuación se describen estos modelos.

Escenario 1: Conexión Discreta (Encadenado)

Este modelo permite a un punto de conexión del proceso A conectarse con otro punto en el proceso B. Los puntos de conexión pueden estar en cualquier parte de los procesos donde tenga sentido establecerla.

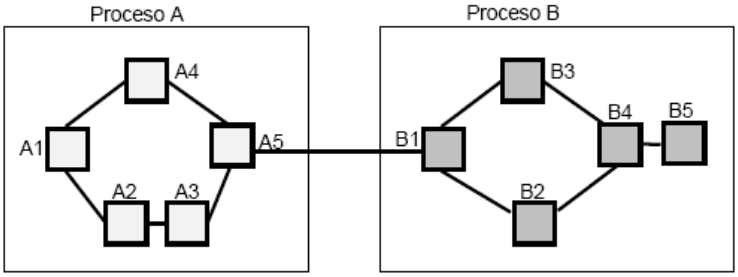


Figura 3.9. Modelo encadenado.

Este modelo soporta la transferencia de un único ítem de trabajo (una instancia del proceso o actividad) entre los dos ambientes del flujo de trabajo, el cual opera independientemente en el segundo ambiente sin sincronización adicional.

Escenario 2: Jerárquico (Sub-procesos anidados)

En este caso se le permite a un proceso ejecutado en un dominio particular del flujo de trabajo, ser completamente encapsulado como una única tarea, dentro de un proceso (superior) ejecutado en un dominio diferente. Existe una relación jerárquica entre el proceso superior y el proceso encapsulado, el cual es un subproceso del superior. La relación jerárquica puede ser extendida a través de distintos niveles, formando un conjunto de sub-procesos anidados. La recursión en este escenario puede ser permitida o no, por cada producto particular.

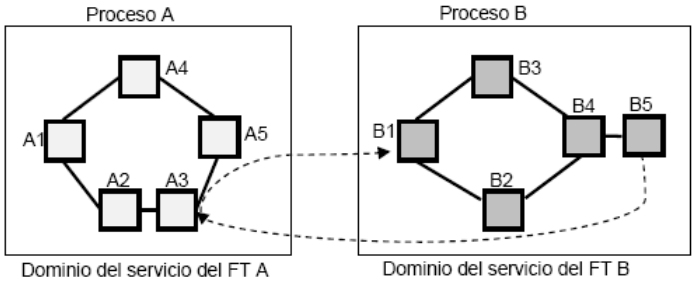


Figura 3.10. Modelo de sub-procesos anidados.

Escenario 3: Conexión No Discreta (Peer-to-Peer)

Este modelo permite un ambiente completamente mezclado. En este escenario, el proceso puede progresar transparentemente de tarea a tarea, sin ninguna acción específica de los usuarios o administradores, con interacciones entre los motores del flujo de trabajo cuando es necesario.

El escenario requiere que ambos motores del flujo de trabajo soporten un conjunto común de APIs para la comunicación y para que ambos puedan interpretar una definición de procesos común, ya sea que esta definición haya sido importada a ambos ambientes desde una herramienta de definición común, o que sea exportada entre los motores durante la fase de ejecución. Los datos de la aplicación y los datos relevantes del flujo de trabajo también deben ser pasados entre los motores. La WfMC intenta definir un número de niveles de conformidad, permitiendo que las especificaciones más antiguas se enfrenten con escenarios simples y las funciones adicionales se enfrenten con escenarios complejos a ser agregados en el futuro.

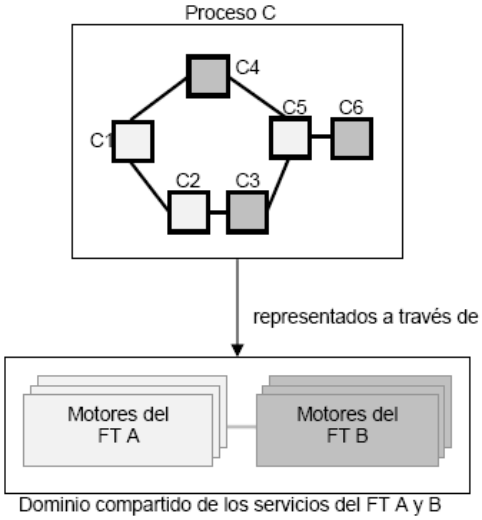


Figura 3.11. Modelo de Peer-to-Peer.

Escenario 4: Sincronización en Paralelo

Este modelo le permite a dos procesos operar independientemente, posiblemente a través de Servicios de Ejecución de Flujos de Trabajo distintos. Pero requiere que existan puntos de sincronización entre los procesos. Esto implica que una vez que los procesos alcanzan un punto predefinido dentro de su secuencia de ejecución se genere un evento común.

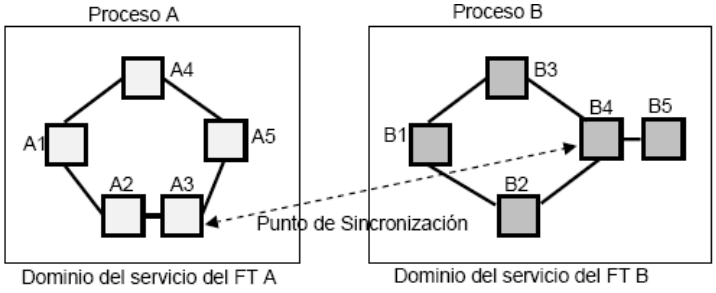


Figura 3.12. Modelo de Sincronización Paralela.

Hay dos grandes aspectos para la necesidad de la interoperabilidad:

- El alcance en el que la interpretación común de la definición de procesos es necesaria y que pueda ser realizada.
- El soporte en tiempo de ejecución para el intercambio de varios tipos de información de control y para la transferencia de los datos relevantes del flujo de trabajo y/o de las aplicaciones, entre los distintos servicios de ejecución.

La sincronización puede ser utilizada como proceso de planificación de la ejecución en paralelo de procesos, además de la transferencia de los datos relevantes del flujo de trabajo entre las diversas instancias de los procesos.

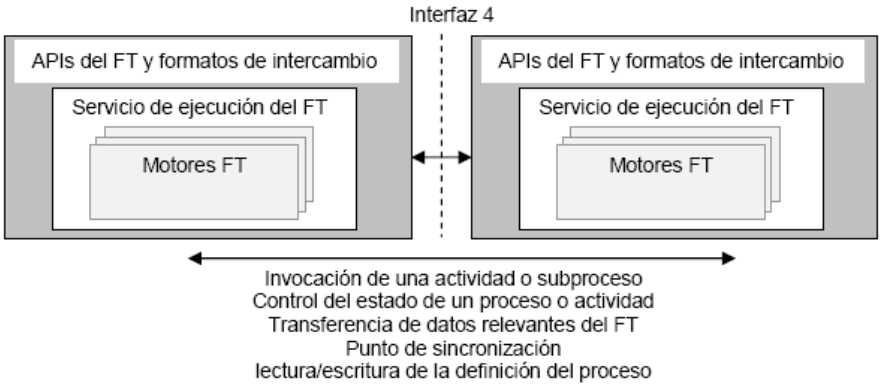


Figura 3.13. Componentes que participan en la Interfaz 4.

Un gran número de comandos WAPI se están desarrollando para soportar la interoperabilidad ya sea por medio de una invocación directa entre dos servicios de flujo de trabajo o vía una función conectora. Muchos de los comandos WAPI discutidos anteriormente también son potencialmente aplicables en las interacciones para la interoperabilidad de flujos de trabajo, como se muestra en la tabla 3.4.

APIs
– Establecimiento de sesión
- Operaciones sobre la definiciones de los flujos de trabajo y sus objetos
- Funciones de estado y control de procesos
- Funciones de gestión de las actividades

- Operaciones de manipulación de datos

Tabla 3.4. APIs para la interoperabilidad de flujos de trabajo

3.5.8 Interfaz 5: Administración y el Monitoreo

El propósito de esta interfaz es permitir una vista completa del estado del flujo de trabajo, además de poder realizar auditorías sobre los datos del sistema. Permite que distintos servicios de ejecución del flujo de trabajo compartan las mismas funciones de administración y monitorización del sistema, como pueden ser, por ejemplo, la gestión de usuarios, el control de los recursos y la supervisión del estado de todo el proceso.

El diagrama de la Figura 3.14 ilustra cómo una aplicación de administración independiente interactúa con los distintos dominios del flujo de trabajo. Es posible implementar otros escenarios posibles, como por ejemplo tener la aplicación de administración y monitoreo dentro del propio servicio de ejecución del flujo de trabajo.

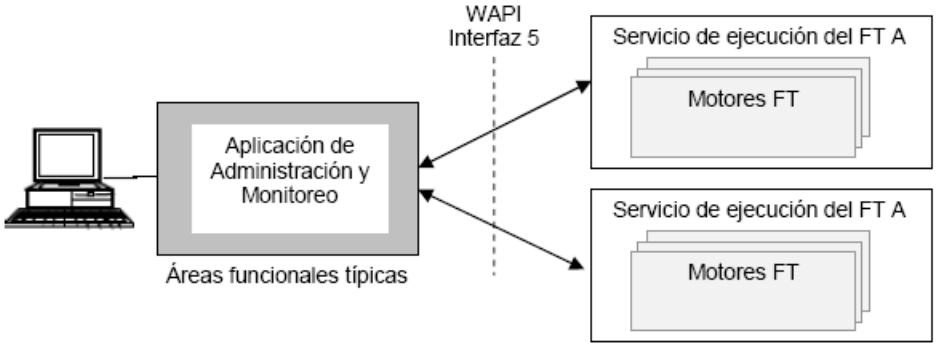


Figura 3.14. Componentes que participan en la Interfaz 5.

La tabla 3.5 muestra las APIs asociadas a esta interfaz.

Categoría	APIs
Operaciones de Gestión del Usuario	– establecimiento/eliminación/suspensión/modificación de privilegios de usuarios o grupos de trabajo
Operaciones de Gestión de Roles	– definición/eliminación/modificación de relaciones entre roles – establecimiento de atributos del rol
Operaciones de Gestión de Auditoría	– consulta/impresión/comienzo/eliminación de una auditoría o registro de evento

Operaciones de Control de Recursos	– definición/eliminación/modificación de procesos o actividades concurrentes – interrogación de datos de control – cambiar el estado operacional de una definición de proceso de flujo de trabajo y/o sus instancias de proceso existentes – habilitar o deshabilitar versiones particulares de una definición de proceso
Funciones de Supervisión de Procesos	– cambiar el estado de todas las instancias de actividad o proceso de un tipo específico – asignar atributos a todos las instancias de actividad o proceso de un tipo específico – terminar todas las instancias de un proceso
Funciones de Estado de los Procesos	– abrir/cerrar una consulta de una instancia de actividad o proceso, definiendo un criterio de filtro opcional – obtener detalles de las instancias de actividad o proceso – obtener detalles de una instancia de actividad o proceso específica

Tabla 3.5. APIs para la definición de procesos

CAPÍTULO 4.

Especificación Funcional de la Interfaz de Aplicaciones Invocadas con Servicios Web

En este capítulo se describe la especificación funcional de la Interfaz de las Aplicaciones Invocadas, propuesta en el estándar de la WfMC, utilizando servicios web. El objetivo es mejorar la selección de aplicaciones en tiempo de ejecución, permitiendo al motor del SGFT invocarlas independizándose de la ubicación exacta de las mismas.

La estructura del capítulo es la siguiente. La sección 4.1 introduce los servicios web y las tecnologías subyacentes. La sección 4.1.1 describe los objetivos de los servicios web. La sección 4.1.2 presenta algunos ejemplos de su uso. La sección 4.1.3 describe la arquitectura básica de los servicios web, con los componentes principales, los roles de los participantes y las operaciones que se llevan a cabo. Y la sección 4.1.4 describe la interacción que se da entre el solicitante y el proveedor de un servicio a la hora de utilizarlo. La sección 4.2 presenta una breve explicación del lenguaje utilizado para especificar servicios web,

denominado WSDL. Finalmente, la sección 4.3 presenta la especificación de la Interfaz de las Aplicaciones Invocadas, en particular, de la operación que permite invocar una aplicación externa.

4.1. Los Servicios Web

En los Sistemas de Gestión de Flujos de Trabajo, a menudo, el usuario requiere la utilización de aplicaciones externas para desarrollar sus tareas, por ejemplo, aplicaciones que le permitan utilizar servicios de e-mail, fax, realizar la administración de sus documentos, u otras aplicaciones propias del usuario. La Interfaz de las Aplicaciones Invocadas definida por la WfMC, permite que el motor del SGFT pueda invocar este tipo de aplicaciones. Esta interfaz requiere conocer la información acerca de la aplicación que se quiere invocar en tiempo de desarrollo. Si bien en muchos Sistemas de Gestión de Flujos de Trabajo se conocen a priori las aplicaciones que se desean utilizar, también se dan muchos casos donde varias aplicaciones que brindan un mismo servicio podrían ser requeridas por el motor. Inclusive si se desea invocar una aplicación específica, sería de gran utilidad abstraerse de la ubicación exacta de la misma. Con el fin de permitir que el motor del SGFT invoque una aplicación externa sin necesidad de conocer su ubicación, se plantea una solución usando servicios web.

Un servicio web es un componente abierto y auto descripto que soporta la composición rápida de aplicaciones distribuidas. Su principal beneficio es que permite que las aplicaciones sean más modulares y desacopladas, facilitando su reutilización en distintas plataformas o lenguajes de programación. Por ello, los servicios web proveen esencialmente un medio estándar de comunicación entre diferentes aplicaciones de software. Su uso en la Web se ha extendido rápidamente debido a la creciente necesidad de comunicación e interoperabilidad entre aplicaciones. Los servicios web se definen como aplicaciones auto-contenidas y auto-descritas, que pueden ser publicadas, localizadas e invocadas a través de la Web. Una vez desarrolladas, otras aplicaciones (y otros servicios web) pueden descubrirlas e invocar el servicio dado.

El desarrollo y la programación de sistemas en componentes ha llevado a lo largo del tiempo a tener la necesidad de reutilizarlos en diferentes proyectos. Ya sean componentes desarrollados por uno mismo o componentes desarrollados por terceros. Hasta la existencia de los servicios web, ésta reutilización se limitaba a un lenguaje de programación o a una plataforma en particular. Por lo tanto, el uso de los Servicios Web facilita la reutilización de

una aplicación en distintas plataformas o lenguajes ya sea para uso personal en distintos proyectos, para comercializarlos o adquirir prestaciones de terceros. De la misma forma que anteriormente se incluía en las aplicaciones referencias a otras librerías, como ser DDLs, ahora se pueden referenciar funciones que se estarán ejecutando en otra computadora o servidor sin importar en qué están programadas ni en qué plataforma están corriendo. Uno de los ejemplos más comunes del uso de servicios web se encuentra en los sitios web de comercio electrónico, los cuales hacen uso de un servicio web para validar los datos de las tarjetas de crédito de sus clientes. Normalmente este servicio web es provisto por algún banco o entidad financiera que actúa como intermediario entre el comercio y las tarjetas de crédito.

La definición oficial provista por la W3C (World Wide Web Consortium) [WSA], especifica un servicio web como *"una aplicación software identificada mediante una URI (Uniform Resource Identifier), cuya interfaz (y uso) es capaz de ser definida, descripta y descubierta mediante artefactos XML, y soportar interacciones directas con otras aplicaciones software usando mensajes basados en XML y protocolos basados en Internet".*

Algunos ejemplos de servicios web son:

- *Información del negocio:* pronóstico del tiempo, calendario, noticias, chequeo de crédito de una tarjeta, subastas, cotizaciones de acciones, planificación de vuelos, etc.
- *Transaccional:* reservas aéreas, acuerdos para rentar un auto, orden de compra, gestión de una cadena de suministro, etc.
- *Externalización de procesos de negocio:* vínculos comerciales a nivel de flujo de trabajo, etc.

Los servicios web se basan en estándares abiertos, lo que permite a las aplicaciones comunicarse más libremente e independientemente de las plataformas en que se están ejecutando. Una característica de los servicios web es que se describen en términos de las operaciones del mismo y los mensajes de entrada y salida de cada operación. Tales definiciones se expresan en el lenguaje de marcado extensible ***XML*** (Extensible Markup Language) [XML], usando el lenguaje de descripción de servicios web ***WSDL*** (Web Service Description Language) [WSDL2.0-0] [WSDL2.0-1]. La noción de describir un servicio web independientemente de la tecnología en la cual ha sido implementado es robustamente capturada en este lenguaje. WSDL especifica claramente la ubicación del servicio junto con las operaciones que necesitan ser invocadas, los parámetros que

deben pasarse, los tipos de datos de cada parámetro, y los distintos formatos de mensajes y tipos.

Otros estándares que se utilizan son:

- Para la representación de los mensajes: un protocolo de aplicación basado en mensajes que permite que una aplicación interactúe con los servicios web. Este protocolo es ***SOAP*** (Simple Object Access Protocol) [SOAP].
- Para el transporte de los mensajes: un protocolo de transferencia que se encarga de transportar los mensajes por Internet. Este protocolo de transporte es ***HTTP*** (Hiper Text Transport Protocol).
- Para el registro y la localización del servicio: un directorio de servicios web distribuido y basado en Web que permite que se listen, busquen y descubran este tipo de software. Este directorio es ***UDDI*** (Universal Description, Discovery and Integration) [UDDI].

El uso de estos protocolos estándares es necesario para lograr la interoperabilidad en ambientes heterogéneos, con independencia del sistema operativo, el lenguaje de programación, etc.

Figura 4.1. Estándares de los Servicios Web

La Figura 4.1 muestra por capas la infraestructura correspondiente a los servicios web [WSA]. La capa de transporte corresponde a los protocolos utilizados para el envío de la

información, la capa de mensajes al protocolo SOAP, la capa de descripción a WSDL y la capa de procesos a las diversas tecnologías para la publicación, descubrimiento, composición, orquestación, desarrollo e integración de servicios Web. Los servicios web no son aplicaciones con una interfaz gráfica con que las personas puedan interactuar, sino que son componentes de software accesibles desde Internet por otras aplicaciones. Estos componentes de software basados en estándares pueden variar en su funcionalidad desde operaciones simples, tales como recuperar el valor al que están cotizando las acciones de una compañía, a sistemas complejos tales como la reserva on-line de un viaje. Una vez desarrollados, otras aplicaciones y servicios web pueden descubrirlo e invocarlo. El mayor logro de los servicios web es que han permitido a las aplicaciones comunicarse entre ellas de manera independiente del lenguaje de programación y de la plataforma, otorgando mayor interoperabilidad.

4.1.1 Objetivos de los servicios web

Para conseguir alta calidad en los servicios, éstos deberían soportar los siguientes requerimientos principales [Labra06]:

- *Independencia del lenguaje y de la plataforma*: separación entre la especificación y la implementación.
- *Interoperabilidad*: los clientes y los servicios se comunican y se entienden entre sí, sin importar sobre qué plataforma se ejecutan. Este objetivo se puede alcanzar sólo si los clientes y los servicios tienen una forma estándar de comunicación entre sí, que sea consistente a través de plataformas, sistemas, y lenguajes. Los servicios web abarcan un conjunto de protocolos y tecnologías que están siendo ampliamente aceptados y utilizados, y que son independientes de la plataforma, el sistema y el lenguaje (XML, SOAP, WSDL, UDDI).
- *Acoplamiento débil*: Un servicio asincrónico ejecuta su procesamiento sin forzar al cliente a permanecer esperando hasta que termine el procesamiento. Un servicio sincrónico fuerza al cliente a que espere. La relativamente limitada interacción requerida para que un cliente se comunique con un servicio asincrónico, especialmente un servicio que maneja un documento tal como una orden de compra, permite que las aplicaciones que utilizan estos servicios escalen sin imponer una fuerte carga de comunicación sobre la red.

- *Modularidad y Reusabilidad*: Los desarrolladores dentro de una organización y entre organizaciones (particularmente, en el caso de negocios asociados) pueden tomar el código desarrollado para las aplicaciones existentes, exponerlos como servicios web, y luego reutilizarlos para atender nuevos requerimientos de negocio. Reutilizar funcionalidades existentes dentro o fuera de una organización, en lugar de desarrollar código que reproduce aquellas funciones, puede resultar en un enorme ahorro de costo y tiempo de desarrollo.
- *Escalabilidad*: las aplicaciones que utilizan servicios web tienden a escalar fácilmente a medida que deben manejar más usuarios. Esto se debe a que existen menos dependencias entre la aplicación solicitante y el servicio que la misma utiliza.
- *Flexibilidad***:** Los servicios débilmente acoplados generalmente son más flexibles que las aplicaciones fuertemente acopladas. En una arquitectura fuertemente acoplada, los diferentes componentes de una aplicación están estrechamente ligados entre sí, compartiendo semánticas, librerías y, a menudo, estados. Esto dificulta la evolución de una aplicación a fin acompañar los cambios en los requerimientos del negocio. La naturaleza débilmente acoplada, basada en documento y asincrónica en una SOA permite que las aplicaciones resulten flexibles, y fáciles de evolucionar conforme cambian los requerimientos.

El objetivo fundamental de los servicios web es permitir que diferentes aplicaciones, realizadas con diferentes tecnologías, y ejecutándose en una variedad de entornos, puedan comunicarse e integrarse. También, la reutilización de desarrollos sin importar la plataforma en que funcionan o el lenguaje en el que estén escritos. Así, los servicios web se constituyen como una capa adicional a las aplicaciones de forma tal que pueden interactuar entre ellas usando para comunicarse tecnologías estándares desarrolladas en el contexto de Internet.

4.1.2 Ejemplos de servicios web

A continuación se presentan algunos ejemplos de uso de servicios web.

Información del Negocio (Business Information):

Estos servicios web permiten acceder a información del negocio disponible en la web (como estado del tiempo, calendario, noticias, verificación de crédito, etc.). Exponen

funcionalidades simples y públicas accesibles desde la web. Los usuarios pueden consultar estas funcionalidades, aunque no realizan ningún tipo de transacción, es decir, no hay interacción entre el usuario y el servicio web.

- *Estado del tiempo*: un servicio web que regularmente colecta datos provenientes de una red de sensores, los procesa y publica el estado del tiempo y el pronóstico para el siguiente día.
- *Calendario*: un servicio web que muestra un calendario con vistas diarias, semanales, mensuales y anuales, a partir del año, e información contextual como la zona horaria del lugar.
- *Noticias*: un portal de noticias que a través de servicios web provee información financiera, de las cotizaciones de acciones, etc., con la posibilidad de personalizar y formatear la presentación de la información según el usuario.
- *Verificación de crédito*: un servicio web que permita comprobar el crédito de un comprador, y determinar si se efectúa la venta o no.
- *Agencia de viajes*: Una agencia de viajes que quiere ofrecer a sus clientes la posibilidad de consultar vía web toda la información de los viajes disponibles con respecto a fechas, horarios y disponibilidad de vuelos, hoteles, alquiler de autos, excursiones, etc. Los proveedores del servicio (compañía aérea, empresas de autobuses, cadenas hoteleras, etc.) proveen servicios web para consultar sus ofertas.

Transaccional (Transactional, for B2B or B2C):

Estos servicios web permiten la interacción de un usuario o empresa con uno o varios servicios web, para conseguir un resultado de valor (como reservas en líneas aéreas, alquiler de autos, orden de compra de un producto, etc.). Dichos servicios web son básicamente extensiones de sistemas ya construidos para que éstos sean accesibles por aplicaciones y sistemas heterogéneos desarrollados bajo cualquier plataforma y lenguaje y que participan en procesos comunes, que implican realizar transacciones.

- *Reservas en líneas aéreas*: Una agencia de viajes que quiere ofrecer a sus clientes la posibilidad de consultar vía web toda la información de los viajes disponibles. Los proveedores del servicio (compañía aérea, empresas de autobuses, cadenas hoteleras, etc.) proveen servicios web para consultar sus ofertas y realizar reservas.

- *Alquiler de autos*: Una agencia de alquiler de autos ofrece a sus clientes la posibilidad de alquilar autos a través de un servicio web que permite acordar el período de tiempo, fecha de retiro y devolución del vehículo, vehículo elegido, etc.
- *Ordenes de compra*: un servicio web que permite a un cliente consultar los precios de los productos en venta, elegir algún producto para comprar, acordar con el vendedor la fecha de entrega, disponer el pago y recibir el comprobante correspondiente.
- *Cadena de suministro*: un sistema de administración que permita categorizar, intercambiar, procesar y administrar productos a lo largo de toda una cadena de suministro, desde la producción en la casa hasta su publicación en la web.

Externalización de procesos de negocio:

Si bien uno de los objetivos de los servicios web es promover la integración de aplicaciones de empresas, para unir sistemas, aplicaciones y fuentes de datos disímiles dentro de las organizaciones en forma integrada, también se apunta a extender las aplicaciones de negocio en forma segura entre distintas empresas, a través de múltiples canales de comunicación y en entornos de e-government para la prestación de servicios al ciudadano y la integración de aplicaciones entre organizaciones gubernamentales.

Este modelo implica que los servicios web se vayan incorporando gradualmente a la web, y de esta forma, las organizaciones puedan diseñar sus aplicaciones con foco en su actividad de negocio y contratar servicios externos especializados que les permitan interoperar con otras compañías. Así, las empresas establecen vínculos comerciales a un nivel de flujo de trabajo con otras empresas, lo cual permite una total integración a nivel de los procesos de negocio.

4.1.3 Arquitectura básica

La Arquitectura Orientada a Servicios (Service-Oriented Architecture - SOA) representa una manera de utilizar y administrar un conjunto de servicios web a través del uso de SOAP, WSDL y UDDI. Los servicios son vistos como unidades lógicas, definidos en términos de la funcionalidad que proveen y están descriptos por documentos WSDL que sólo proveen los detalles necesarios para su invocación como los tipos de datos, las operaciones, protocolos y ubicación correspondiente a la implementación de los mismos [WSA]. Su invocación se realiza mediante el intercambio de mensajes de acuerdo a los estándares antes mencionados.

SOA está basada en las interacciones entre el proveedor, las agencias de descubrimiento y el solicitante (Figura 4.2). Básicamente la relación que existe entre ellos corresponde con publicar la descripción de un servicio, encontrar cuales se encuentran disponibles y enlazar dichos servicios al cliente o solicitante. En esta arquitectura básica, el solicitante y el proveedor interactúan basados en la información provista por la descripción del servicio web, la cual ha sido publicada por el proveedor y descubierta por el solicitante a través de alguna agencia de descubrimiento. Los proveedores y solicitantes interactúan por medio del intercambio de mensajes.

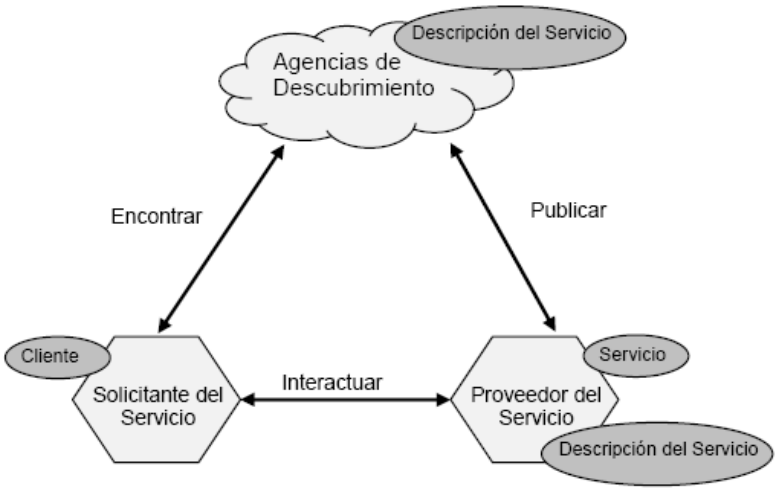


Figura 4.2. Operaciones de los Servicios Web

Los componentes que participan son:

- *Servicio:* es un módulo de software desarrollado sobre plataformas accesibles desde la web, y provisto por un proveedor de servicios. El mismo es invocado o interactúa con el solicitante del servicio. También puede funcionar como un solicitante, utilizando otro servicio web en su implementación.
- *Descripción del Servicio:* contiene los detalles de la interface e implementación del servicio web. Incluye sus tipos de datos usados, operaciones, información del tipo de enlace (*binding)*, y ubicación en la red. Su descripción completa puede realizarse como un conjunto de documentos de descripción XML. La descripción del servicio puede publicarse directamente por un proveedor o por una Agencia de Descubrimiento.

Los roles de los participantes son:

- *Proveedor del Servicio:* desde una perspectiva del negocio, es el dueño del servicio. Desde una perspectiva de la arquitectura, es la plataforma que permite el acceso al

servicio. También puede ser referenciado como el ambiente de ejecución del servicio, o el contenedor del servicio.

- *Solicitante del Servicio:* desde una perspectiva del negocio, es quien requiere cierta funcionalidad que ofrece el servicio. Desde una perspectiva de la arquitectura, es la aplicación que está buscando e iniciando una interacción con el servicio. Este rol puede ser utilizado por un browser conducido por una persona o un programa sin una interface de usuario, por ejemplo, otro servicio web.
- *Agencia de Descubrimiento:* es un conjunto de descripciones de servicios web donde los proveedores de servicios publican sus descripciones. Esta agencia puede ser distribuida o centralizada. Los solicitantes pueden encontrar servicios y obtener información del enlace (binding), para usar durante el desarrollo o la ejecución del servicio.

Las principales operaciones son:

- *Publicar:* para poder ser accesible desde la web, un servicio web necesita publicar su descripción, de manera tal que el solicitante puede encontrarlo. Lo que se publica es la descripción del servicio.
- *Encontrar:* a través de esta operación, el solicitante del servicio recupera una descripción directamente o consulta al registro de servicios por el tipo de servicio requerido. Esta operación se invoca tanto cuando se busca la descripción del servicio, como cuando se desea acceder al servicio en tiempo de ejecución.
- *Interactuar:* en tiempo de ejecución, el que requiere el servicio inicia una interacción al invocar el servicio web, haciendo uso de la descripción para la ubicación, contacto e invocación del servicio.

4.1.4 Dinámica de los servicios web

La interacción que se da entre un cliente y el servicio web requerido se resume en la Figura 4.3.

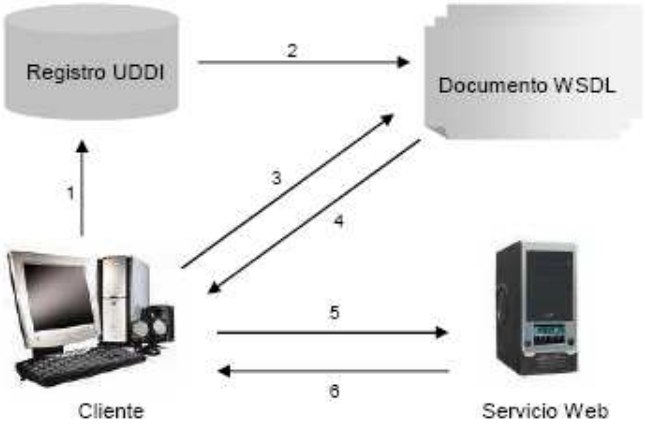


Figura 4.3. Dinámica de los Servicios Web

1. El Cliente se comunica con el Registro UDDI para encontrar un servicio.
2. El Registro UDDI le indica al cliente un documento WSDL.
3. El Cliente accede al documento WSDL.
4. WSDL le provee lo necesario para interactuar con el servicio web.
5. El cliente envía un requerimiento usando SOAP.
6. El servicio web retorna una respuesta SOAP.

La interacción se da a través de un patrón de mensajes (Message Exchange Patterns, MEPs) que define la secuencia de intercambio. Se debe especificar una descripción del servicio web, que indique su funcionalidad. Esta descripción incluye cómo invocar el servicio web y el resultado que retornará. Involucra además los tipos de datos, estructura, MEPs, y la dirección del proveedor del servicio.

Uno de los pilares de esta tecnología es la reusabilidad del software. Para ello, el desarrollador *publica* el servicio web para que esté disponible para el cliente que lo necesita. El servicio web se *publica* en la Agencia de Descubrimiento (Service Discovery Agency), donde el proveedor registra el módulo de software para que sea accesible desde la Red. El proveedor debe dar una descripción del servicio web para que el cliente analice si realmente otorga la funcionalidad que necesita. Cuando el solicitante *descubre* el servicio en la Agencia de Descubrimiento, lo puede invocar de acuerdo a la descripción ofrecida.

Lo interesante de las tecnologías que usan XML, SOAP, es que pueden integrarse en aplicaciones ya existentes sin demasiada complejidad. Productos que usan SOAP pueden usar servidores HTTP existentes y procesadores XML que tenga el cliente.

4.2. El lenguaje WSDL

WSDL es el lenguaje de especificación utilizado para definir los servicios web. Permite proveer información acerca de la ubicación del servicio y su interfaz. Con esta información el usuario sabe como interactuar con el servicio. WSDL describe la interfaz de un servicio web como un conjunto de puntos finales de comunicación (métodos) capaces de intercambiar mensajes, es decir, recibir llamadas con sus parámetros correspondientes y generar respuestas con el resultado que le corresponda. WSDL es un archivo XML que describe el conjunto de métodos expuestos por un servicio web, es decir describe la funcionalidad del servicio. Esta descripción incluye el número de argumentos, y tipo de cada uno de los parámetros de los métodos, y la descripción de los elementos que retornan. Este archivo XML se utiliza para describir los mensajes SOAP y cómo estos mensajes son intercambiados. Cuando se crea un servicio web y se pretende que otras aplicaciones lo utilicen, éstas deben acceder a un documento WSDL para conocer los métodos que expone el servicio web y cómo acceder a ellos, es decir, cuáles son los nombres de los métodos y qué tipo de parámetros espera cada uno de ellos.

WSDL separa la descripción de un servicio en dos partes [WSA]. Dentro de cada sección se utilizan un número de constructores que promueven la reusabilidad de la descripción y permiten separarla de los detalles de diseño. Una de ellas es la *interfaz abstracta*, la cual describe las operaciones soportadas por el servicio, los parámetros de la operación y los tipos de datos abstractos. Esta descripción es completamente independiente de la dirección de red concreta, el protocolo de comunicación o las estructuras de datos concretas del servicio. La otra parte es la *implementación concreta*, la cual liga la interfaz abstracta a una dirección de red, protocolo y estructuras de datos concretas. WSDL provee toda esta información a través de elementos XML. Los dos elementos a definir en la interfaz abstracta son:

- *types*: provee la definición de los tipos de datos utilizados para describir los mensajes a intercambiar.
- *interface*: describe la secuencia de mensajes que el servicio envía o recibe.

Los dos elementos a definir en la implementación concreta son:

- *binding*: define los detalles de implementación concretos del servicio.

- *service*: se usa para asociar un binding con el URL donde está realmente el servicio.

Por otro lado, la descripción de un servicio web tiene dos componentes principales: sus características funcionales y no funcionales [Papazoglou08]. La descripción funcional detalla las características operacionales que definen el comportamiento general de un servicio web, es decir, define los detalles de cómo es invocado el servicio, su ubicación, etc. Esta descripción focaliza en los detalles de la sintaxis de los mensajes y cómo configurar los protocolos de red para enviar estos mensajes. La descripción no funcional se concentra en sus atributos de calidad (Quality of Service - QoS), tales como costo del servicio, métricas de performance, como por ejemplo tiempo de respuesta, atributos de seguridad, autorización, autenticación, integridad transaccional, fiabilidad, escalabilidad y disponibilidad. Las descripciones no funcionales fuerzan al solicitante del servicio a especificar, en tiempo de ejecución, los atributos de calidad que pueden influir en la elección de un servicio web ofrecido por un proveedor. En la siguiente sección se presenta la descripción funcional del servicio web que permite invocar una aplicación externa, a partir de la definición de los cuatro elementos que componen el documento WSDL.

4.3. Especificación Funcional de la Interfaz de Aplicaciones Invocadas

La Interfaz de las Aplicaciones Invocadas define un conjunto de WAPIs (Workflow Aplication Programming Interfaces), altamente recomendadas para ser usadas por los componentes de un Sistema de Gestión de Flujo de Trabajo (tanto las aplicaciones de cliente como el motor del SGFT), para controlar dispositivos de aplicaciones especializadas, llamados *agentes de aplicación* (t*ool-agents)*. Estos agentes de aplicación contienen una variedad de métodos para la invocación de aplicaciones en todos los entornos de plataformas y redes en una interfaz estándar. Los agentes de aplicación comienzan y terminan las aplicaciones, pasan información relevante del SGFT y la aplicación a y desde la aplicación y controlan el estado a nivel de ejecución de la aplicación. Así, las WAPI's de la Interfaz de Aplicaciones Invocadas trabajan solamente con los agentes de aplicación.

Como la Interfaz de Aplicaciones Invocadas debe manejar requerimientos bi-direccionales (requerimientos a y desde la aplicación), dicha interacción con el agente de

aplicación depende de la interfaz y arquitectura de la aplicación, lo cual restringe la selección dinámica de las aplicaciones. Por otro lado, esta interfaz debería permitir el requerimiento y la actualización de datos de la aplicación y otras funcionalidades importantes en tiempo de ejecución. Para permitir todo esto y además, brindar mayor flexibilidad al motor del SGFT a la hora de invocar una aplicación externa, se propone especificar las WAPIs de esta interfaz con servicios web.

Una de las funciones principales de la Interfaz de Aplicaciones Invocadas es permitir invocar aplicaciones externas al SGFT. Esta función permite comenzar o activar una aplicación específica, asociada a un *item de trabajo*. La invocación de una aplicación siempre incluye el pasaje de información adicional, tal como parámetros de la aplicación y modo de invocación. En la especificación de esta función es necesario definir tipos de datos específicos para representar esta información adicional. Para poder invocar una aplicación, el motor del SGFT debe recibir información de entrada sobre la aplicación que se desea invocar y además, del item de trabajo específico que requiere dicha aplicación. Esta información se detalla a continuación:

- *Tipo de Aplicación*: Se refiere, por un lado, a aplicaciones relacionadas al SGFT específico de que se trate. Por ejemplo, en el caso de un sistema que implemente el proceso de desarrollo de software OpenUP/Basic, aplicaciones que generen código Java a partir de una diagrama de diseño, que estimen costos, duración total de un proyecto de software, esfuerzos, planificación de tareas, etc. Por otro lado, también se puede referir a aplicaciones no específicas del SGFT, como por ejemplo aplicaciones de oficina como procesadores de texto, clientes de correo o planillas de cálculo.
- *Información del SGFT*: Contiene información relevante para el motor del SGFT, que le permite identificar la relación entre la aplicación y la instancia de proceso que la requiere, el item de trabajo específico y los posibles modos de invocación de la aplicación (create, update, read-only, print, etc.).
- *Parámetros*: Almacenan información relevante para la aplicación, por ejemplo, el nombre de un archivo, el identificador de un registro, etc.
- *Requerimientos Adicionales*: Almacena información relevante para el motor del SGFT sobre características específicas de la aplicación buscada, por ejemplo, el Sistema Operativo, la ubicación geográfica, o el nombre concreto de la aplicación (en el caso de que el usuario desee invocar una aplicación concreta).

Como resultado de la innovación, el motor del SGFT retorna el nombre y la ubicación de la aplicación elegida. Puede ocurrir que la aplicación no pueda invocarse (que no se pueda comenzar, que no este definida, que este ocupada, etc.), lo cual deberá informase a quien realiza el requerimiento. Por lo tanto, el servicio web que permite invocar una aplicación debe definir la información de entrada, los datos de salida y los posibles errores en la invocación.

4.3.1. Definición de los tipos de mensajes

WSDL permite que los tipos de mensajes sean definidos directamente dentro del documento, en el elemento *types* (Figura 4.4), el cual es un hijo del elemento *description*.

En este caso, se definen tres tipos nuevos.

- ***InvokeApp***: el cual contiene los tres elementos informados por el usuario del SGFT al momento de invocar una aplicación externa, y también la información específica del SGFT:
  - appType: el tipo de aplicación.
  - parameters: los parámetros.
  - aditionalReq: la información adicional.
  - wfInformation: la información específica del SGFT.
- ***AppFound***: que contiene el resultado de la aplicación elegida para invocarse.
- ***WFException***: que contiene información relacionada a los mensajes de falla en la invocación de la aplicación.

En WSDL 2.0 todos los tipos de los mensajes se definen como elementos simples al nivel más alto, aunque cada elemento puede contener cualquier cantidad de subestructuras dentro. Este es el caso de los tres elementos especificados anteriormente: InvokeApp, AppFound y WFException, los cuales se definen a través de tipos complejos, formados a su vez por una secuencia de elementos simples.

```
<types>
      <xs:element name="InvokeApp" type="tInvokeApp"/>
      <xs:complexType  name="tInvokeApp">
         <xs:sequence>
            <xs:element name="appType" type="xs: string"/>
            <xs:element name="wfInformation" type="xs: tWFInformation"/>
                  <xs:complexType  name="tWFInformation">
                        <xs:sequence>
                              <xs:element name="procInstId" type="xs: string"/>
                              <xs:element name="workItemId" type="xs: string"/>
                              <xs:element name="appMode" type="xs:  string"/>
                        </xs:sequence>
                  </xs:complexType>
            <xs:element name="parmeters" type="xs: tParameter"/>
                  <xs:complexType  name="tParameter">
                        <xs:sequence>
                              <xs:element name="name" type="xs: string"/>
                              <xs:element name="description" type="xs: string"/>
                              <xs:element name="value" type="xs: string"/>
                        </xs:sequence>
                  </xs:complexType>
            <xs:element name="aditionalReq" type="xs: tAditionalRequeriments"/>
                  <xs:complexType  name=" tAditionalRequeriments">
                        <xs:sequence>
                              <xs:element name="name" type="xs: string"/>
                              <xs:element name="description" type="xs: string"/>
                              <xs:element name="value" type="xs: string"/>
                        </xs:sequence>
                  </xs:complexType>
         </xs:sequence>
      </xs:complexType>
      <xs:element name="AppFound" type="tAppFound"/>
      <xs:complexType  name="tAppFound">
         <xs:sequence>
            <xs:element name="appName" type="xs: string"/>
            <xs:element name="appLocation" type="xs: string"/>
         </xs:sequence>
      </xs:complexType>
      <xs:element name="WFException" type="xs: tException"/>
      <xs:complexType  name="tException">
         <xs:sequence>
            <xs:element name="name" type="xs: string"/>
            <xs:element name="message" type="xs: string"/>
         </xs:sequence>
      </xs:complexType>
  </xs:schema>
</types>
```

Figura 4.4. Descripción del elemento *types* del servicio web *InvokeApplication*

4.3.2. Definición de la interfaz abstracta

La interfaz abstracta de la operación *InvokeApplication* se decribe dentro del elemento *interface* (Figura 4.5), como un conjunto de operaciones abstractas. Cada operación representa una interacción simple entre el cliente y el servicio y especifica los tipos de mensajes que el servicio web puede enviar o recibir como parte de esa operación. Cada

operación define también el patrón de intercambio de mensajes para indicar la secuencia en la cual los mensajes asociados son transmitidos entre las partes. Para este servicio web se define una interfaz conteniendo una operación simple *opInvokeApp*, usando los tipos de mensajes antes definidos.

El patrón de intercambio de mensajes de la operación es de tipo *in-out*, esto es, contiene un mensaje de entrada y un mensaje de salida. El mensaje de entrada de la operación esta asociado al tipo de datos InvokeApp, el cual representa los datos que son enviados desde el consumidor al proveedor del servicio. El mensaje de salida de la operación esta asociado al tipo de datos AppFound, el cual encapsula los datos de retorno.

Además, una operación puede contener elementos asociados a los potenciales mensajes de falla que pueden intercambiarse entre el proveedor y el consumidor del servicio web. En este caso se define el elemento outfault, el cual referencia al tipo de mensaje de falla WFException.

```
<interface name = "InvokeAppInterface">
   <fault name = "InvokeAppFault_WorkflowException" element = "ians: WFException"/>

   <operation name="opInvokeApp"
         pattern="http://www.w3.org/2004/03/wsdl/in-out"
         style=http://www.w3.org/ns/wsdl/style/iri
         xsdlx:safe = "true">
         <input messageLabel="In" element="ians: InvokeApp" />
         <output messageLabel="Out" element="ians: AppFound" />
         <outfault ref="tns: InvokeAppFault_WorkflowException" messageLabel="Out"/>
   </operation>
</interface>
```

Figura 4.5. Descripción del elemento *interface* del servicio web *InvokeApplication*

Esta descripción de la interfaz del servicio provee un claro entendimiento de la sección de la interface abstracta del mismo, pero carece de información sobre el acceso concreto en la red y el protocolo. A continuación, se define esta información correspondiente a la implementación concreta del servicio.

4.3.3. Definición de la implementación concreta

Para asociar la descripción de la *interface* del servicio a una implementación existente se utiliza el elemento *binding*. El mismo provee información del protocolo de transmisión y los formatos de datos concretos del servicio web. En la Figura 4.6, los detalles de binding

para cada operación y falla son especificados en los elementos *operation* y *fault*, dentro del elemento *binding*.

```
<binding name="InvokeAppSOAPBinding"
   interface="tns: InvokeAppInterface"
   type="http://www.w3.org/ns/wsdl/soap"
   wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">

   <fault ref="tns: InvokeAppFault_WorkflowException"
   wsoap:code="soap:Sender"/>

   <operation ref="tns: opInvokeApp"
   wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
</binding>
```

Figura 4.6. Descripción del elemento *binding* del servicio web *InvokeApplication*

4.3.4. Definición del acceso al servicio web

El elemento *service* permite especificar desde dónde será accedido el servicio web. Especifica una interfaz simple que el servicio soportará, y una lista de puntos finales (endpoints) donde el servicio puede ser accedido. Cada punto final referencia un binding previamente definido, para indicar qué protocolos y formatos de transmisión se usarán. En la Figura 4.7 el elemento *interface* referencia a la interfaz definida en la sección 4.3.2., y el único elemento *endpoint* referencia al binding definido en la sección previa.

```
<service name="InvokeAppService"
   interface="tns: InvokeAppInterface">

   <endpoint name="InvokeAppEndpoint"
         binding="tns: InvokeAppSOAPBinding"
         address ="..."/>
</service>

</description>
```

Figura 4.7. Descripción del elemento *service* del servicio web *InvokeApplication*

Como se mencionó anteriormente, la descripción de un servicio web está dada tanto por sus características funcionales, como por sus características no funcionales. En el siguiente capítulo, se presenta la descripción no funcional del servicio web *InvokeApp*.

CAPÍTULO 5.

Especificación No Funcional de la Interfaz de Aplicaciones Invocadas con Servicios Web

La descripción sintáctica de un servicio web en el protocolo UDDI focaliza en los aspectos funcionales del mismo, sin contener ninguna descripción de los atributos de calidad (Quality of Service - QoS). Con el incremento del número de servicios web que proveen similares funcionalidades, es importante poner más énfasis en encontrar el mejor servicio web que satisface las necesidades del solicitante, incluyendo tanto sus requerimientos funcionales como los no funcionales. Las descripciones no funcionales fuerzan al solicitante del servicio a especificar, en tiempo de ejecución, los atributos de calidad que pueden influir en la elección de un servicio web ofrecido por un proveedor. En este capítulo se propone incorporar al proceso de selección de servicios web el análisis de los atributos de calidad requeridos por el solicitante, siempre en el marco de optimizar la invocación de aplicaciones

desde un SGFT. Se presenta la especificación no funcional de la Interfaz de las Aplicaciones Invocadas del estándar de la WfMC.

La estructura del capítulo es la siguiente. La sección 5.1 introduce los conceptos generales de las características de calidad de los servicios web y detalla las características que se tienen en cuenta en este trabajo. La sección 5.2 presenta la especificación no funcional de la Interfaz de las Aplicaciones Invocadas, en particular, de la operación que permite invocar una aplicación externa al SGFT. Finalmente, la sección 5.3 presenta la especificación completa del servicio web, incorporando a la especificación funcional presentada en el capítulo anterior la especificación no funcional.

5.1. Características de calidad de los servicios web

En el capítulo anterior se presentó la especificación funcional del servicio web que permite invocar una aplicación externa. Como se expresó en dicho capitulo, esta descripción sintáctica del servicio web focaliza en los aspectos funcionales del mismo. Una exigencia importante de las aplicaciones basadas en SOA es operar de forma tal de resultar fiables y ofrecer un servicio consistente a una variedad de niveles. Por ello, los requerimientos de un servicio web no deben focalizar solamente en las propiedades funcionales del mismo, sino que también se deben concentrar en describir el ambiente que lo aloja, es decir, describir las capacidades no funcionales o características de calidad (QoS) del servicio. Cada servicio puede ofrecer diversas opciones de características no funcionales basadas en los requerimientos técnicos resultantes de la demanda de disponibilidad, performance y escalabilidad, políticas de seguridad y privacidad, etc., todas las cuales deben ser descriptas.

Por otro lado, como se expresó anteriormente, el incremento del número de servicios web que proveen similares funcionalidades también revela que es importante poner más énfasis en encontrar el mejor servicio web que satisface las necesidades del solicitante. Es así que la especificación de las características QoS ofrecidas por un servicio web se vuelve de alta prioridad tanto para el proveedor como para sus clientes. La descripción no funcional de un servicio web se concentra en describir estas características de calidad del servicio.

Las características de calidad de un servicio web se refieren a la habilidad del servicio de responder a las invocaciones y llevarlas a cabo en consonancia con las expectativas del proveedor y de los clientes [Papazoglou08]. Diversos factores de calidad que reflejan las expectativas del cliente, como disponibilidad, conectividad y alta respuesta, se vuelven clave para mantener un negocio competitivo y viable, ya que pueden tener un serio impacto en la provisión del servicio web. Así, las características QoS resultan un criterio importante para determinar la usabilidad y utilidad de los servicios, lo cual influye en la popularidad de los mismos y son un importante punto de diferenciación entre los proveedores de dichos servicios web. Tener en cuenta las características de calidad de un servicio web es un desafío crítico y significativo dada la naturaleza dinámica e impredecible de Internet. Aplicaciones con muy diferentes características y requerimientos compiten por toda clase de recursos en la web. Los cambios en los patrones de tráfico, las transacciones de negocio críticas en cuanto a la seguridad, y los efectos de las fallas en la infraestructura, crean la necesidad de estandarizar las QoS de un servicio web. A menudo, características QoS no resueltas causan que aplicaciones transaccionales críticas sufran niveles inaceptables de degradación de la performance.

Tradicionalmente, las QoS se miden por el grado en el cual las aplicaciones, sistemas, redes y otros elementos de la tecnología de Internet soportan la disponibilidad de los servicios a un nivel requerido de performance, en todos los accesos y condiciones de descarga. Mientras que las métricas tradicionales de QoS son aplicables, las características particulares de los entornos de los servicios web demandan lograr mayor disponibilidad de las aplicaciones e incrementar la complejidad en términos de acceso y gestión de los servicios.

En el contexto de los servicios web, las QoS se pueden ver como la capacidad de ofrecer garantía sobre un conjunto de características cuantitativas, referidas específicamente a los aspectos no funcionales de los servicios. Estas características son un requerimiento necesario para entender el comportamiento subyacente del servicio de manera tal que otras aplicaciones y servicios puedan invocarlos y ejecutarlos como parte de su proceso de negocio.

En este trabajo se define una forma de incorporar las características de calidad de los servicios web, que posteriormente será tenida en cuenta para la selección del servicio web más adecuado. La Tabla 5.1 detalla los elementos clave que se tienen en cuenta para soportar las características de calidad QoS en el ambiente de los servicios web. En particular, se tienen en cuenta las características relacionadas al tiempo de ejecución del

servicio. En el Anexo B se detallan todas las características correspondientes a estas categorías de QoS.

Característica de Calidad	Definición	
Performance	Velocidad para completar un requerimiento de un servicio, se mide en términos de rendimiento (throughput), tiempo de respuesta, latencia, tiempo de ejecución, tiempo de transacción, etc.	
	Rendimiento	Número de requerimientos de servicios web atendidos en un período de tiempo dado.
	Tiempo de Respuesta	Tiempo esperado entre que se envía un requerimiento y se recibe una respuesta.
	Latencia	Tiempo entre que un requerimiento de un servicio arriba y el requerimiento comienza a ser atendido.
	Tiempo de Ejecución	Es el tiempo que necesita un servicio web para procesar su secuencia de actividades.
	Tiempo de Transacción	Representa el tiempo que transcurre mientras el servicio web está completando una transacción.
Disponibilidad	Probabilidad de que el servicio web este disponible	
Fiabilidad	Habilidad de un servicio para funcionar correcta y consistentemente y proveer la misma calidad a pesar de las fallas en la red o el sistema	
Precisión	Radio de error producido por un servicio	
Robustez	Grado en el cual un servicio puede funcionar correctamente en presencia de entradas inválidas, incompletas o conflictivas	
Reputación	Medida de la reputación otorgada al servicio por el usuario final	

Tabla 5.1. Características de calidad de los servicios web

La incorporación de las características de calidad de los servicios web ofrece importantes beneficios, tanto para los usuarios como los proveedores. A los usuarios les permite expresar sus necesidades y utilizar el servicio web que mejor se ajusta a las

mismas, mientras que a proveedores pueden publicar mejor las capacidades de sus servicios.

5.2. Especificación no funcional de la Interfaz de las Aplicaciones Invocadas

Las características de calidad de los servicios web son muy importantes tanto para el proveedor del servicio como para el consumidor, dado que el número de servicios web que proveen similares funcionalidades se está incrementando en la web. Las tecnologías actuales para la descripción, publicación y descubrimiento de los servicios, tales como WSDL y UDDI, consideran solamente los requerimientos funcionales y soportan el descubrimiento en tiempo de diseño, o descubrimiento estático del servicio. Los requerimientos no funcionales, es decir, las características de calidad QoS, no son soportadas por los registros actuales de UDDI [Ran03].

Por ello se propone especificar los requerimientos no funcionales de la operación *InvokeApp*, complementando la especificación funcional presentada en el capítulo anterior. En la Figura 5.1 se presenta dicha especificación, donde los requerimientos no funcionales se definen como un elemento nuevo en la sección *types* del documento WSDL.

```
<xs:element name="QoSReq" type="xs: tQoSReq"/>
    <xs:complexType  name="tQoSReq">
        <xs:sequence>
            <xs:element name="QoSname" type="xs: string"/>
            <xs:element name="QoSunitType" type="xs: tType"/>[1]
            <xs:element name="QoSminValue" type="xs: float"/>
            <xs:element name="QoSmaxValue" type="xs: float"/>
        </xs:sequence>
    </xs:complexType>
```

Figura 5.1. Descripción del elemento *QoSReq* del servicio web *InvokeApplication*

El tipo de datos ***tType*** representa un tipo de datos enumerado que especifica las posibles unidades de medida de las características QoS (numérico, milisegundos, porcentaje, etc.).

La Figura 5.2 ilustra un ejemplo de especificación WSDL del tiempo de respuesta requerido para el servicio web que permite invocar una aplicación. Los valores mínimo y máximo indican que el tiempo demorado para invocar la aplicación debe estar entre 0,2 y 0,5 milisegundos.

```
< QoSname> tiempo de respuesta < /QoSname>
< QoSunitType/> milisegundos < /QoSunitType>
< QoSminValue/> 0,2 < /QoSminValue>
< QoSmaxValue/> 0,5 < /QoSmaxValue>
```

Figura 5.2. Especificación de la característica de calidad *Tiempo de Respuesta*

En la Figura 5.3 se puede observar otro ejemplo de especificación WSDL de las características de calidad, correspondiente a la disponibilidad del servicio web. Los valores mínimo y máximo indican que la probabilidad de que el servicio web este disponible debe estar entre el 95% y el 99%.

```
< QoSname> disponiblidad < /QoSname>
< QoSunitType/> porcentaje < /QoSunitType>
< QoSminValue/> 95 < /QoSminValue>
< QoSmaxValue/> 99 < /QoSmaxValue>
```

Figura 5.3. Especificación de la característica de calidad *Disponibilidad*

Las Figuras 5.4 y 5.5 muestran la especificación de las características throughput y reputación, respectivamente. El throughput indica que el número de requerimientos de servicios web atendidos en un período de tiempo dado debe ser mayor a 100 y menor a 150.

```
< QoSname> throughput < /QoSname>
< QoSunitType/> numérico < /QoSunitType>
< QoSminValue/> 100 < /QoSminValue>
< QoSmaxValue/> 150 < /QoSmaxValue>
```

Figura 5.4. Especificación de la característica de calidad *throughput*

Por su parte, la reputación otorgada al servicio web por el usuario final no deber ser menor a 8.

```
< QoSname> reputación < /QoSname>
< QoSunitType/> numérico < /QoSunitType>
< QoSminValue/> 8 < /QoSminValue>
< QoSmaxValue/> 10 < /QoSmaxValue>
```

Figura 5.5. Especificación de la característica de calidad *Reputación*

Al incorporar la descripción de las características de calidad de los servicios web, estas se pueden tener en cuenta al momento de realizar la correspondencia entre los

requerimientos del usuario y los servicios web disponibles, como se verá en el siguiente capítulo.

5.3. La especificación completa

Los cuatro elementos especificados en el capítulo 4 y el nuevo elemento especificado en la sección 5.2, permiten definir la *interface abstracta* y la *implementación concreta* de un servicio web. Estos elementos están contenidos en el elemento raíz del documento WSDL denominado *description*. En este elemento también se detalla la ubicación específica del espacio de nombres (namespace) y la documentación adicional, donde se puede agregar información del propósito y uso del servicio, el significado de los mensajes, las restricciones sobre su uso, y la secuencia en la cual las operaciones deberían ser accedidas. Las Figuras 5.6 y 5.7 presentan el documento WSDL completo que permite invocar una aplicación externa usando servicios web.

De manera similar es posible especificar el resto de las operaciones de la Interfaz de las Aplicaciones Invocadas, para completar la especificación con servicios web de dicha interfaz.

```
<?xml version="1.0" encoding="utf-8" ?>
<description
   xmlns="http://www.w3.org/ns/wsdl"
   targetNamespace= "http://.."
   xmlns:tns="http://..."
   xmlns:ians="http://..."
   xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
   xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions">
<types>
   <xs:schema
      xmlns:xs=http://www.w3.org/2001/XMLSchema
      targetNamespace= "http://..."
      xmlns="http://...">

      <xs:element name="InvokeApp" type="tInvokeApp"/>
      <xs:complexType  name="tInvokeApp">
         <xs:sequence>
            <xs:element name="appType" type="xs: string"/>
            <xs:element name="wfInformation" type="xs: tWFInformation"/>
                  <xs:complexType  name="tWFInformation">…</xs:complexType>
            <xs:element name="parmeters" type="xs: tParameter"/>
                  <xs:complexType  name="tParameter"> …</xs:complexType>
            <xs:element name="aditionalReq" type="xs: tAditionalRequeriments"/>
                  <xs:complexType  name=" tAditionalRequeriments">…</xs:complexType>
            <xs:element name="QoSReq" type="xs: tQoSReq"/>
                  <xs:complexType  name="tQoSReq"> … </xs:complexType>
         </xs:sequence>
      </xs:complexType>

      <xs:element name="AppFound" type="tAppFound"/>
      <xs:complexType  name="tAppFound">
         <xs:sequence>
            <xs:element name="appName" type="xs: string"/>
            <xs:element name="appLocation" type="xs: string"/>
         </xs:sequence>
      </xs:complexType>

      <xs:element name="WFException" type="xs: tException"/>
      <xs:complexType  name="tException">
         <xs:sequence>
            <xs:element name="name" type="xs: string"/>
            <xs:element name="message" type="xs: string"/>
         </xs:sequence>
      </xs:complexType>
  </xs:schema>
</types>
```

Figura 5.6. El documento completo del servicio web *InvokeApplication* (types)

```
<interface name = "InvokeAppInterface" >
    <fault name = "InvokeAppFault_WorkflowException" element = "ians: WFException"/>

   <operation name="opInvokeApp"
           pattern="http://www.w3.org/2004/03/wsdl/in-out"
           style=http://www.w3.org/ns/wsdl/style/iri
           xsdlx:safe = "true">
           <input messageLabel="In" element="ians: InvokeApp" />
           <output messageLabel="Out" element="ians: AppFound" />
           <outfault ref="tns: InvokeAppFault_WorkflowException" messageLabel="Out"/>
   </operation>
</interface>

<binding name="InvokeAppSOAPBinding"
    interface="tns: InvokeAppInterface"
    type="http://www.w3.org/ns/wsdl/soap"
    wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">

    <fault ref="tns: InvokeAppFault_WorkflowException"
    wsoap:code="soap:Sender"/>

    <operation ref="tns: opInvokeApp"
    wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
</binding>

<service name="InvokeAppService"
    interface="tns: InvokeAppInterface">

    <endpoint name="InvokeAppEndpoint"
           binding="tns: InvokeAppSOAPBinding"
           address ="http:..."/>
</service>

</description>
```

Figura 5.7. Documento completo del servicio web *InvokeApplication* (interface, binding y service)

CAPÍTULO 6.

Invocación de Aplicaciones Externas con Reglas de Transformación de Grafos

En la actualidad los servicios web son registrados a través del protocolo UDDI [UDDI] para ser localizados y usados por las aplicaciones. Este protocolo requiere la descripción sintáctica de la aplicación o servicio concreto que se quiere invocar y si hay más de una aplicación que brinda el mismo servicio, es necesario definir en tiempo de desarrollo cual se invocará.

Para permitir que el motor del SGFT pueda seleccionar la aplicación más conveniente, entre varias que semánticamente se comportan igual, aunque tienen diferente descripción sintáctica, se propone optimizar esta selección de servicios usando Reglas de Transformación de Grafos e incorporando los atributos de calidad a la selección automática. Las Reglas de Transformación de Grafos permiten proveer una especificación semántica más precisa de un servicio web, y establecer una correspondencia entre los requerimientos del usuario y los servicios web disponibles, facilitando así su descubrimiento automático, en tiempo de ejecución. En esta especificación semántica más precisa del servicio web es posible describir también los atributos de calidad de los mismos, para poder incorporar los

aspectos no funcionales a la selección del mejor servicio web que satisface los requerimientos del usuario del SGFT.

La estructura de este capítulo es la siguiente. La sección 6.1 presenta los conceptos generales de la técnica de transformación de grafos. La sección 6.2 establece una relación entre dicha técnica y el modelado orientado a objetos, el cual utiliza el lenguaje de modelado UML. La sección 6.3 describe el esquema de selección automática de servicios web con reglas de transformación de grafos y la sección 6.4 describe el servicio web ofrecido por un proveedor y la formulación del requerimiento de un usuario del SGFT como grafos con atributos, para establecer la correspondencia utilizando esta técnica.

6.1. Las Reglas de Transformación de Grafos

La transformación de grafos es una técnica gráfica, formal, declarativa y de alto nivel muy usada para transformación y simulación de modelos, chequeos de consistencia entre modelos o vistas y optimización de diversos contextos [BH02]. La transformación de grafos es un mecanismo formal para la manipulación de grafos basado en reglas. En analogía a las gramáticas de Chomski sobre cadenas de caracteres, las reglas de grafos están formadas por una parte izquierda y una parte derecha que contienen grafos. La transformación de grafos es una modificación de estos grafos basada en las reglas, como se muestra en la Figura 6.1.

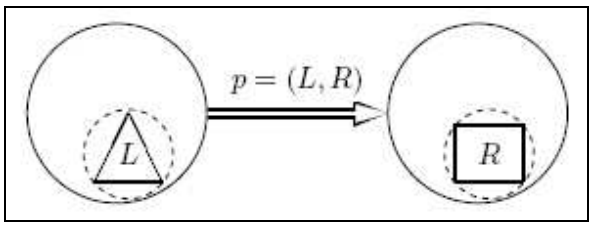


Figura 6.1. Modificación de grafos basada en reglas

Intuitivamente, una regla de transformación de grafos o producción p = (L, R) contiene un par de grafos (L, R), denominados la parte izquierda L y la parte derecha R. Aplicar la regla p = (L, R) significa encontrar una ocurrencia de L en el grafo anfitrión y reemplazarla por R, obteniendo el grafo destino. Para aplicar una regla al grafo anfitrión, se debe encontrar un morfismo de correspondencia entre la parte derecha de la regla y el grafo. Si

dicho morfismo es encontrado, por un proceso de derivación, la regla se aplica sustituyendo la imagen de morfismo encontrado en el grafo, por la parte derecha de la regla. Dicho de otro modo, una regla de transformación de grafos, también llamada producción, (L, K, R) consiste en tres grafos L, K y R. Una producción (L, K, R) es aplicable a un grafo G si G contiene un subgrafo que es una imagen de L. Una producción (L, K, R) es aplicable a un grafo G si L es un subgrafo de G. El grafo L constituye la parte izquierda de la producción, y formula las condiciones bajo las cuales resulta aplicable la producción, o sea, es un subconjunto del conjunto de objetos (nodos y arcos) del universo. El grafo de contacto K, que generalmente es un subgrafo de L y de R, describe los subobjetos de la parte izquierda que se deben preservar en el proceso de aplicación de la producción. Por lo tanto, la diferencia entre L y K: L\K, contiene a todos aquellos subobjetos que se deben eliminar al aplicar la producción ((1) Figura 6.2). Análogamente, la diferencia R\K contiene a todos aquellos subobjetos que se deberán agregar al aplicar la producción ((2) Figura 6.2). Este grafo intermedio K describe el contexto en el cual se integran los subobjetos agregados.

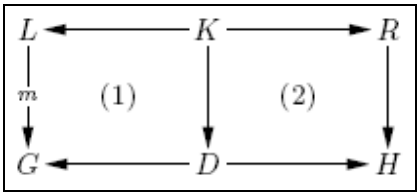


Figura 6.2. Transformación de Grafos

6.2. Las Reglas de Transformación de Grafos en el Modelado Orientado a Objetos

Los diagramas son muy usados para describir estructuras y sistemas complejos y modelar conceptos e ideas de una forma directa e intuitiva. Ellos proveen una simple y poderosa herramienta para modelar una gran variedad de problemas típicos de la ingeniería de software. Los lenguajes de modelado gráfico estándar, como UML, poseen muchas ventajas, son fáciles de entender y de usar, proveen un cierto nivel de formalismo, son de propósito general y están soportados por diversas herramientas. Los diagramas de clases de UML son ampliamente aceptados para visualizar conceptos y estructuras de un dominio particular, y proveer un entendimiento común a los desarrolladores.

Desde un punto de vista matemático, los modelos generados con lenguajes de modelado gráfico pueden considerarse grafos. La conversión de modelos a grafos es prácticamente directa. Esto hace que las gramáticas de grafos sean fácilmente aplicables en la definición de transformación entre modelos y en el análisis de compatibilidad de modelos. Siguiendo esta estrategia, una transformación puede definirse mediante una gramática de grafos que toma como entrada un modelo expresado con elementos de un metamodelo específico (el metamodelo de UML) y devuelve otro modelo expresado con elementos de otro metamodelo o inclusive el mismo.

Las gramáticas de grafos poseen ventajas respecto a otras técnicas para la transformación de modelos:

- Poseen una sólida base matemática con una larga tradición.
- Permiten una representación gráfica, lo que facilita la especificación y comprensión de las transformaciones de forma intuitiva.
- Existen algoritmos y herramientas que permiten su aplicación.

En el modelado orientado a objetos los grafos ocurren a dos niveles: a nivel de tipos (los diagramas de clases) y a nivel de instancia (diagramas de objetos). Esta noción se puede generalizar en el concepto de grafos tipados, los cuales pueden verse como una representación abstracta de los diagramas de clases de UML (Unified Modeling Language) [UML]. Los diagramas de objetos son grafos equipados con una correspondencia a los grafos tipados (que preserva la estructura), formalmente expresada como un homomorfismo de grafos.

En este contexto, una regla de transformación de grafos está formada por una parte izquierda y una parte derecha. En su parte izquierda define un subgrafo anfitrión y en su parte derecha define un subgrafo destino o de sustitución. Cuando se aplica una regla a un grafo de entrada, las ocurrencias de la parte izquierda de la regla son sustituidas por la parte derecha, tal cual lo expresado en la sección anterior.

6.3. Selección automática de servicios web con Reglas de Transformación de Grafos

La Figura 6.3 presenta un esquema de la propuesta de invocación de servicios web con Reglas de Transformación de Grafos. En la misma se visualizan los servicios web

ofrecidos por los proveedores, a través de UDDI, los servicios web solicitados a través de la invocación de aplicaciones externas, y la correspondencia realizada por el motor del SGFT, y la selección del servicio web que finalmente se invocará. Aquí se puede observar el comportamiento interno del SGFT a la hora de seleccionar una aplicación externa, en particular, cómo hace el motor del SGFT para elegir la aplicación e invocarla, valiéndose de los servicios web disponibles en la web. En el esquema propuesto, el componente principal es el motor del SGFT, quien actúa como una agente de descubrimiento de servicios, basándose en los requerimientos funcionales y las características de calidad especificadas por el usuario, y los requerimientos mínimos estipulados por el propio motor. Así, el motor establece un ranking de los servicios web disponibles y elige el más adecuado. También el motor es el responsable de reunir y procesar la información de los servicios web ofrecidos por los proveedores, y mantenerla actualizada a medida que se realizan las invocaciones.

Para realizar la correspondencia y selección final, el motor trabaja en dos etapas.

- En la *primera etapa* (paso 1) establece la correspondencia teniendo en cuenta los grafos de la pre y pos condición del solicitante y los grafos de las pre y poscondiciones de los proveedores, especificados con Reglas de Transformación de Grafos. Del solicitante, además del tipo de aplicación, los parámetros y los requerimientos adicionales, se tienen en cuenta los requerimientos no funcionales. Estos se refieren a las características de calidad del servicio web (performance, disponibilidad, fiabilidad, precisión, robustez, reputación). El solicitante detalla los atributos de calidad requeridos en el documento WSDL de la operación *invokeApp*. Los atributos de calidad de los servicios web ofrecidos por los proveedores son monitoreados por el motor, y registrados internamente para poder realizar la correspondencia.

  Como resultado de la correspondencia, el motor obtiene una lista de los servicios web candidatos a invocarse.

- En la *segunda etapa* (pasos 2 y 3), el motor del SGFT analiza cada uno de los servicios web candidatos, y considera dos aspectos más:

    - *Historia de las ejecuciones previas*: es la información que le sirve al motor del SGFT para saber cuáles fueron las aplicaciones que en ocasiones anteriores, y bajo las mismas condiciones, se eligieron para invocarse. Esta información es almacenada por el propio motor del SGFT, a medida que se van invocando las aplicaciones e incluye, por un lado, datos como el

nombre de la aplicación, el tipo, su ubicación, el sistema operativo y la fecha. Por otro lado, el motor del SGFT evalúa también las características de calidad de dichas aplicaciones y las registra en este historial (monitorea el nivel de calidad par asegurar que se respetan los requerimientos del solicitante).

- *Características de calidad del propio motor del SGFT*: el motor del SGFT posee un registro de las características de calidad mínimas que acepta al invocar una aplicación. Este registro es adicional a las características de calidad requeridas por el solicitante, especificadas en la precondición del mismo. Su objetivo es permitir el análisis de aquellas características que no fueron explícitamente requeridas por el solicitante, en base a los requerimientos mínimos que el mismo motor del SGFT tiene preestablecidos.

Con todos estos elementos, el motor del SGFT finalmente elige uno de los servicios web candidatos (paso 4), lo invoca y lo registra en la historia de la ejecuciones previas.

Esta formalización de la especificación como un grafo con atributos permite realizar una correspondencia con los servicios web disponibles en la red que cumplen con el morfismo establecido, y seleccionar el servicio web más adecuado según los requerimientos de los usuarios del SGFT, tanto funcionales como no funcionales.

6.4. Su utilización en la invocación de una aplicación externa

En los capítulos 4 y 5 se presentó la especificación del servicio web *InvokeApp*, que permite la invocación de una aplicación externa a un sistema de gestión de flujo de trabajo, independizándose de la ubicación exacta de la misma y facilitando así la distribución de las aplicaciones en la red. El objetivo ahora es analizar la compatibilidad de los servicios web provistos en la red y los requeridos de un usuario del SGFT.

Se comenzará especificando la interfaz de los servicios web solicitados y ofrecidos con grafos en la sección 6.4.1 y se proseguirá con el análisis de la correspondencia entre los grafos generados, en la sección 6.4.2.

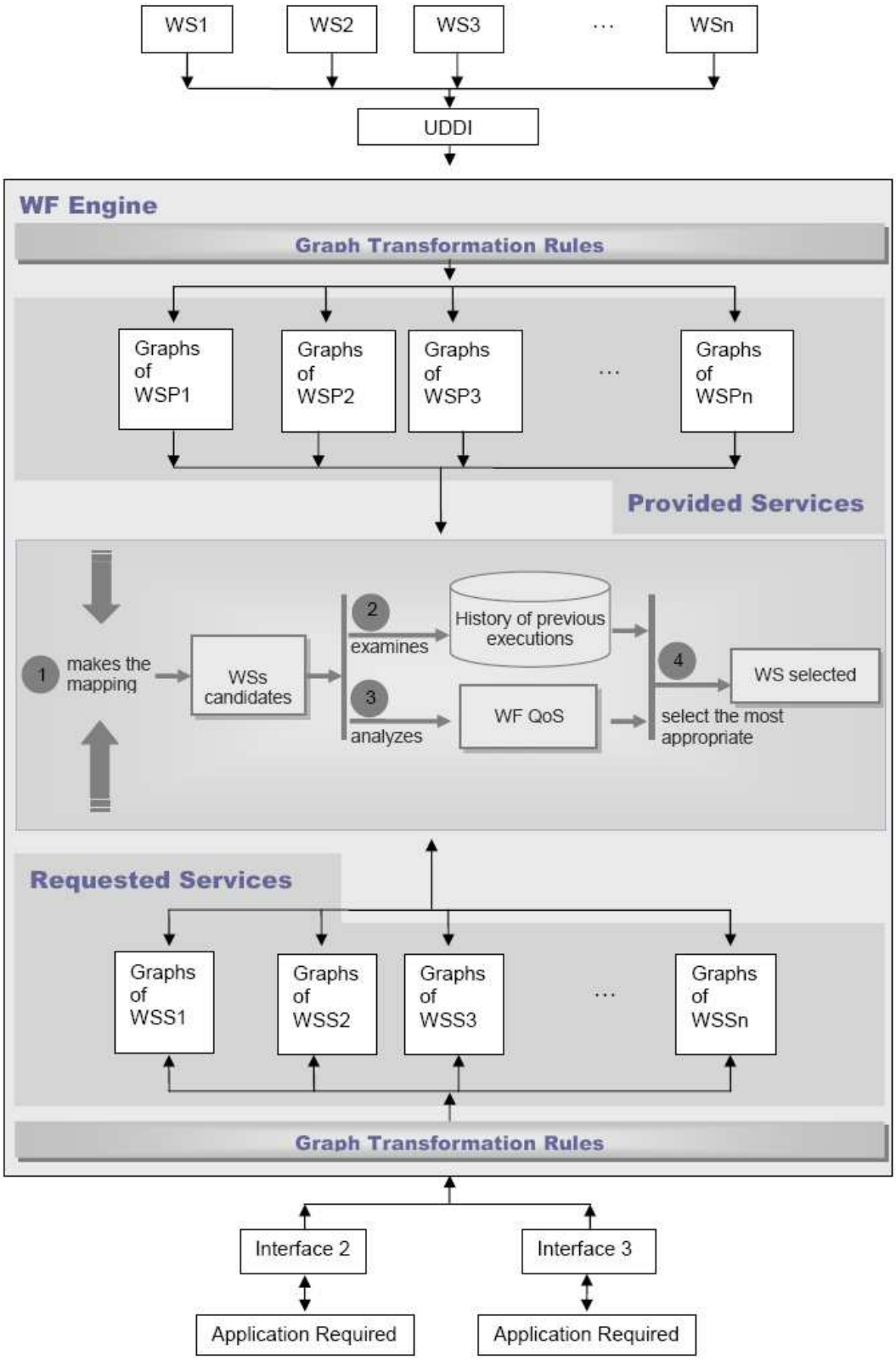


Figura 6.3. Esquema de selección de Servicios Web

6.4.1 Grafos del solicitante y los proveedores de servicios web

Como se expresó en el capítulo 4, uno de los elementos que debe definirse al especificar un servicio web es la *interface*, la cual describe la secuencia de mensajes que el servicio envía o recibe, agrupándolos en una colección de operaciones, que son accesibles desde la red a través de mensajes XML estandarizados. En base a los conceptos antes mencionados sobre Reglas de Transformación de Grafos en el modelado orientado a objetos y las propuestas de [CHH04] y [HHL03], se analiza la compatibilidad del comportamiento de estas operaciones que constituyen las interfaces de los servicios web.

Estas interfaces sólo contienen información sobre la estructura de la operación. El comportamiento de la misma puede representarse mediante *contratos*, expresados usando Reglas de Transformación de Grafos. Un contrato consiste de una precondición especificando el estado del sistema antes de que el comportamiento sea ejecutado, y una poscondición especificando el estado del sistema después de la ejecución del comportamiento. Usando UML como notación visual para describir los contratos, y Reglas de Transformación de Grafos para establecer la correspondencia, se describe el servicio web ofrecido por un proveedor y la formulación del requerimiento de un usuario del SGFT, expresados como grafos con atributos (Figuras 6.4 a 6.7).

En la Figura 6.4, el grafo representa la precondición del servicio solicitado, es decir, el estado del sistema antes de la ejecución del servicio web. Este grafo tiene una relación directa con la especificación WSDL del elemento *InvokeApp*, dentro del elemento *types* del servicio web (capítulo 4, sección 4.2.1), donde se define la información que necesita el motor del SGFT para poder invocar una aplicación externa. También aparece la clase QoSReq, que representa los atributos de calidad requeridos del servicio web, los cuales se agregaron como un nuevo elemento en la especificación de *InvokeApp*, en el capítulo 5, sección 5.2.

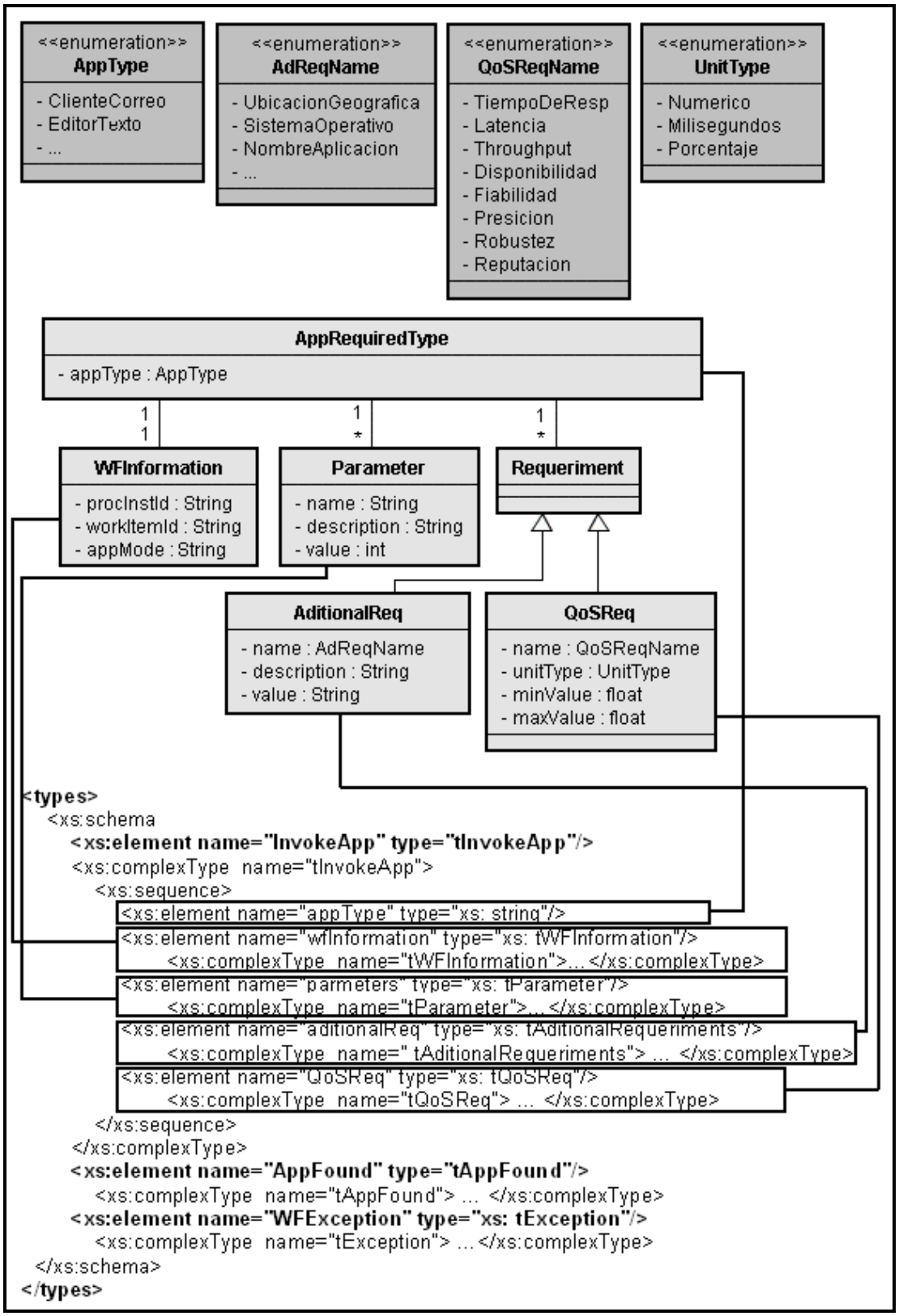


Figura 6.4. Grafo de la precondición del servicio solicitado

En la Figura 6.5, el grafo representa la poscondición del servicio solicitado, es decir, la situación después de satisfacerse la ejecución del servicio web, desde el punto de vista del solicitante. El grafo tiene una relación directa con la especificación WSDL del elemento *AppFound*, dentro del elemento *types* del servicio web (capítulo 4, sección 4.2.1), donde se define el resultado de invocar una aplicación externa.

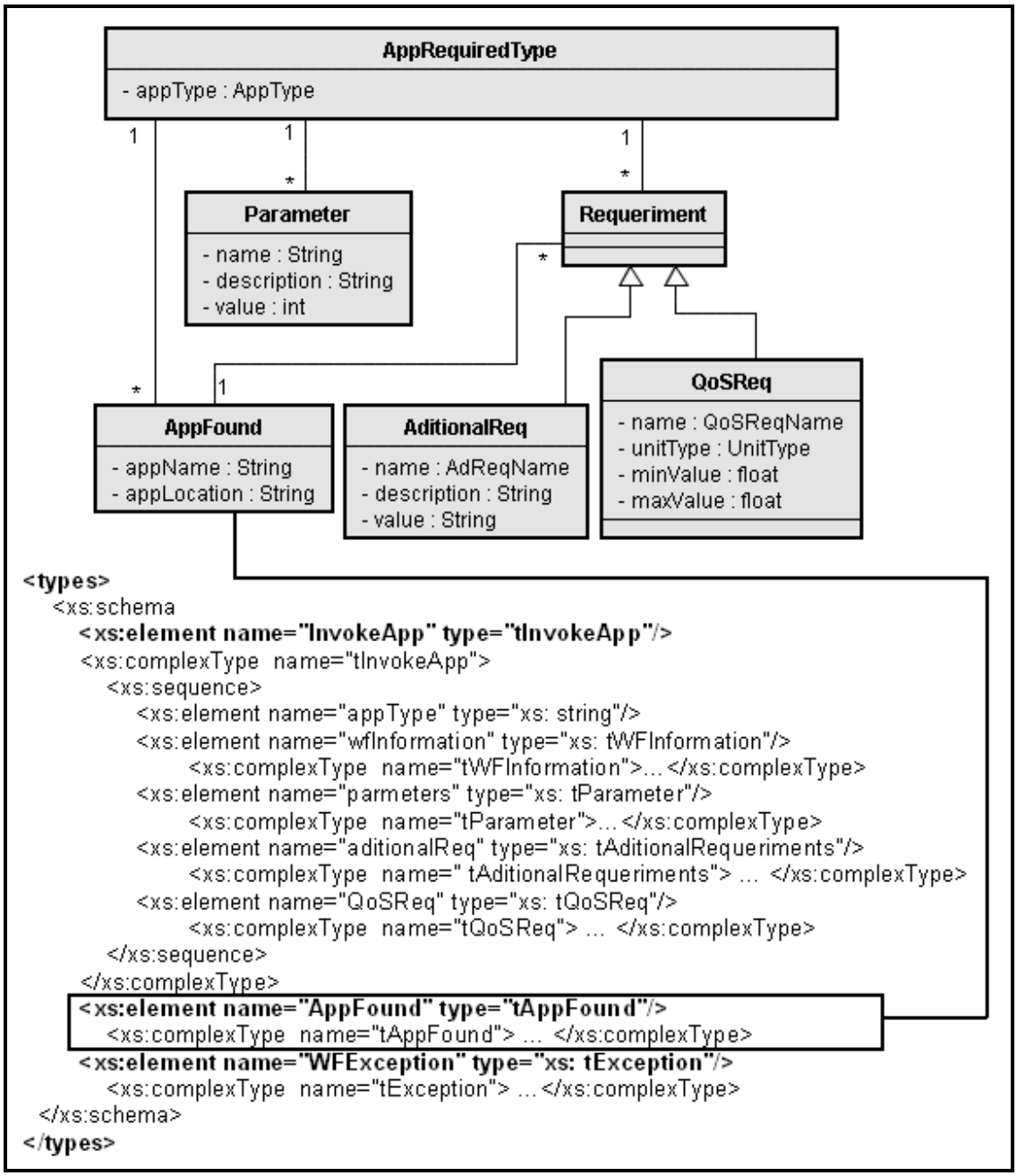


Figura 6.5. Grafos de la poscondición del servicio solicitado

Análogamente a la Figura 6.4, el grafo de la Figura 6.6 también representa una precondición, en este caso la del servicio web ofrecido por el proveedor. Se pueden observar las mismas clases que en la Figura 6.4, aunque no aparece la clase WFInformation, ya que la información definida en la misma no es relevante para el proveedor del servicio web, por lo que no se tiene en cuenta.

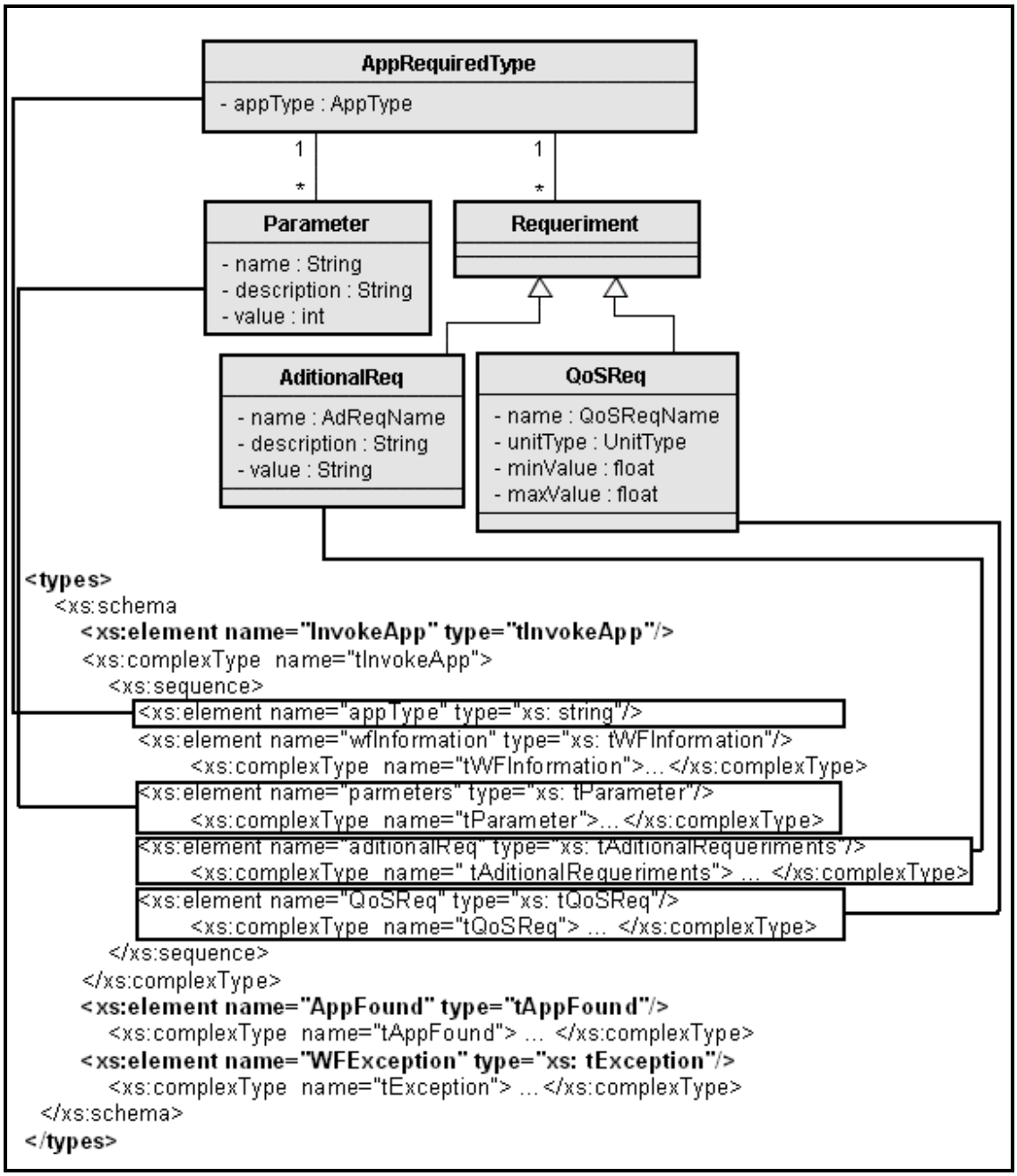


Figura 6.6. Grafos de la precondición del servicio ofrecido por el proveedor

Finalmente, la Figura 6.7 muestra el grafo que representa la poscondición del servicio ofrecido, es decir, la situación después de satisfacerse la ejecución del servicio web desde el punto de vista del proveedor. Aquí aparece la misma clase *AppFound* presente en la precondición del solicitante, y se agrega la clase *ErrorReturnValue*, que representa el elemento *WFException*, también definido en la especificación WSDL del elemento *InvokeApp* (capítulo 4, sección 4.2.1).

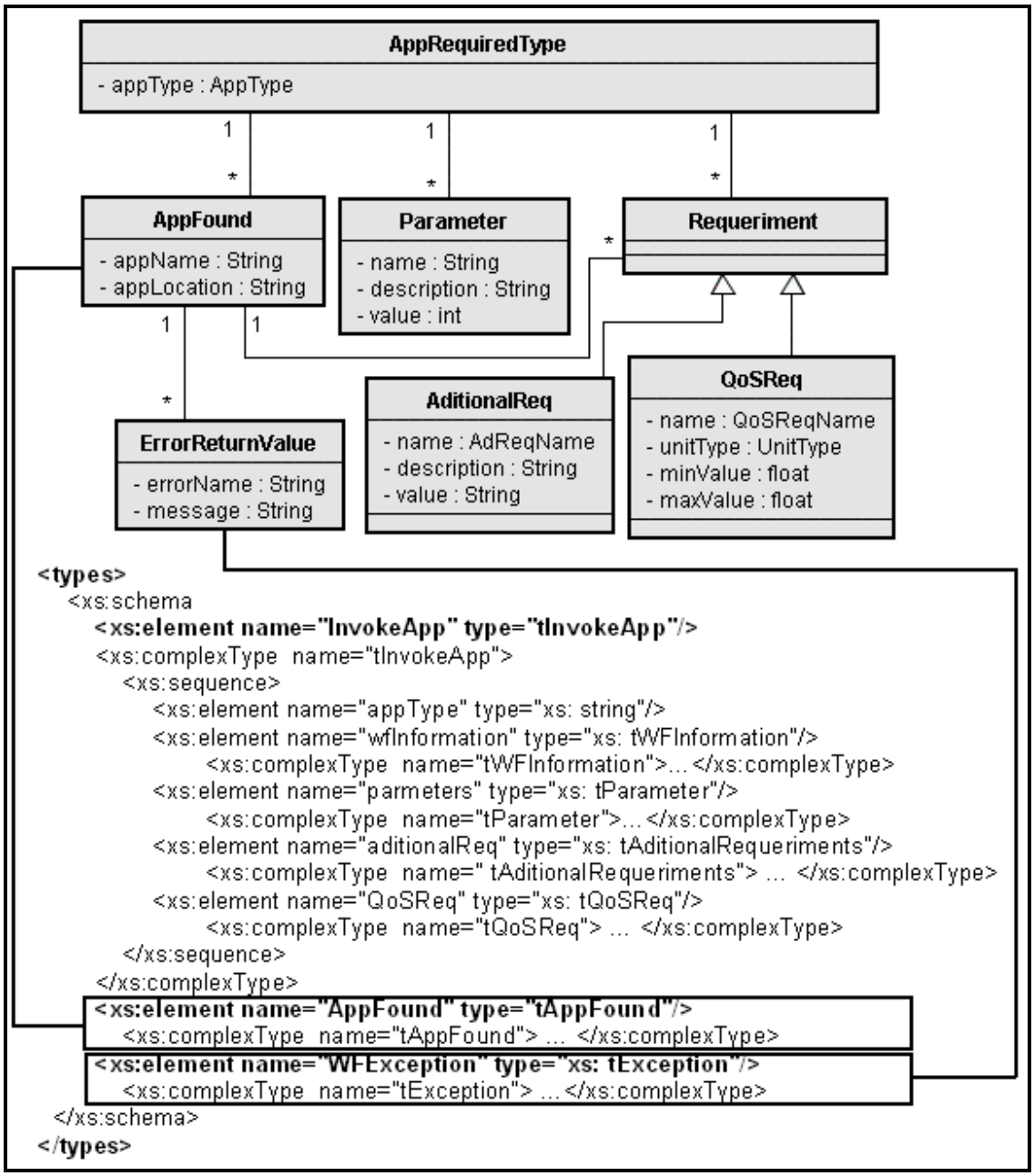


Figura 6.7. Grafos de la poscondición del servicio ofrecido por el proveedor

Estos 4 grafos, representan las pre y poscondiciones del servicio web solicitado y los servicios web ofrecidos por los proveedores. Estos grafos forman la parte izquierda y la parte derecha de las Reglas de Transformación de Grafos.

6.4.2 Análisis de la correspondencia entre los grafos generados

Ahora, para decidir automáticamente si un servicio web ofrecido por un proveedor satisface la demanda del solicitante, es necesario comparar los grafos generados por dicho solicitante y proveedor. Se formaliza usando relaciones de subgrafos. Si la precondición del proveedor es un subgrafo de la precondición del solicitante, entonces el proveedor provee toda la información necesaria para ejecutar el servicio web.

En la Figura 6.8, parte derecha, se observa que la precondición del proveedor contiene cinco clases: *AppRequiredType*, *Parameter*, *Requeriment*, *AditionalReq* y *QoSReq*. Por su parte, la precondición del solicitante (parte izquierda) posee estas mismas cinco clases, más la clase *WFInformation*, la cual almacena información que sólo es importante para el motor del SGFT. Por ello esta clase no esta especificada en el proveedor.

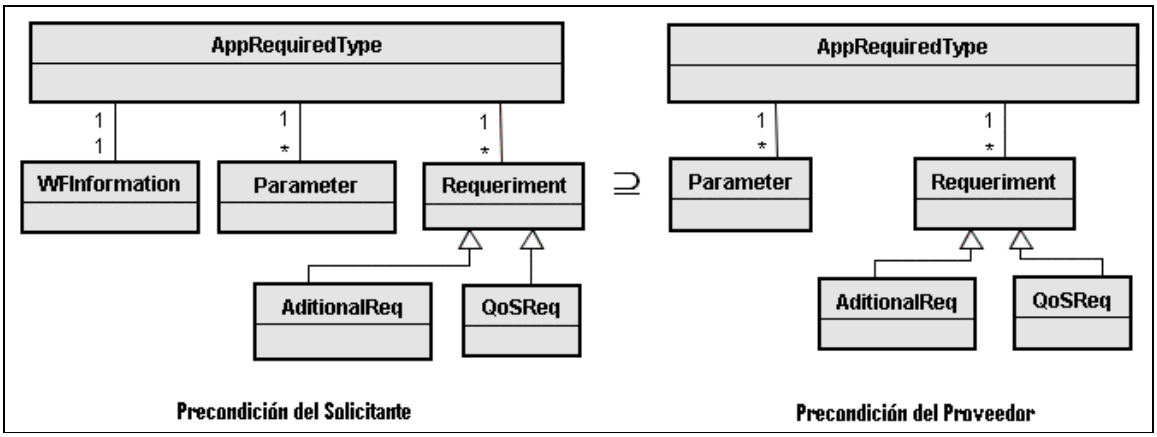


Figura 6.8. Grafos de las precondiciones del solicitante y el proveedor

Por otro lado, si la poscondición del solicitante es un subgrafo de la poscondición del proveedor, entonces el servicio genera todos los efectos esperados por el solicitante, además de algunos efectos adicionales, como en este caso los errores posibles.

En la Figura 6.9, parte derecha, se observa que la poscondición del solicitante contiene todas las clases especificadas en la poscondición del proveedor (*AppRequiredType*,

Parameter, *Requeriment*, *AditionalReq*, *QoSReq*, *AppFound*), mientras que en la Figura 6.9, parte derecha, aparecen estas mismas seis clases más la clase denominada ErrorReturnValue, la cual almacena la información suministrada en caso de ocurrir un error durante la invocación.

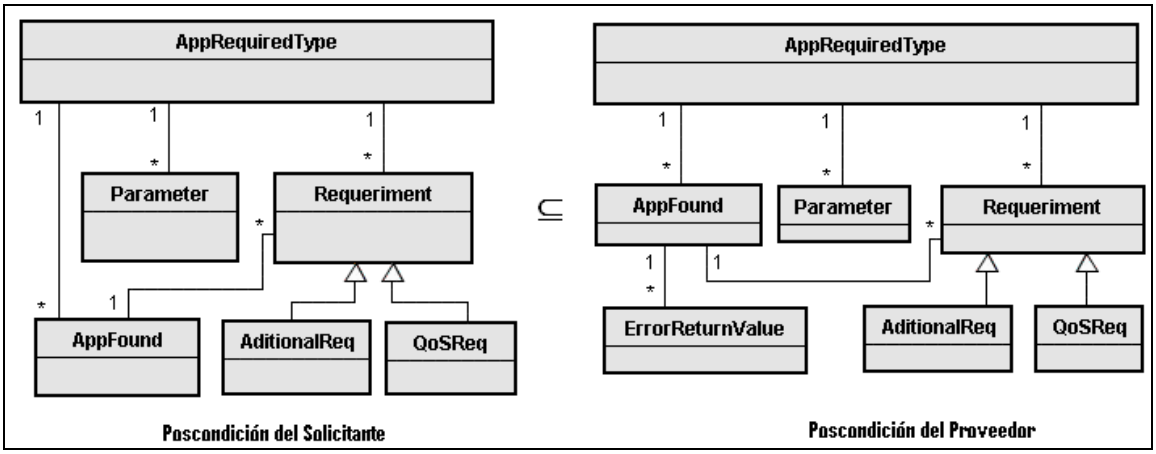


Figura 6.9. Grafos de las poscondiciones del solicitante y el proveedor

Sobre las instancias de estos cuatro grafos, correspondientes a especificaciones concretas de servicios web solicitados y ofrecidos, es posible analizar si se cumple con el morfismo. En el siguiente capítulo se presenta un caso de estudio donde se puede apreciar, con un ejemplo concreto, cómo quedan especificadas las instancias de estos grafos y cómo se establece la correspondencia.

CAPÍTULO 7.

Caso de Estudio: Invocación de Aplicaciones Externas en OpenUP/Basic

En este capítulo se presenta un caso de estudio que plantea el uso de servicios web seleccionados automáticamente en la invocación de aplicaciones externas al SGFT, a través de su aplicación en el marco de las tareas involucradas en el proceso de desarrollo de software denominado OpenUP/Basic.

La estructura del capítulo es la siguiente. La sección 7.1 describe las características generales del proceso de desarrollo de software OpenUp. La sección 7.2 presenta el proceso OpenUp/Basic, destinado al desarrollo ágil de software con equipos de trabajo pequeños y bien organizados. Contiene cinco subsecciones, las cuales describen los roles, disciplinas, tareas, artefactos y ciclo de vida del proceso, respectivamente. La sección 7.3 presenta la aplicación de la selección automática de servicios web para invocar aplicaciones externas al SGFT, utilizadas durante el proceso de desarrollo de software.

7.1. El proceso de desarrollo de software OpenUP

OpenUP es una familia de plug-ins de procesos de fuente abierta [EPF]. OpenUP/Basic es el proceso principal en OpenUP y está orientado a un equipo pequeño y organizado. OpenUP se basa en *Racional Unified Process* (RUP) [RUP], el cual a su vez se funda en el Proceso Unificado de Desarrollo de Software (The Unified Software Development Process) [JBR99].

OpenUP es un proceso de desarrollo de software mínimo, en el sentido de que sólo el contenido fundamental está incluido [Balduino07]. Así, no provee una guía sobre algunos tópicos que un proyecto podría tratar, tales como tamaño grande los equipos, conformidad, situaciones contractuales, cuestiones de seguridad, aplicaciones con misiones críticas, cuestiones específicas de la tecnología, etc. Sin embargo, OpenUP es completo en el sentido de que puede manifestarse como un proceso completo para construir un sistema. Para direccionar las necesidades que no son cubiertas en este proceso, OpenUP es extensible, es decir, puede ser usado como una base sobre la cual agregar contenido o adaptarlo como sea necesario.

OpenUP es un proceso ágil. La mayoría de las prácticas ágiles reconocidas tienen como objetivo mantener al equipo de desarrollo altamente comunicado, para fomentar el intercambio de información y mantener un entendimiento compartido del proyecto, sus objetivos, alcance y avances.

Si bien OpenUP mantiene las mismas características del RUP, en cuanto al desarrollo iterativo, la definición de casos de uso y escenarios de conducción de desarrollo, la gestión de riesgos, y el enfoque centrado en la arquitectura; define un proceso iterativo mínimo, completo y extensible, es decir, provee la mínima cantidad de procesos para un equipo pequeño, y puede ser usado como está o ser extendido y personalizado para propósitos específicos. OpenUP se basa en los casos de uso y escenarios, la gestión de riesgos y un enfoque centrado en la arquitectura.

En OpenUP un proyecto se divide en iteraciones, las cuales son planificadas en un intervalo definido de tiempo que no supera las pocas semanas. El OpenUP tiene elementos que ayudan a los equipos de trabajo a enfocar los esfuerzos a través del ciclo de vida de cada iteración de tal forma que se puedan distribuir funcionalidades incrementales de una manera predecible, y obtener una versión totalmente probada y funcional al final de cada iteración.

OpenUP se caracteriza por cuatro principios básicos que se soportan mutuamente:

- ***Colaboración*** para alinear los intereses y un entendimiento compartido
- ***Balance*** para confrontar las prioridades (necesidades y costos técnicos) para maximizar el valor para los stakeholders
- ***Enfoque*** en articular la arquitectura para facilitar la colaboración técnica, reducir los riesgos y minimizar excesos y trabajo extra
- ***Evolución*** continua para reducir riesgos, demostrar resultados y obtener retroalimentación de los clientes

En OpenUP el trabajo se organiza a nivel personal, del equipo y de los participantes, como se puede observar en la Figura 7.1.

A nivel personal, los miembros del equipo contribuyen con su trabajo en ***micro-incrementos***, los cuales representan el resultado de pocas horas o pocos días de trabajo. La aplicación evoluciona un micro-incremento a la vez y el progreso se visualiza efectivamente cada día. Los miembros del equipo abiertamente comparten su progreso diario, lo cual incrementa la visibilidad de la labor realizada, la veracidad y el trabajo en equipo.

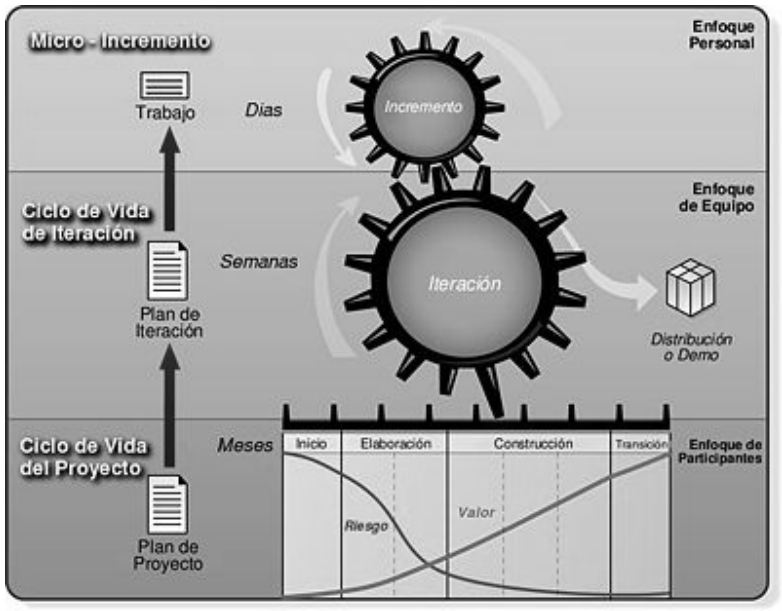


Figura 7.1. Capas de OpenUP: micro-incrementos, ciclo de vida de la iteración y ciclo de vida del proyecto

7.2. OpenUP/Basic

OpenUP/Basic es un proceso de desarrollo de OpenUP destinado al desarrollo de software de código abierto diseñado para pequeños equipos bien organizados, interesados en realizar un desarrollo ágil. Es un proceso iterativo mínimo, completo y extensible. OpenUP/Basic esta diseñado para equipos pequeños, trabajando juntos en la misma localidad. El equipo tiene que participar a plenitud de la interacción diaria cara a cara. Los miembros del equipo incluyen los stakeholders, desarrolladores, arquitectos, gestores de proyecto y testeadores. Ellos participan en una colaboración significativa, tomando sus propias decisiones en cuanto a lo que se necesita realizar, cuales son las prioridades, y la mejor manera de abordar los requerimientos de los stakeholders. La organización debe apoyar al equipo permitiéndoles esta responsabilidad. Los miembros del equipo colaboran ampliamente. La presencia de los stakeholders como miembros del equipo es crítica para realizar exitosamente OpenUP/Basic. Los miembros del equipo participan a diario en las reuniones stand-up para comunicar el estado y sus asuntos. Los problemas se abordan fuera de las reuniones diarias.

OpenUP/Basic se enfoca en reducir significativamente el riesgo de manera temprana en el ciclo de vida. Esto requiere unas reuniones regulares de revisión de los riesgos y una implementación rigurosa de las estrategias de mitigación. Todo el trabajo será listado, seguido y asignado a través de la "*lista de items de trabajo*". Los miembros del equipo están en este único repositorio para todas las tareas que necesitan ser registradas y seguidas. Esto incluye todos los requerimientos de cambio, errores y requerimientos de los stakeholders. Los casos de uso son utilizados para elicitar y describir los requerimientos. Los miembros del equipo deben desarrollar habilidades para escribir buenos casos de uso. Los stakeholders son responsables de revisar y certificar que los requerimientos son correctos. Los casos de uso son desarrollados de manera colaborativa.

Los requisitos arquitectónicamente más importantes deben ser identificados y estabilizados dentro de la fase de *Elaboración* de tal forma que una arquitectura robusta sea creada, la cual será el corazón del sistema. Un cambio de un requisito arquitectónicamente significativo puede surgir posteriormente en el desarrollo, el cual debe ser abordado, pero el riesgo de que esto ocurra es reducido significativamente dentro de la iteración de *Elaboración*.

OpenUP/Basic no incluye contenido para el despliegue, gestión del cambio, o entorno (tal como personalizar este proceso o preparar el entorno de desarrollo). OpenUP/Basic se enfoca en un equipo único, y estas áreas son tratadas a nivel organizacional o empresarial. Revisar las extensiones de OpenUP que abordan estas áreas más ampliamente. OpenUP/Basic es un proceso de desarrollo de software que es mínimo, completo y extensible.

Los *Roles* desempeñan *Tareas* que consumen y producen *Artefactos*.

7.2.1. Roles

Las habilidades necesarias para el equipo de desarrollo se representan por los siguientes roles:

- ***Stakeholder***: representa el grupo de interés cuyas necesidades deben ser satisfechas por el proyecto. Este es un rol que puede ser jugado por cualquiera que este (o potencialmente pueda estar) afectado por el resultado del proyecto.
- ***Analista***: representa las preocupaciones del cliente y el usuario final reunidas por los stakeholders para entender el problema y capturar las prioridades de los requerimientos.
- ***Arquitecto***: es el responsable de diseñar la arquitectura del software, lo cual incluye tomar decisiones técnicas clave que regirán el diseño y la implementación del proyecto.
- ***Desarrollador***: es el responsable de desarrollar una parte del sistema, incluyendo un diseño que se ajuste a la arquitectura, la implementación, la prueba de unidad, y la integración de los componentes que forman parte de la solución.
- ***Testeador***: es responsable de identificar, definir, implementar y conducir las pruebas necesarias, así como también registrar la salida de las pruebas y analizar los resultados.
- ***Gestor del Proyecto***: planifica el proyecto en colaboración con los stakeholders y el equipo, coordina las interacciones con los stakeholders y mantiene al equipo del proyecto focalizado en alcanzar los objetivos planteados.
- ***Cualquier Rol***: cualquiera en el equipo puede cumplir este rol para llevar a cabo tareas generales.

7.2.2. Disciplinas

Las disciplinas que proveen organización a las tareas en OpenUP son:

- **Análisis y Diseño**: explica cómo crear un diseño de los requisitos que pueda ser implementado por los desarrolladores.
- **Configuración y Gestión de cambios**: explica cómo controlar los cambios en los artefactos, asegurando la evolución sincronizada de la serie de productos de trabajo que componen un sistema de software.
- **Implementación**: explica cómo implementar una solución técnica conforme al diseño, trabajando conforme a la arquitectura y soportando los requisitos. Con cada iteración, las tareas en esta disciplina evolucionan hacia una versión operacional del sistema (o parte de un sistema) más capaz y más estable.
- **Gestión del Proyecto**: el propósito de esta disciplina es:
  - Mantener al equipo focalizado en la entrega continua de productos de software probados para la evaluación de los participantes
  - Ayudar a priorizar la secuencia de trabajo
  - Ayudar a crear un ambiente de trabajo eficiente para maximizar la productividad del equipo
  - Mantener a los participantes y el equipo informados sobre el progreso del proyecto
  - Proveer un marco para la gestión de riesgos del proyectos y la continua adaptación al cambio
- **Requerimientos**: esta disciplina define las tareas mínimas requeridas para obtener, analizar, especificar, validar y gestionar los requerimientos del sistema.
- **Prueba**: esta disciplina define el mínimo conjunto de tareas requeridas para planificar, implementar, ejecutar y evaluar la prueba de un sistema. Una buena prueba se basa en la filosofía de realizar pruebas tempranas y realizar pruebas a menudo.

Otras disciplinas y áreas de interés son omitidas, tales como modelado de negocio, herramientas de gestión de la configuración de software y gestión de requerimientos

avanzados. Estas cuestiones se consideran innecesarias en los proyectos pequeños o son manejadas por otras áreas de la organización, fuera del equipo de desarrollo.

7.2.3. Tareas

Una tarea es una unidad de trabajo que un rol puede ser convocado a realizar. En OpenUp, existen 18 tareas que los roles desarrollan ya sea como desarrollador primario (el responsable de ejecutar la tarea) o desarrollador adicional (soportando y proveyendo información utilizada en la ejecución de la tarea).

7.2.4. Artefactos

Un artefacto es algo que puede ser producido, modificado, o usado por una tarea. Los roles son los responsables de crear y actualizar los artefactos. Los artefactos están sujetos al control de versiones a través del ciclo de vida del proyecto. Los 17 artefactos en OpenUP se consideran los artefactos esenciales que un proyecto debería usar para capturar información relacionada al proyecto y al producto. No hay obligación de capturar en artefactos formales. La información podría ser capturada en pizarras, notas de las reuniones, etc.

7.2.5. Ciclo de Vida

OpenUP/Basic es un proceso iterativo distribuido a través de cuatro fases: Inicio, Elaboración, Construcción y Transición. Cada fase consiste de una o más iteraciones, donde se trabaja en versiones estables del software que son desarrolladas y liberadas, la culminación de cada iteración representa un *milestone* menos para el proyecto y una contribución al éxito arquitectónico del hito mayor de la fase.

OpenUP ayuda al equipo a focalizar apropiadamente su esfuerzo a través del ***ciclo de vida de la iteración***, con el objetivo de entregar un valor incremental a los participantes de una manera predecible y fuertemente probada al final de cada iteración.

El ciclo de vida del proyecto provee a los interesados un mecanismo de supervisión y dirección para controlar los fundamentos del proyecto, su ámbito, la exposición a los riesgos, el aumento de valor y otros aspectos.

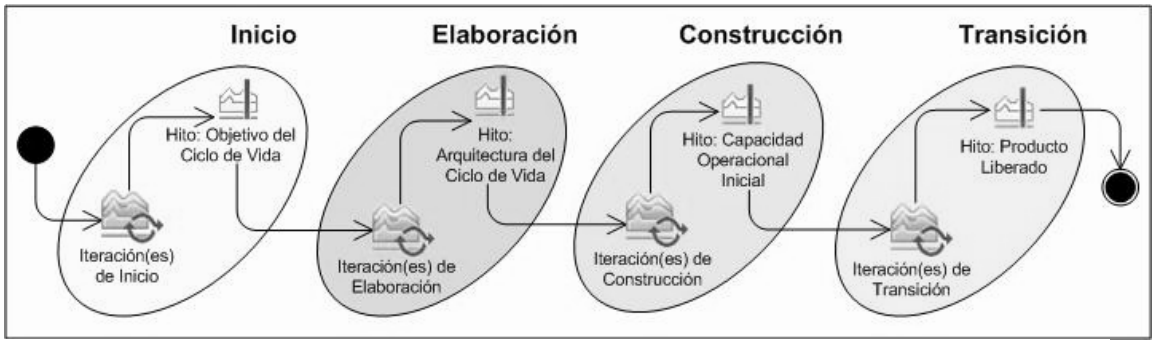


Figura 7.2. Fases del Ciclo de Vida en OpenUP/Basic

7.3. Los Servicios Web en la Invocación de Aplicaciones Externas en OpenUP/Basic

Para ejemplificar el uso de servicios web seleccionados automáticamente en la invocación de aplicaciones externas al SGFT, se plantea su aplicación en el marco de las tareas involucradas en el proceso de desarrollo de software OpenUP/Basic [OUPB].

7.3.1. Generación automática de código a partir de un modelo de diseño

Una de las disciplinas de OpenUP/Basic es la implementación. La implementación es la disciplina donde se obtiene una solución técnica consistente con el diseño, enmarcada en la arquitectura y que soporta los requerimientos. El objetivo principal es que la abstracción del diseño (clases, componentes, etc.) sea detallada para transformarse en la implementación. Algunos de los fines de esta disciplina son:

- La reutilización de código: la reutilización de código y las herramientas de generación de código producen código más robusto y son preferibles a escribir código a mano. El código existente a menudo ya ha sido altamente probado, resultando más estable y entendible que el código nuevo. El código fuente creado a partir de una herramienta de generación de código (como una herramienta de modelado visual) automatiza las tareas monótonas de codificación, como por ejemplo la creación de los *getters* y los *setters*.
- La transformación del diseño en implementación: la transformación del diseño en código implementa la estructura del sistema en el lenguaje fuente elegido. También implementa

el comportamiento del sistema definido en lo requerimientos funcionales. Implementar el comportamiento del sistema significa escribir el código que permite a las diferentes partes de la aplicación (clases o componentes) colaborar para realizar ese comportamiento del sistema. Existen varias técnicas para transformar el diseño en implementación automáticamente. Una de las técnicas para hacer esto es detallar los modelos y usarlos para generar una implementación. Tanto la estructura (definida en los diagramas de clases y paquetes) como el comportamiento (definido en los diagramas de estados y actividades) puede usarse para generar código ejecutable. Estos prototipos luego pueden ser más refinados, cuando sea necesario.

Una organización dedicada al desarrollo de software que utilice OpenUP/Basic como proceso de desarrollo necesitará generar código automáticamente a partir del diseño. El código obtenido será el esqueleto (prototipo) de las clases a implementar.

Para realizar esta actividad el motor del SGFT podría invocar una aplicación externa que se encargue de generar, a partir de los diagramas clases obtenidos en el diseño, el código correspondiente. La generación del código se podría realizar con cualquier aplicación disponible en la red, de manera de independizarse de la ubicación específica de dicha aplicación. El motor del SGFT, de acuerdo a los requerimientos recibidos, se encarga de realizar la correspondencia con los servicios web disponibles que satisfacen dichos requerimientos. Para ello, deberá transformar la formulación del requerimiento en grafos con atributos, para luego compararlos con los grafos que describen los servicios web ofrecidos por el proveedor.

7.3.1.1. Grafos de los servicios web del solicitante y los proveedores

Instanciando los grafos de la pre y poscondición del servicio web requerido, se obtienen los grafos resultantes para este ejemplo. Los grafos de las Figuras 7.3 y 7.4 representan instancias (diagrama de objetos) de los diagramas de clases que modelan la pre y poscondición del solicitante. En la Figura 7.3, el objeto *diagClases:Parameter*, el atributo *value* almacena una referencia a la ubicación física del archivo fuente. El objeto *rpta:QoSReq* especifica que el solicitante de la aplicación requiere que el tiempo de respuesta esté entre los valores 0,2 y 0,5 y el objeto *SO:AditionalReq* indica que el sistema operativo requerido es Windows XP.

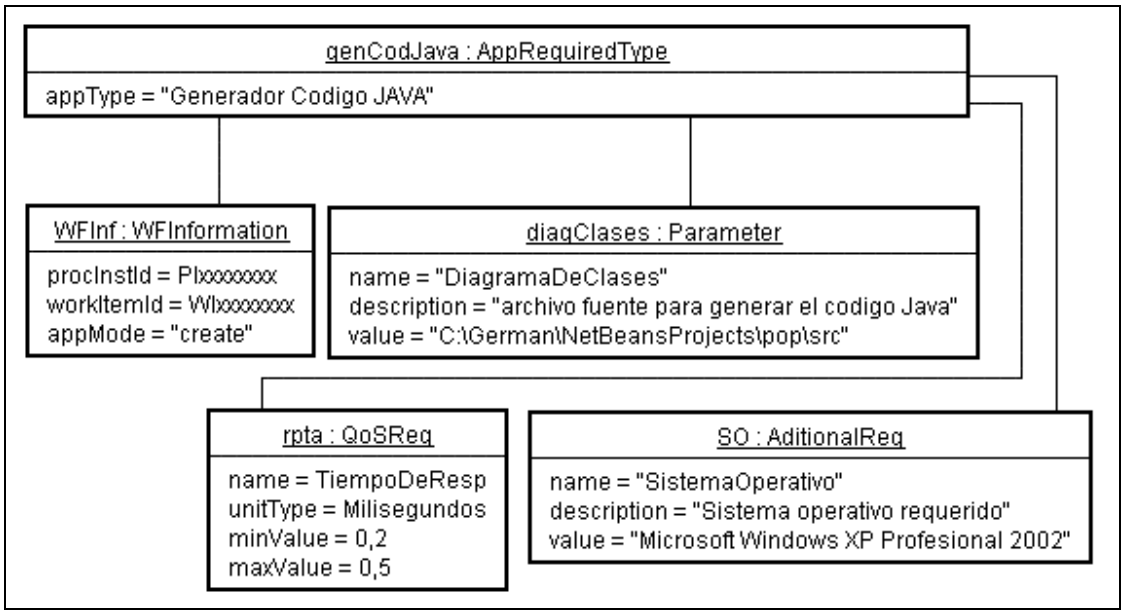


Figura 7.3. Grafo de la precondición del solicitante

En la Figura 7.4, aparece el objeto a:AppFound, el cual no tiene especificado ningún valor para sus atributos, ya que el solicitante no conoce las aplicaciones que cumplirán sus requerimientos.

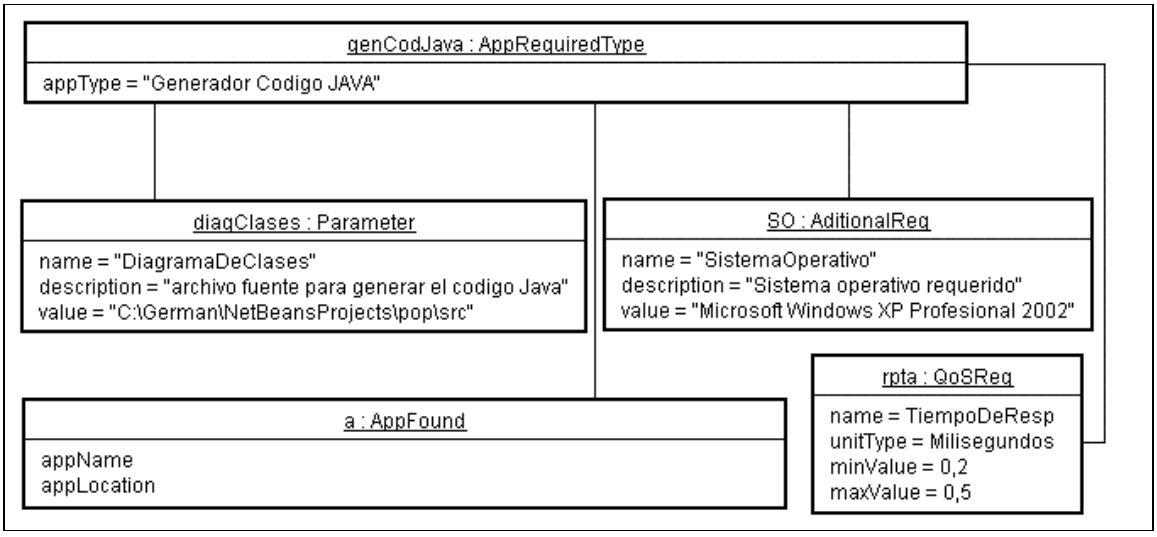


Figura 7.4. Grafo de la poscondición del solicitante

Los grafos de las Figuras 7.5, 7.6, 7.7 y 7.8 representan instancias (diagrama de objetos) de los diagramas de clases que modelan la pre y poscondición de cada uno de los dos proveedores que poseen servicios web para invocar una aplicación de generación de código Java. En la Figura 7.5, los objetos *:QoSReq* y *:AditionalReq* almacenan la información que el

proveedor1 necesita conocer del servicio web requerido, en cuanto al parámetro de entrada y al tiempo de respuesta y sistema operativo.

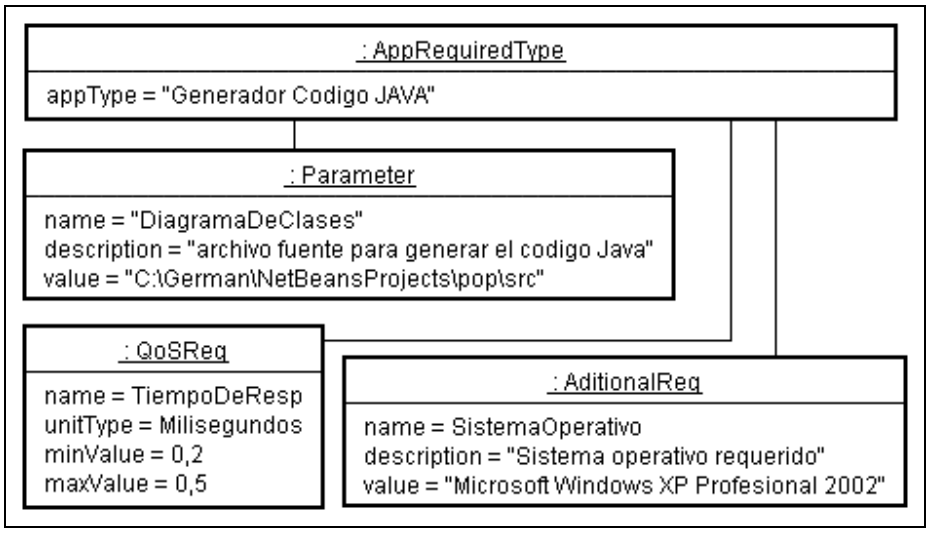


Figura 7.5. Grafo de la precondición del proveedor1

Por su parte, en la Figura 7.6, los objetos *:QoSReq* y *:AditionalReq* almacenan la información del *proveedor2* sobre el tiempo de respuesta y el sistema operativo de su servicio web ofrecido. Aparece además un objeto *a1T:QoSReq*, que almacena el throughput de la aplicación *RationalSoftwareArchitect.* Si bien sobre este no ha sido un requerimiento especificado por el solicitante, la información puede resultar útil para el motor del SGFT, en el caso de necesitar algún otro elemento para seleccionar el servicio web más adecuado.

En la Figura 7.7, los objetos *:QoSReq* y *:AditionalReq* almacenan la información que el *proveedor2* necesita conocer del servicio web requerido, en cuanto al parámetro de entrada y al tiempo de respuesta y sistema operativo. El objeto *a1QR:QoSReq* indica que el tiempo de respuesta para invocar la aplicación *RationalSoftwareArchitect* es de 0,3 milisegundos.

Finalmente, en la Figura 7.8, los objetos *:QoSReq* y *:AditionalReq* almacenan la información del *proveedor2* sobre el tiempo de respuesta y el sistema operativo de su servicio web ofrecido. El objeto *a2QR:QoSReq* indica que el tiempo de respuesta para invocar la aplicación *QuickUML* es de 0,5 milisegundos. Si bien este valor es un tanto superior al de la aplicación a1, igualmente se encuentra dentro de los valores requeridos por el solicitante. Al igual que en la poscondición del proveedor2, aparece un objeto nuevo *a2D:QoSReq*, que en este caso almacena el porcentaje de disponibilidad de la aplicación *QuickUML.*

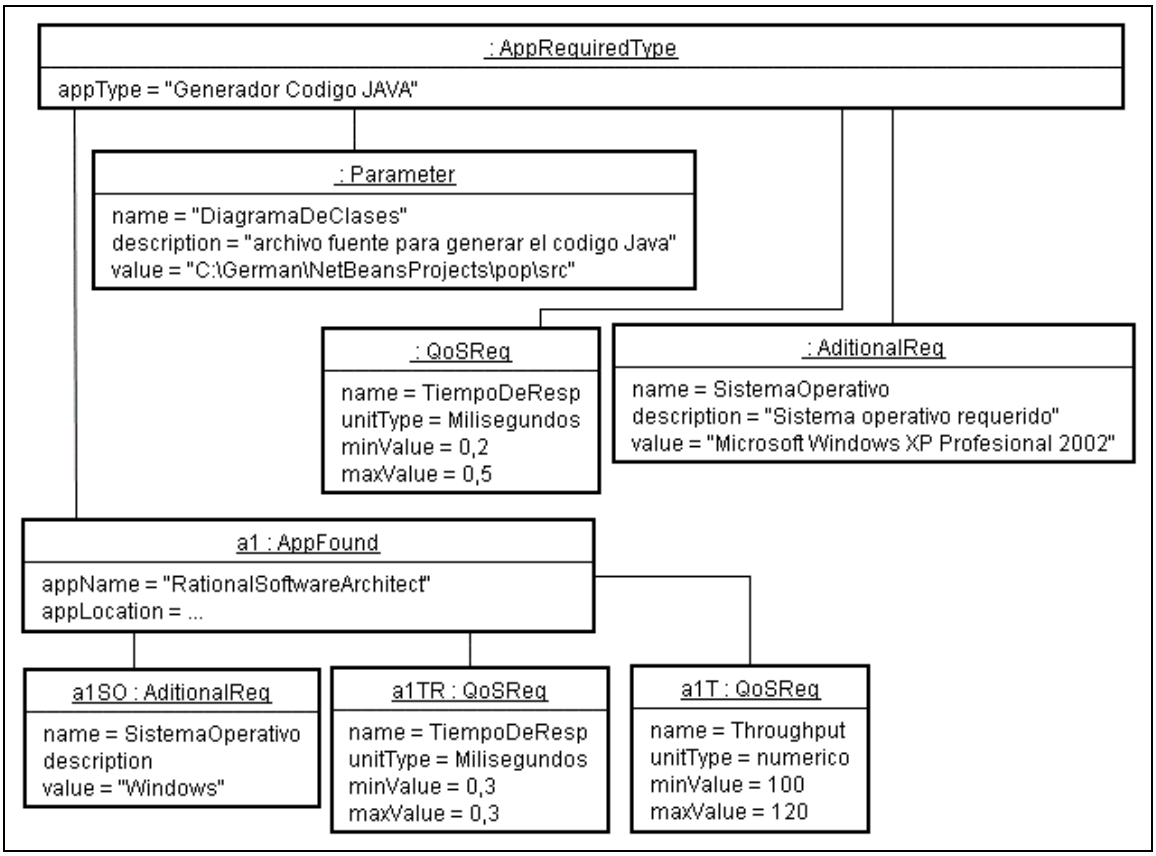


Figura 7.6. Grafo de la poscondición del proveedor1

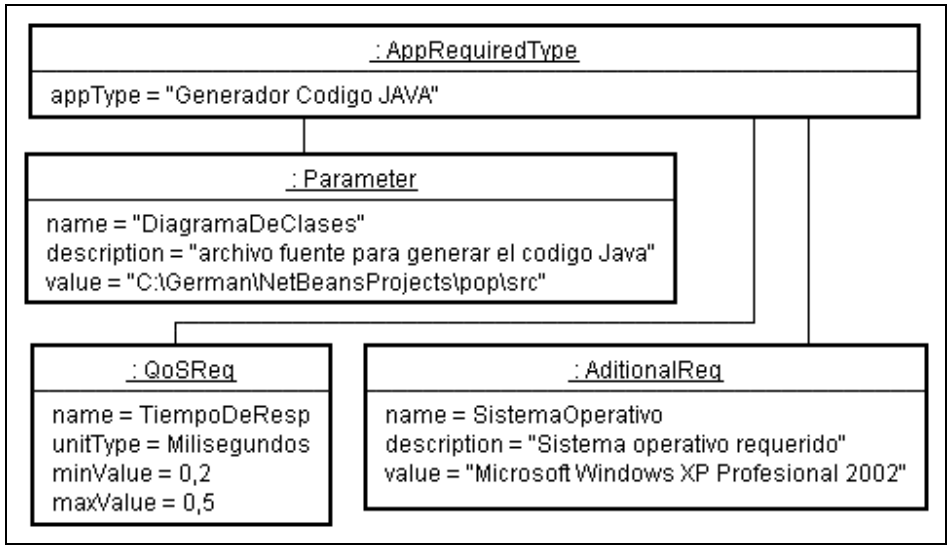


Figura 7.7. Grafo de la precondición del proveedor2

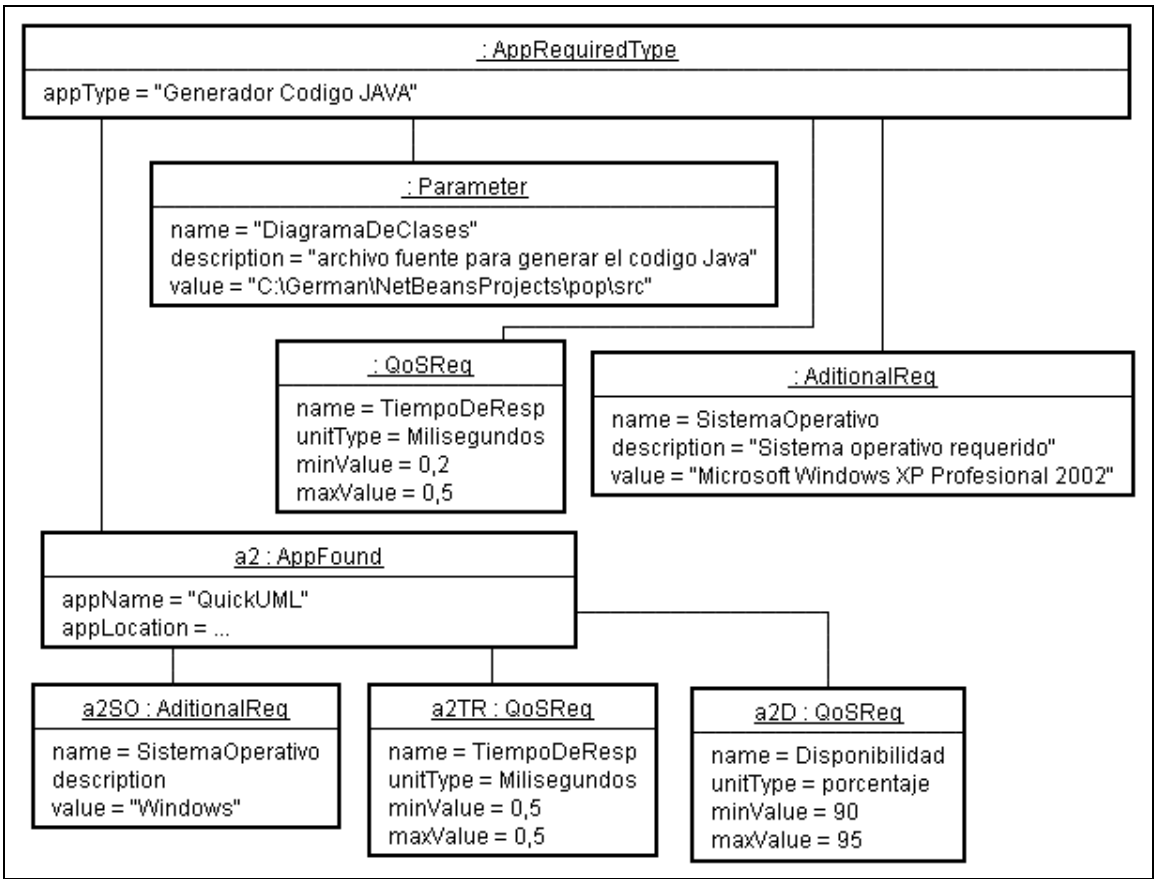


Figura 7.8. Grafo de la poscondición del proveedor2

7.3.1.2. Correspondencia entre los grafos

El motor del SGFT debe realizar la correspondencia entre los grafos del servicio web requerido (una aplicación para generar código Java a partir de un modelo de diseño representado con un diagrama de clases UML) y los grafos de los servicios web disponibles, para obtener los servicios web candidatos, que en este caso son dos y corresponden a los proveedores 1 y 2.

En la Figura 7.9 se puede observar que la precondición del proveedor 1 es un subgrafo de la precondición del solicitante, y que la poscondición del solicitante es un subgrafo de la poscondición del proveedor 1.

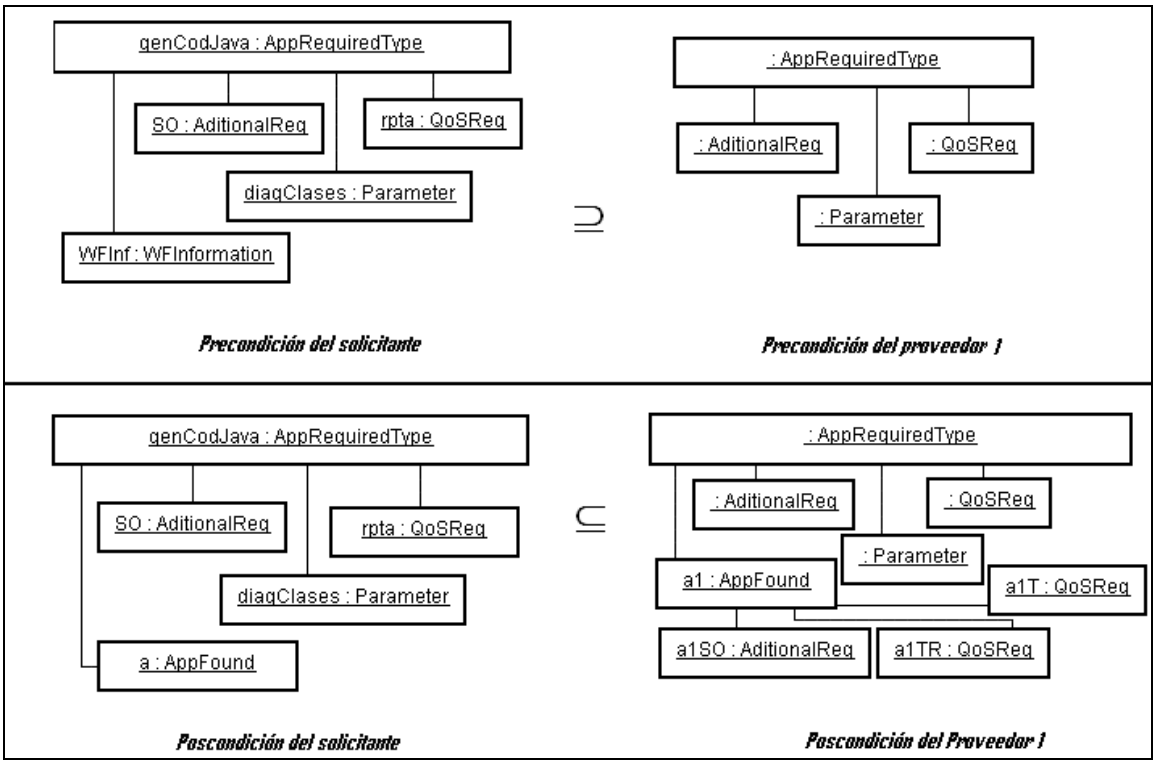


Figura 7.9. Grafos de la pre y poscondición del solicitante y el proveedor 1

Análogamente, en la Figura 7.10 se puede observar que la precondición del proveedor 2 también es un subgrafo de la precondición del solicitante, y que la poscondición del solicitante también es un subgrafo de la poscondición del proveedor 2. Por lo tanto, ambos proveedores ofrecen servicios web que son compatibles con los requerimientos del solicitante. Aunque sus especificaciones no son iguales, ya que el proveedor 1 especifica la una característica de calidad adicional, el throughput, mientras que el proveedor 2 especifica la disponibilidad.

Para que el motor del SGFT finalmente determine cuál es la aplicación que invocará resta analizar la historia de las ejecuciones previas y las características de calidad del propio motor del SGFT. La información adicional especificada por ambos proveedores (throughput y disponibilidad) se confronta con las características de calidad del motor del SGFT y sirven como otro elemento para decidir cuál servicio web es el más adecuado.

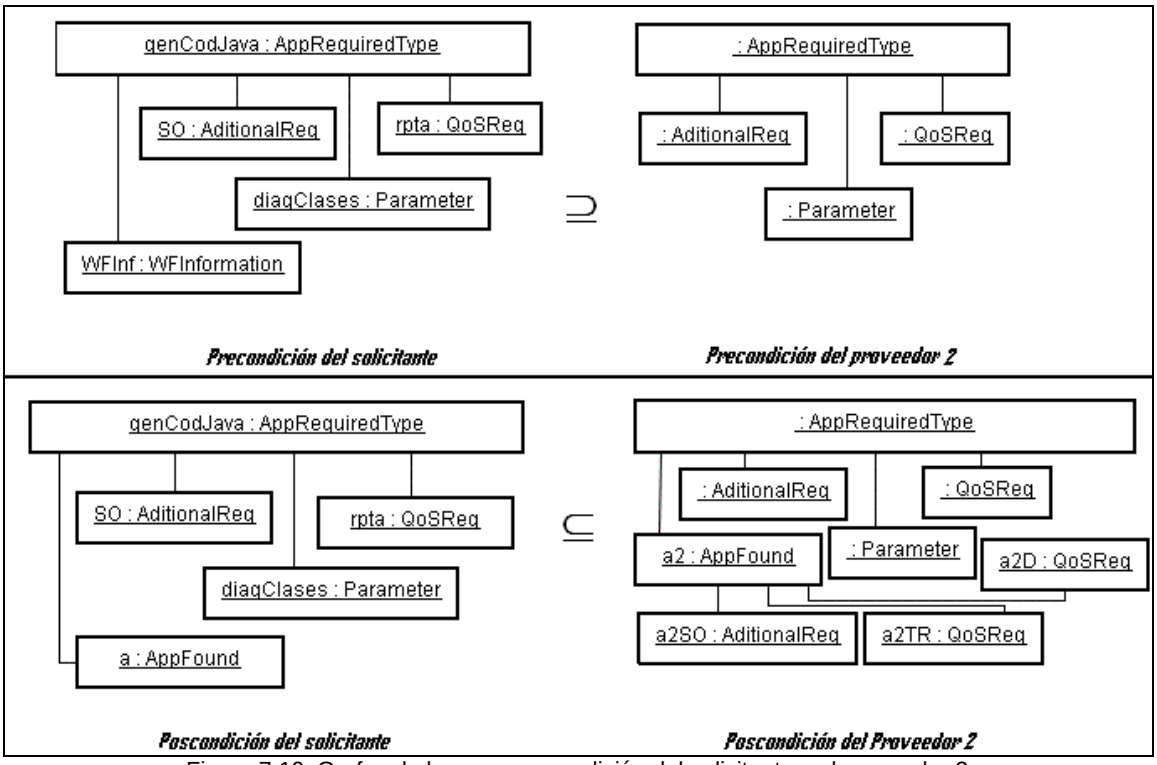


Figura 7.10. Grafos de la pre y poscondición del solicitante y el proveedor 2

Si finalmente siguen quedando las dos aplicaciones como candidatas existen dos opciones: requerir la interacción con el usuario que realizó la solicitud para que éste decida, o definir un criterio de selección adicional (por ejemplo, la primera de la lista, la que requiera menos recursos de hardware, etc.) y devolver la aplicación que lo cumpla. Esto otorga mayor flexibilidad al motor del SGFT, a la hora de invocar aplicaciones externas al sistema de administración de del SGFT de la organización, y favorece la distribución de las aplicaciones en la red.

7.3.2. Planificación del Proyecto y de sus iteraciones

En OpenUP/Basic, la disciplina Gestión del Proyecto plantea entre sus tareas principales la planificación del proyecto y la planificación de las iteraciones, dando lugar a los productos de trabajo Plan de Proyecto y Plan de Iteración, respectivamente.

El plan de proyecto reúne toda la información necesaria para gestionar el proyecto, y su propósito es que cualquier miembro del equipo pueda encontrar en este documento la información sobre cómo se administrará el proyecto. Define los parámetros para el

seguimiento del avance y los objetivos de alto nivel de las iteraciones y sus hitos. Además, define cómo esta organizado el proyecto y qué funciones se asignan a cada miembro del equipo. El director del proyecto es el responsable de desarrollar el plan del proyecto, trabajando muy estrechamente con el resto del equipo. El plan del proyecto permite a los miembros del equipo y demás interesados tener un panorama general del proyecto y saber cuándo se espera alcanzar un nivel de funcionalidad.

El nivel de detalle y formalidad de este documento dependerá de las características y necesidades específicas del proyecto de que se trate. Un documento más complejo podría basarse en diagramas de Gantt o Pert utilizados para realizar la planificación.

El objetivo de la planificación del proyecto de software es proporcionar un marco de trabajo que permita al director realizar estimaciones razonables de recursos, costos, y planificación temporal. Estas estimaciones se hacen en un tiempo limitado al comienzo del proyecto, y deben actualizarse regularmente a medida que progresa el mismo.

El modelo de estimación COCOMO utiliza fórmulas derivadas empíricamente para predecir esfuerzo, costos y duración como una función de las líneas estimadas de código y los puntos de función (medida indirecta orientada a la funcionalidad del sofware) [Pressman01]. La jerarquía de modelos de Boehm está constituida por tres modelos: básico, intermedio y avanzado. Estos modelos se definen para tres tipos de proyectos: orgánico (proyectos pequeños), semi-acoplado (proyectos intermedios en tamaño y complejidad) y empotrado (proyectos muy complejos).

Una organización dedicada al desarrollo de software que utilice OpenUP/Basic como proceso de desarrollo necesitará realizar la planificación de sus proyectos. Para ello deberá estimar la duración total del proyecto, los costos y los esfuerzos. La organización podría realizar estas estimaciones en base al modelo COCOMO antes descripto, para lo cual existen muchas herramientas libres disponibles en la web que implementan este modelo.

Para realizar esta actividad el motor del SGFT podría invocar una aplicación externa que se encargue de estimar, a partir de la información provista por la organización sobre el proyecto en cuestión. La estimación se podría realizar con cualquier aplicación disponible en la red, de manera de independizarse de la ubicación específica de dicha aplicación. El motor del SGFT, de acuerdo a los requerimientos recibidos, se encarga de realizar la correspondencia con los servicios web disponibles que satisfacen dichos requerimientos. Para ello, deberá transformar la formulación del requerimiento en grafos con atributos, para luego compararlos con los grafos que describen los servicios web ofrecidos por el proveedor.

7.3.2.1. Grafos de los servicios web del solicitante y los proveedores

Los grafos de las Figuras 7.11 y 7.12 representan instancias (diagrama de objetos) de los diagramas de clases que modelan la pre y poscondición del servicio web que representa la aplicación requerida por la solicitante. En la Figura 7.11, los objetos p1, p2, p3 y p4 de tipo Parameter, almacenan los parámetros necesarios para realizar la estimación, es decir, la información necesaria para poder invocar la aplicación.

El objeto p1 almacena la cantidad estimada de líneas de código del software. El objeto p2 almacena el monto establecido por desarrollador. Los objetos p3 y p4 se refieren al modelo de COCOMO elegido y el tipo de proyecto, en este caso el básico y orgánico. El objeto rpta:QoSReq especifica que el solicitante de la aplicación requiere que el tiempo de respuesta esté entre los valores 0,5 y 0,7 y el objeto SO:AditionalReq indica que el sistema operativo requerido es Windows XP.

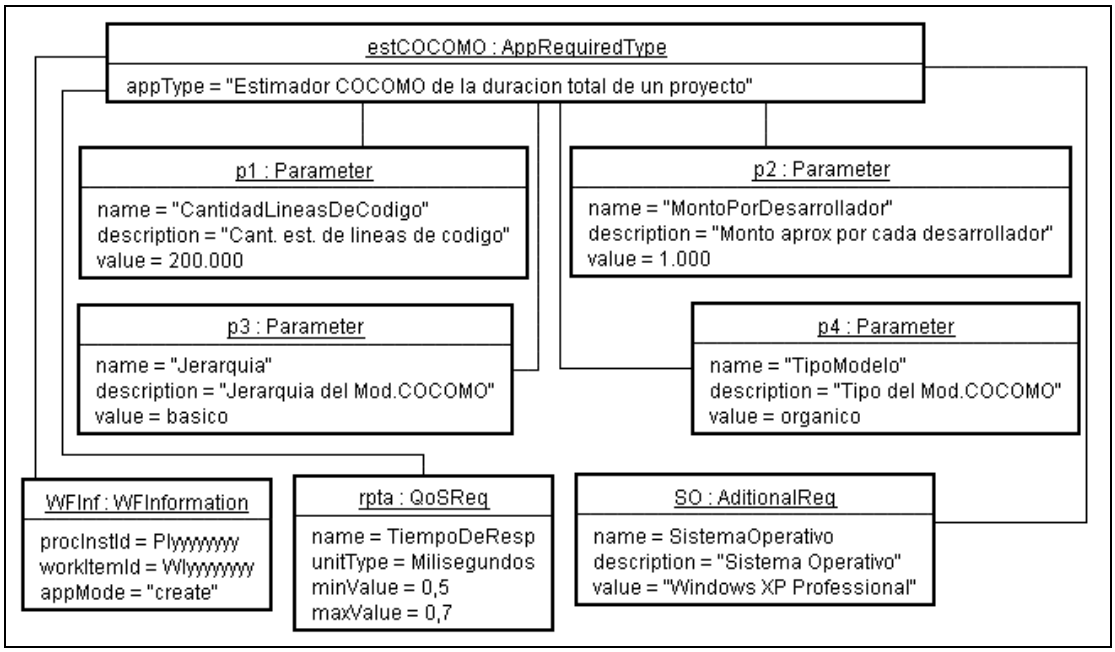


Figura 7.11. Grafo de la precondición del solicitante

En la Figura 7.12, el objeto a:AppFound, no especifica valor para sus atributos, ya que el solicitante al inicio no conoce los servicios que cumplirán sus requerimientos.

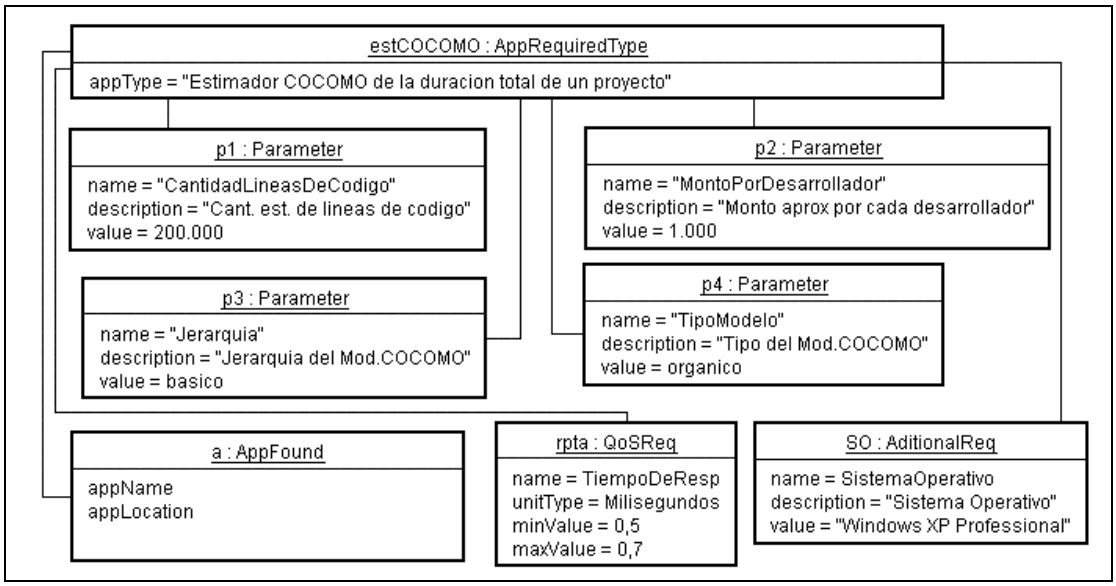


Figura 7.12. Grafo de la poscondición del solicitante

Los grafos de las Figuras 7.13, y 7.14 instancian los diagramas de clases que modelan la pre y poscondición del proveedor que ofrece un servicio, que permite invocar una aplicación de estimación según el modelo COCOMO. En la Figura 7.13, los objetos :QoSReq y :AditionalReq almacenan la información del tiempo de respuesta y el sistema operativo del servicio web ofrecido por el proveedor.

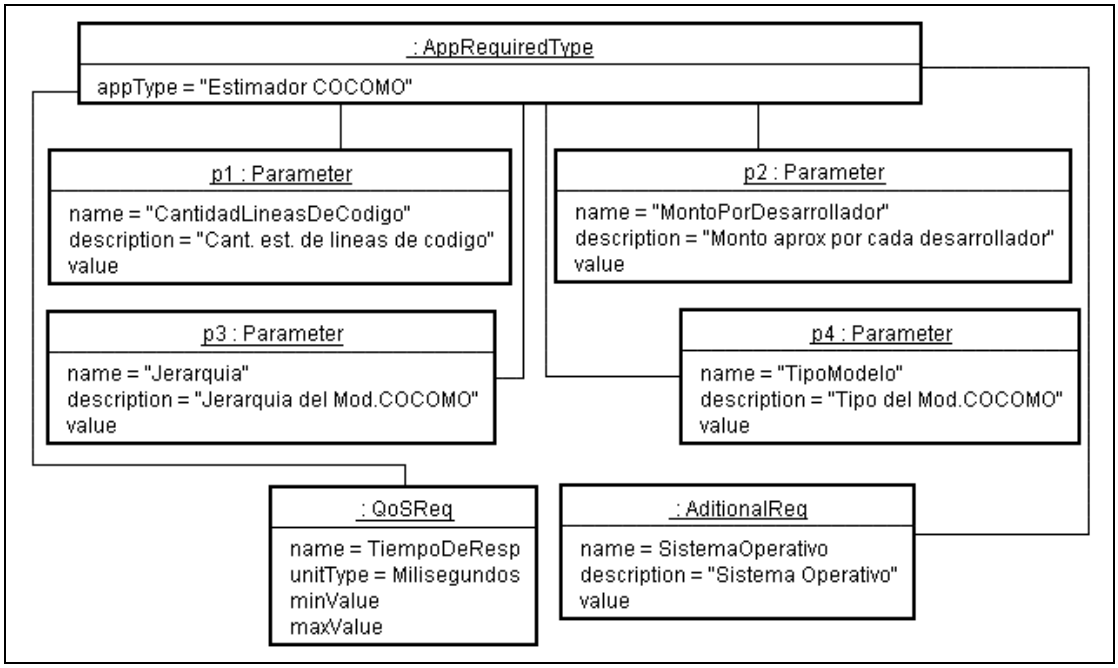


Figura 7.13. Grafo de la precondición del proveedor

Por su parte, en la Figura 7.14, los objetos :QoSReq y :AditionalReq almacenan la información del proveedor sobre el tiempo de respuesta y el sistema operativo de su servicio web ofrecido. El objeto :QoSReq indica que el tiempo de respuesta para invocar la aplicación ProyectoCocomoApplication es de 0,7 milisegundos.

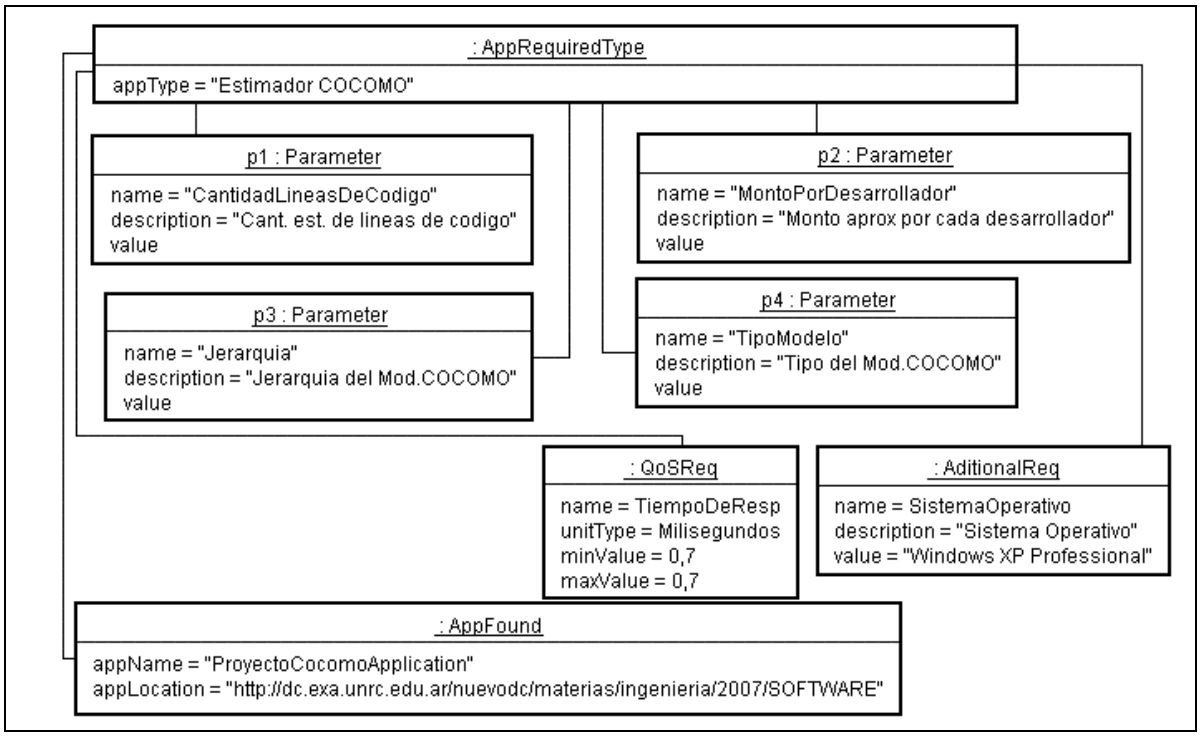


Figura 7.14. Grafo de la poscondición del proveedor

7.3.2.2. Correspondencia entre los grafos

Nuevamente resta analizar si el servicio web ofrecido por el proveedor satisface la demanda del solicitante, comparando los subgrafos generados. En los dos grafos superiores de la Figura 7.15 se observan las precondiciones del solicitante y del proveedor. La del proveedor contiene siete objetos, :AppRequiredType, p1, p2, p3 y p4: Parameter, :AditionalReq y :QoSReq. Por su parte, la precondición del solicitante, posee estos mismos siete objetos, más el objeto :WFInformation, el cual almacena información que sólo es importante para el motor del SGFT. Claramente, la precondición del proveedor contiene todos los objetos de la precondición del solicitante, por lo tanto, es un subgrafo.

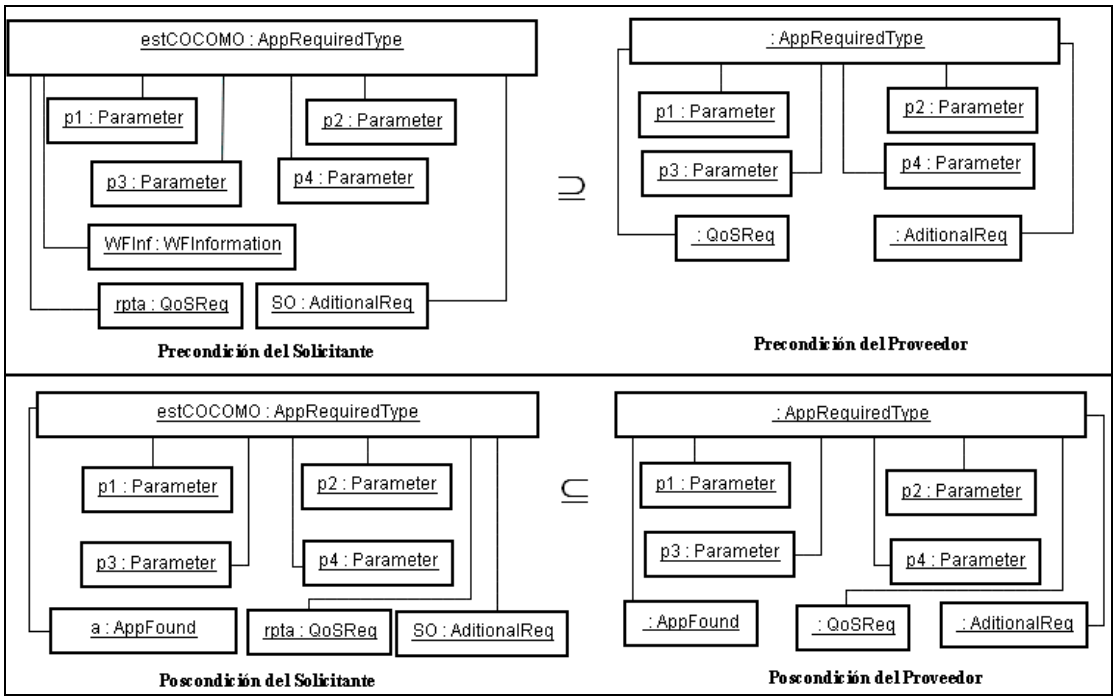


Figura 7.15. Grafos de la pre y poscondición del solicitante y el proveedor

Por otro lado, en los dos grafos inferiores de la Figura 7.15, se observa que la poscondición del solicitante contiene todos los objetos especificados en la poscondición del proveedor (AppRequiredType, Parameter, AditionalReq, QoSReq, AppFound). Entonces, también se cumple que la poscondición del solicitante contiene todos los objetos de la poscondición del proveedor, por lo tanto, es un subgrafo.

Como se cumple con el morfismo establecido, entonces es posible asegurar que el proveedor cumple los requerimientos del solicitante, tanto los funcionales como los no funcionales. Así, la aplicación ofrecida por el proveedor es una aplicación adecuada para estimar la duración total del proyecto, costos y esfuerzos, y puede utilizarse durante la planificación del mismo.

Otros elementos que tiene el motor del SGFT para determinar cuál es la aplicación que invocará, en el caso de haber más de una que satisfaga los requerimientos, es la historia de las ejecuciones previas y las características de calidad del propio motor. La información adicional especificada por los proveedores, como por ejemplo otras características de calidad que en principio no son requeridas por el solicitante, se confronta con las características de calidad del motor y sirven como otro elemento para decidir cuál servicio web es el más adecuado.

CAPÍTULO 8.

Conclusiones y Trabajos Futuros

En este capítulo se presentan las conclusiones de la Tesis. En la sección 8.1 se resumen las principales contribuciones a los Sistemas de Gestión de Flujos de Trabajo (SGFT), según los objetivos inicialmente planteados. En la sección 8.2 se plantean las líneas de investigación futuras y, finalmente, en la sección 8.3 se muestran las publicaciones derivadas de la Tesis.

8.1. Conclusiones

La automatización de los procesos de negocio o flujos de trabajo ha producido en las organizaciones una importante reducción de costos, incremento de eficiencia, menores oportunidades de cometer errores, y mejor control y calidad en beneficio de todos. En la actualidad la automatización de los procesos de negocio se ha convertido un objetivo primordial para el crecimiento y desarrollo de las organizaciones.

En esta Tesis se han elaborado las siguientes propuestas, expuestas en detalle en los capítulos previos:

- El uso de servicios web como parte de la Interfaz de las Aplicaciones Invocadas de los SGFT,
- la incorporación de las características de calidad para describir los aspectos no funcionales de las invocaciones,
- el empleo de técnicas de transformación de grafos para especificar formalmente la funcionalidad solicitada,
- la aplicación de la propuesta a un caso de estudio concreto.

Con respecto al uso de servicios web en la Interfaz de las Aplicaciones Invocadas, el principal beneficio en este sentido es la posibilidad de tener las aplicaciones externas al SGFT distribuidas en la red y que ello resulte transparente al motor del SGFT. Así, la elección de dichas aplicaciones se realiza de manera dinámica, teniendo lugar durante la ejecución propia del flujo de trabajo. Esta distribución transparente permite al motor del SGFT invocar las aplicaciones externas sin la necesidad de conocer su ubicación exacta, con el importante beneficio de que las mismas pueden cambiar su ubicación en la red sin que esto implique ninguna modificación en su invocación.

Por otra parte, debido a que los ambientes de los SGFT actuales son heterogéneos, ampliamente distribuidos, y dada la creciente necesidad de integración entre las organizaciones, existe un mayor requerimiento de interoperabilidad, es decir que los distintos SGFT puedan cooperar intercambiando datos y sincronizando las ejecuciones de sus aplicaciones. Está interoperabilidad involucra la operación y desarrollo de procesos de negocio que se ejecutan a través y entre organizaciones, donde los SGFT son el soporte tecnológico obvio para lograrlo. Otra contribución importante de esta Tesis es la optimización de la comunicación entre los distintos SGFT y las aplicaciones con que deben interactuar, permitiendo compartir recursos distribuidos a gran escala e integrarse, y manejar sus procesos de una forma más flexible y eficiente. El uso de servicios web plantea una forma unificada de acceder a las aplicaciones, con la posibilidad de acceder a servicios y/o métodos comunes y favorecer la interoperabilidad a través de las diferentes plataformas. Los servicios web poseen numerosas ventajas tendientes a favorecer la distribución de las aplicaciones y la interoperabilidad de los sistemas: permiten la utilización de aplicaciones de componentes abiertas y auto descriptas, la composición rápida de aplicaciones distribuidas, el uso de aplicaciones modulares y desacopladas, la independencia de la plataforma subyacente y el lenguaje de programación, el

acoplamiento débil, etc., por lo que se convierten en un excelente recurso para permitir la distribución transparente de las aplicaciones externas al flujo de trabajo.

Con respecto a la incorporación de las características de calidad de los servicios web, actualmente, un cliente que desea acceder a un servicio no está interesado solamente en los aspectos funcionales del servicio, sino también en sus características de calidad, esto es, el conjunto de atributos no funcionales que pueden tener un alto impacto en la calidad del servicio ofrecido por un proveedor. Por ello su incorporación es vital cuando se desea seleccionar el servicio web más adecuado que satisfaga todos los requerimientos del usuario.

Además, como existen múltiples servicios web en el registro UDDI que proveen similares funcionalidades, las características de calidad se pueden utilizar para afinar la búsqueda. Con su incorporación a la descripción de los servicios web se hace posible evaluar la calidad de los mismos, filtrar la lista de todos los servicios descubiertos y obtener los más adecuados, con la ventaja de que esta incorporación no requiere modificar el modelo de servicio web actual definido por el protocolo UDDI.

Asimismo, el hecho de especificar las características de calidad en el mismo lenguaje WSDL en el que se describen los aspectos funcionales del servicio otorga un importante beneficio a la hora de realizar la correspondencia entre los servicios disponibles en la web y los solicitados por los usuarios, ya que esta se puede realizar en un mismo paso y con un mismo criterio de contrastación.

Otro beneficio importante está dado por la utilización de las técnicas de transformación de grafos, lo cual permite que los resultados de las consultas en el protocolo UDDI no estén limitados por las palabras clave definidas en tiempo de diseño, favoreciendo la selección dinámica de los servicios web.

La técnica de gramáticas de grafos posee una sólida base matemática, y una sintaxis gramatical que facilita la especificación visual y operativa de los servicios web. Además, existen algoritmos y herramientas que permiten su aplicación. Su representación gráfica facilita la especificación y comprensión de las correspondencias entre servicios, de forma intuitiva. Además, permite lograr una especificación semántica más precisa del servicio web, que ayuda a establecer la correspondencia entre los requerimientos del usuario y los servicios web disponibles. La formalización de la especificación como un grafo con atributos, permite realizar una correspondencia con los servicios web disponibles en la red que cumplen con el morfismo establecido, otorgando mayor flexibilidad al motor del SGFT a la hora de seleccionar un servicio para invocar una aplicación. Además, facilita la

incorporación de servicios nuevos, y la invocación no está limitada a una aplicación específica, definida en tiempo de desarrollo.

Finalmente, en este trabajo se han presentado dos casos de estudio que ejemplifican la propuesta, ambos basados en el proceso de desarrollo de software OpenUP/Basic. Los ejemplos muestran, en una situación concreta, el comportamiento del motor del SGFT cuando necesita invocar una aplicación externa, de la cual desconoce su ubicación. En los mismos se puede observar los grafos instancia que surgen de los requerimientos del usuario del SGFT y los servicios web disponibles, y el morfismo que se establece entre dichos grafos. También se puede observar cómo actúa el motor del SGFT cuando encuentra más de un servicio web que satisface los requerimientos del usuario, valiéndose de las características de calidad adicionales de los mismos.

8.2. Trabajos Futuros

La propuesta planteada en esta Tesis abre nuevas líneas de trabajo, las cuales se detallan en las siguientes secciones.

8.2.1. Continuación del trabajo realizado

En lo que respecta a la continuación y consolidación del trabajo realizado, resulta necesario afianzar la descripción de los requisitos no funcionales del servicio web, en cuanto a la especificación WSDL y los elementos definidos en dicha especificación, para asegurar que dicha descripción abarque todas las posibilidades de acuerdo a las diferentes características de calidad que se pueden especificar. Para ello es imperioso estudiar más ejemplos de situaciones reales, y analizar si las mismas se pueden representar según la especificación propuesta.

También es necesario reforzar el proceso de contrastación de los servicios web, con técnicas de transformación de grafos. Se debe analizar con más detalle cuál es la información que necesita registrar el motor del SGFT en cuanto a las características de calidad mínimas requeridas por el propio motor para aceptar un servicio ofrecido en la web y la historia de las invocaciones previas, principalmente, cómo hace el motor para alimentar esta información, asegurándose de validar correctamente lo publicado por los proveedores sobre las características de calidad de sus servicios.

8.2.2. Extensiones al trabajo realizado

En cuanto a las posibles extensiones al trabajo presentado, sería interesente el estudio de la extensión a otras interfaces del Modelo de Referencia de Workflow de la WfMC, en particular, a la Interfaz de las Aplicaciones de Clientes (Interfaz 2). Esta Interfaz es la encargada de manejar la interacción entre el motor del SGFT y el Administrador de la Lista de Trabajo, responsable de organizar el trabajo relacionado a un recurso de usuario y controlar la interacción con los usuarios. El motor deposita en la Lista de Trabajo los ítems ha ser ejecutados por cada usuario y el Administrador de la Lista de Trabajo maneja la interacción entre los participantes del SGFT y el Servicio de Ejecución de Flujo de Trabajo. Este administrador soporta un amplio rango de interacción con las aplicaciones de clientes. Para la invocación de dichas aplicaciones, resulta interesente plantear la optimización con servicios web y reglas de transformación de grafos, con el objetivo de otorgar mayor flexibilidad en tiempo de ejecución, permitiendo la distribución de las aplicaciones en la red y haciendo transparente para el motor del SGFT la ubicación de las mismas.

8.3. Publicaciones derivadas de la Tesis

En esta sección se presentan las publicaciones realizadas en el marco de esta Tesis. Las mismas están clasificadas en congresos internacionales y nacionales. También se incluyen los trabajos finales de carreras dirigidos o co-dirigidos, directamente relacionados a esta Tesis.

8.3.1. Congresos Internacionales

- SADIO Electronic Journal of Informatics and Operations Research - EJS 2009 - Marcela Daniele, Paola Martellotto, Daniel Riesco: “Mejorando la Invocación de Aplicaciones desde un Workflow: Un Caso de Estudio”, – JAIIO 2009, 10th Argentine Symposium on Software Engineering – ASSE 2009.

8.3.2. Congresos Internacionales

- Marcela Daniele, Paola Martellotto, Daniel Riesco: *"Mejorando la Invocación de Aplicaciones desde un Workflow: Un Caso de Estudio"*, XXXV Conferencia Latinoamericana de Informática - XXXV CLEI, Universidad Federal de Pelotas – Universidad Católica de Pelotas, Pelotas – Brasil, 22 al 25 de Septiembre de 2009.

- Narayan Debnath, Paola Martellotto, Marcela Daniele, Daniel Riesco, Germán Montejano: *"Using Web Services to optimize communication among the Workflow Engine and its Client Application Interface"*, Second International Conference on Information Systems, Technology and Management – ICISTM 2008 – Ghaziabad, Institute of Management Technology, Dubai – Emiratos Árabes Unidos, Marzo de 2008.

8.3.3. Congresos Nacionales

- Paola Martellotto, Marcela Daniele, Daniel Riesco: *"Invocación de Aplicaciones desde el Workflow con Servicios Web considerando sus Requerimientos No Funcionales"*, 38th Argentine Conference on Informatics – JAIIO 2009, 10th Argentine Symposium on Software Engineering – ASSE 2009, Mar del Plata – Argentina, pp. 160-165, ISSN 1850-2792, 24 al 28 de Agosto de 2009.

- Daniel Riesco, Paola Martellotto, Marcela Daniele: "Implementación de la comunicación entre el Workflow y las Aplicaciones Externas con Servicios Web y Reglas de Transformación de Grafos", 14mo. Congreso Argentino de Ciencias de la Computación – CACIC 2008, 5to. Workshop de Ingeniería de Software y Bases de Datos – WISBD, Universidad Nacional de Chilecito, La Rioja – Argentina, Páginas 550 pp, ISBN 978-987-24611-0-2, Octubre de 2008.

- Paola Martellotto, Marcela Daniele, Daniel Riesco: *"Selección de aplicaciones invocadas por el motor Workflow basada en Servicios Web y reglas de Transformación de Grafos"*, 10mo Workshop de Investigadores en Ciencias de la Computación – WICC 2008, Universidad Nacional de La Pampa – Facultad de Ingeniería, General Pico – La Pampa – Argentina, ISBN 978-950-863-101-5, Mayo de 2008,

- Paola Martellotto, Marcela Daniele: "Optimizando la comunicación entre las Aplicaciones de Clientes y el motor del Modelo de Referencia de Workflow con Servicios Web", 13mo. Congreso Argentino de Ciencias de la Computación – CACIC 2007, 4er. Workshop de Ingeniería de Software y Bases de Datos – WISBD, Universidad Nacional del Nordeste - Facultad de Ciencias Exactas y Naturales y Agrimensura, Universidad Tecnológica Nacional - Facultad Regional Resistencia Corrientes y Resistencia – Argentina, Octubre de 2007.

- Paola Martellotto, Marcela Daniele, Daniel Riesco: *"Especificación de la Interfaz de Aplicaciones de Cliente del Modelo de Referencia de Workflow utilizando Web Services"*, 9mo Workshop de Investigadores en Ciencias de la Computación – WICC 2007 Universidad Nacional de la Patagonia San Juan Bosco, Trelew - Chubut – Argentina, Páginas 370-374, ISBN 978-950-763-075-0, Mayo de 2007.

8.3.4. Trabajos Finales de Carreras

Los siguientes trabajos finales se realizan en el marco del presente trabajo, en el Departamento de Computación de la Facultad de Ciencias Exactas, Fco-Qcas y Naturales de la Universidad Nacional de Río Cuarto:

- Pablo Gil, Mauro Garat: "Desarrollo de una herramienta que permita la comunicación del workflow y las aplicaciones de clientes y externas con servicios web", Trabajo Final de la carrera Licenciatura en Ciencias de la Computación, Departamento de Computación, Facultad de Ciencias Exactas, Fco-Qcas y Naturales, Universidad Nacional de Río Cuarto.

- Sergio Gadpen: "Desarrollo de una herramienta que permita la selección automática de servicios web con reglas de transformación de grafos, incorporando características de calidad (QoS)", Trabajo Final de la carrera Licenciatura en Ciencias de la Computación, Departamento de Computación, Facultad de Ciencias Exactas, Fco-Qcas y Naturales, Universidad Nacional de Río Cuarto.

Referencias Bibliográficas

[ACKM04] Alonso, G., Casati, F., Kuno, H., Machiraju, V., Web Services. Concepts, Architectures and Applications. Springer-Verlag. ISBN: 3-540-44008-9. 2004.

[Allen01] Allen R., Workflow: An Introduction. Extracted from the Workflow Handbook, 2002.United Kingdom Chair, WfMC External Relations Committee.

[Balduino07] Balduino R. Introduction to OpenUP (Open Unified Process). http://www.eclipse.org/epf/general/OpenUP.pdf. 2007. Último acceso Marzo de 2009.

[Baresi97] Baresi, L. Formal Customization of Graphical Notations. PhD thesis, Dipartimento di Elettronica e Informazione – Politecnico di Milano, Italian. 1997.

[BH02] Baresi L., Heckel R. Tutorial Introduction to Graph Transformation: A Software Engineering Perspective. First International Conference on Graph Transformation. Spain. v. 2505, pp. 402-429. (ICGT 2002). 2002.

[BRJ05] Booch G., Rumbaugh J., Jacobson I., The Unified Modeling Language. Addison Wesley, Second Edition. 2005.

[BSSB08] Braun, I ., Strunk, A , Stoyanova, G ., Buder, B., ConQo – A Context-And QoS-Aware Service Discovery. Freiburg. IADIS. In proceedings of www/Internet. October 2008.

[CHH04] Cherchago A., Heckel R., Specification Matching of Web Services Using Conditional Graph Transformation Rules, In G. Engels, H. Ehrig, F. Parisi-Presicce, and G. Rozenberg (Editors): Proc. 2nd International Conference on Graph Transformation (ICGT 04), Roma, Italy, Volume 3256 of Lecture Notes in Computer Science. Springer-Verlag, 2004.

[CS05] Cardozo J., Shelt A., Introduction to Semantic Web Services and Web Process Composition. In Semantic Web Process: powering next generation of processes with Semantics and Web Services, Lecture Notes in Computer Science, Springer, 2005.

[D'Ambrogio06] D'Ambrogio, A.: A Model-driven WSDL Extension for Describing the QoS of Web Services. In IEEE proceedings of ICWS. Pages 789-796. ISBN:0-7695-2669-1. 2006.

[DBRG00] Dramis, L., Britos, P., Rossi, B. y García Martínez, R. Verificación de Bases de Conocimiento Basada en Algebra de Grafos. Revista del Instituto Tecnológico de Buenos Aires. Volumen 23. Páginas 74-86. ISSN: 0326-1840. 2000.

[EEPT06] Ehrig, H., Ehrig, K., Prange, U., Taentzer, G. Fundamentals of Algebraic Graph Transformation. Chapter 1. General Introduction. Monographs in Theoretical Computer Science. An EATCS Series, Springer-Verlag, New York, Inc., XIV, 388 p. 41 illus., Hardcover, ISBN: 978-3-540-31187-4, 2006.

[EPF] Eclipse Process Framework (EPF) Community. http://www.eclipse.org/epf/

[Gramaje02] Gramaje Penadés C., Una Aproximación Metodológica al Desarrollo de Flujos de Trabajo. Tesis de Doctor en Informática. Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia. Enero de 2002.

[HCH04] Heckel R., Cherchago A., Application of Graph Transformation for Automating Web Service Discovery, Proc. Dagstuhl Seminar 04101 Language Engineering for Model-Driven Software Development, Dagstuhl, Germany, 2004.

[Heckel06] Heckel, R. Graph Transformation in a Nutshell. Electronic Notes in Theoretical Computer Science. 148, pp. 187-198. Elsevier 2006.

[HHL03] Hausmann J.H., Heckel R., Lohmann M., Towards Automatic Selection of Web Services Using Graph Transformation Rules. Berliner XML Tage. XML-Clearinghouse. 286-291, 2003.

[HHL04] Hausmann J.H., Heckel R., Lohmann M., Model-based Discovery of Web Services, Proceedings of the IEEE International Conference on Web Services, 2004.

[HHL05] Hausmann J.H., Heckel R., Lohmann M., Model-based development of Web services descriptions enabling a precise matching concept. International Journal of Web Service Research. Volume. 2. Issue: 2. p. 67 – 84. 2005.

[HM05] Heckel R., Mariani L., Automatic Conformance Testing of Web Services. Fundamental approaches to software engineering. International conference No8, Edinburgh , ROYAUME-UNI. 1973, vol. 3442, pp. 34-48, XIII, 371 p., ISBN 3-540-25420-X. 2005.

[HSD08] Al Hunaity, M., El Sheikh, A., Dudin, B., A New Web Service Discovery Model Based On QoS. In IEEE proceedings of ICTTA08. Arab Academy for Banking and Financial Sciences, JORDAN, ISBN: 978-1-4244-1751-3. April 2008.

[JBR99] Jacobson I., Booch G., Rumbaugh J. The Unified Software Development Process. Addison Wesley, 1999.

[Kaya01] Kaya A., Workflow Interoperability: The WfMC Reference Model and an implementation. Master thesis. Technical University Hamburg-Harburg Germany. Abril de 2001.

[Kokash05] Kokash N.: Web Service Discovery with Implicit QoS Filtering. In proceedings of the IBM PhD Student Symposium, colocated with the "International Conference on Service-Oriented Computing (ICSOC)". P 61-66, Amsterdam, December 2005.

[Labra06] Labra Gayo, J.E., Introducción a los servicios web. http://www.di.uniovi.es/~labra/cursos/Web20/ServiciosWeb.pdf, Octubre

de 2006.

[Lara04] Lara, J. Transformación de Modelos. Transparencias del Curso de Postgrado: Diseño de Software basado en Modelado y Simulación. Universidad Autónoma de Madrid. Febrero de 2004.

[LH03] Laukkanen M., Helin H.: Composing Workflows of Semantic Web Services. Workshop on Servicios Web and Agent-based Engineering. AAMAS'2003. Melbourne, Australia. 2003.

[MY05] Maciel L. A. H., Yano E.T.: Uma Linguagem de WF Para Composicao de Web Services. XIX Simpósio Brasileiro de Engenharia de Software. Uberlandia, MG, Brasil. 2005.

[OUPB] OpenUP/Basic. http://epf.eclipse.org/wikis/openupsp/index.htm

[Papazoglou08] Papazoglou M., Web Services: Principles and Technology. Ed. Pearson Education Limited, Prentice Hall, First Edition. 2008.

[Plesums02] Plesums Ch., An Introduction to Workflow. Extracted from the Workflow Handbook, 2002.Computer Sciences Corporation, Financial Services Group. 2002.

[Pressman01] Pressman, Roger. Software Engineering: A Practitioner's Approach. Fifth edition. Ed. McGraw Hill. ISBN: 0071181822. 2001.

[Ran03] Ran, S., A Model for Web Services Discovery with QoS, ACM SIGecom Exchanges 4(1): pp. 1-10, 2003.

[RUP] Rational Unified Process. http://www-306.ibm.com/software/rational/

[Schmidt98] Schmidt M.T., Building Workflow Business Objects. IBM Software Group OOPSLA'98 Business Object Workshop IV.

[SOAP] World Wide Web Consortium. SOAP Version 1.2 Part 1: Messaging Framework. http://www.w3.org/TR/soap12-part1/. Último acceso Mayo 2009.

[UDDI] OASIS. Universal Description, Discovery an Integration of Web Services –UDDI. Versión 3.0.2. http://uddi.org/pubs/uddi_v3.htm. Último acceso Mayo 2009.

[UML] Object Management Group. UML Specification Version 1.4,

http://www.celigent.com/omg/umlrtf/, 2001.

[VHA06] Vu, L., Hauswirth, M., Aberer, K., Towards P2P-Based Semantic Web Service Discovery with QoS Support. In proceedings of 3rd Business Process Management Workshops, pages 18-31, ISBN 978-3-540-32595-6 2006.

[W3C03-QoS] W3C Working Group. QoS for Web Services: Requirements and Possible Approaches. http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/. Último acceso Julio de 2009. 2003.

[WfMC09] Workflow Management Coalition. A Common Object Model Discussion Paper. WfMC Document WfMC-TC10-22. http://www.newr.com/doc/wfmc/TC-1022_commom_Object%20Model_Paper.pdf. Último acceso Abril de 2009.

[WfMCa] Workflow Management Coalition. The Workflow Reference Model. WfMC-TC00-1003. http://www.wfmc.org/reference-model.html. Último acceso Mayo 2009.

[WfMCb] Workflow Management Coalition. Programming Interface 2&3 Specification. WfMC-TC-1009. V2.0. http://www.wfmc.org/index.php?option=com_docman&task=doc_details&Itemid=72&gid=47. Último acceso Mayo 2009.

[WSA] World Wide Web Consortium. Web Services Architecture. http://www.w3.org/TR/ws-arch/. Último acceso Mayo 2009.

[WSDL1.1] World Wide Web Consortium. Web Services Description Language (WSDL) Version 1.1, http://www.w3.org/TR/wsdl. Último acceso Mayo 2009.

[WSDL2.0-0] World Wide Web Consortium. Web Services Description Language (WSDL) Version 2.0 Part 0: Primer. http://www.w3.org/TR/wsdl20-primer. Último acceso Mayo 2009.

[WSDL2.0-1] World Wide Web Consortium. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. http://www.w3.org/TR/wsdl20. Último acceso Mayo 2009.

[XML] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Fifth Edition). http://www.w3.org/TR/xml. Último acceso Mayo 2009.

[YLX05] Yang M., Liang H., Xu B., S-WFMS: A service-based WFMS in Grid Environment. Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05). IEEE. 2005.

[ZHYJ07] Zhang, W., Huang, Ch., Yang, Y., Jia, X., QoS-driven Service Selection Optimization Model and Algorithms for Composite Web Services, in IEEE proceedings of COMPSAC'07. 2007.

[ZTP05] Zimmermann, O., Tomlinson, M., Peuser, S., Perspectives on Web Services. Applying SOAP, WSDL and UDDI to Real-World Projects. Springer-Verlag. ISBN: 10-3-540-00914-0. 2005.

ANEXO A.

Tecnologías de los Servicios Web

Este anexo detalla las tecnologías básicas de los servicios web. Se presenta el lenguaje utilizado para describir los servicios, denominado WSDL (Web Service Description Language), el cual se basa en el lenguaje de marcas extensible conocido como XML (Extensible Markup Language). También se describe el protocolo de comunicación SOAP (Simple Object Access Protocol) y el protocolo que permite registrar y acceder a los servicios publicados en la web. Se ilustran las especificaciones relacionadas a los servicios web a través de ejemplos simples.

La estructura del anexo es la siguiente. La sección 1 ilustra las tecnologías subyacentes descriptas brevemente en el capítulo 4. La sección 2 describe la organización de los documentos WSDL que describen los servicios web, con sus cuatro elementos principales. Y detalla las características principales del lenguaje de marcado XML. La sección 3 presenta el protocolo SOAP para la representación de los mensajes, y finalmente, la sección 4 el directorio UDDI utilizado para el registro y la localización de Servicios Web.

A.1 Introducción

La arquitectura de los servicios web involucra muchas capas y tecnologías interrelacionadas. Existen muchas formas de visualizar estas tecnologías, así como existen muchas maneras de crear y utilizar los servicios web. La Figura A.1 ilustra esta familia de tecnologías.

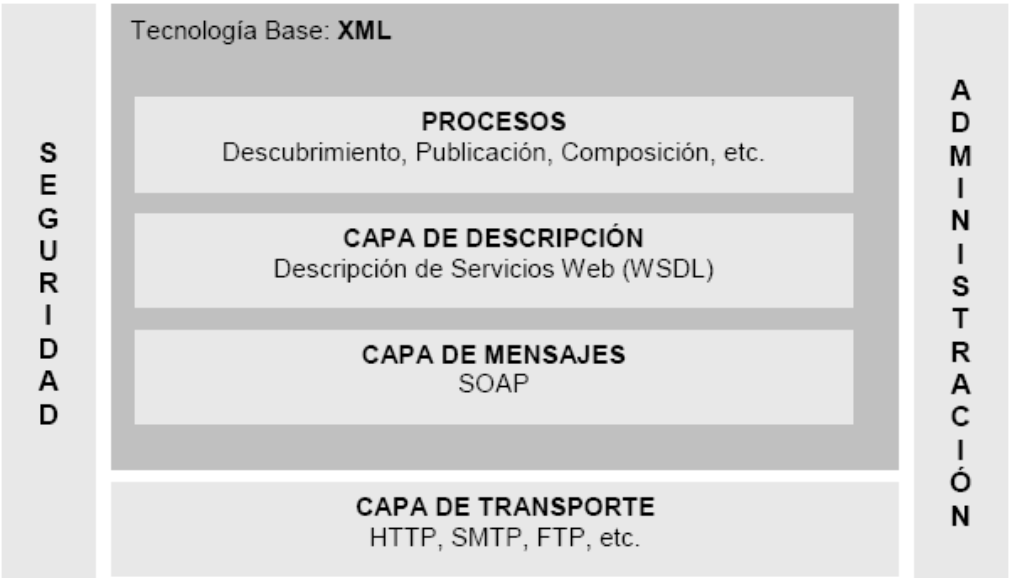


Figura A.1. Tecnologías de los Servicios Web

A continuación se describen las tecnologías que son vistas como críticas en cuanto al rol que las mismas cumplen en la arquitectura.

A.2 El Protocolo de Transferencia HTTP

El protocolo de transferencia de hipertexto HTTP (HyperText Transfer Protocol) es el protocolo usado en cada transacción en la Web. HTTP fue desarrollado por el consorcio W3C (World Wide Web Consortium) y la IETF (Internet Engineering Task Force), colaboración que culminó en 1999 con la publicación de una serie de RFC (Request For Comments), siendo el más importante de ellos el RFC 2616, que especifica la versión 1.1.

HTTP define la sintaxis y la semántica que utilizan los elementos software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. Al cliente que efectúa la petición (un navegador o un spider) se lo conoce como "user agent" (agente del usuario). A la información transmitida se la llama recurso y se la identifica mediante un URL. Los recursos pueden ser archivos, el resultado de la ejecución de un programa, una consulta a una base de datos, la traducción automática de un documento, etc.

HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Para esto se usan las cookies, que es información que un servidor puede almacenar en el sistema cliente. Esto le permite a las aplicaciones web instituir la noción de "sesión", y también permite rastrear usuarios ya que las cookies pueden guardarse en el cliente por tiempo indeterminado.

Una transacción HTTP está formada por un encabezado seguido, opcionalmente, por una línea en blanco y algún dato. El encabezado especificará cosas como la acción requerida del servidor, o el tipo de dato retornado, o el código de estado. El uso de campos de encabezados enviados en las transacciones HTTP le da gran flexibilidad al protocolo. Estos campos permiten que se envíe información descriptiva en la transacción, permitiendo así la autenticación, cifrado e identificación de usuario. Un encabezado es un bloque de datos que precede a la información propiamente dicha, por lo que muchas veces se hace referencia a él como metadato (porque tiene datos sobre los datos). Si se reciben líneas de encabezado del cliente, el servidor las coloca en las variables de ambiente de CGI con el prefijo HTTP_ seguido del nombre del encabezado. Cualquier carácter guión (-) del nombre del encabezado se convierte a caracteres "_". El servidor puede excluir cualquier encabezado que ya esté procesado, como *Authorization*, *Content-type* y *Content-length*. El servidor puede elegir excluir alguno o todos los encabezados si incluirlos exceden algún límite del ambiente de sistema. Ejemplos de esto son las variables *HTTP_ACCEPT* y *HTTP_USER_AGENT*.

- *HTTP_ACCEPT*. Los tipos MIME que el cliente aceptará, dado los encabezados HTTP. Otros protocolos quizás necesiten obtener esta información de otro lugar. Los elementos de esta lista deben estar separados por una coma, como lo dice la especificación HTTP: tipo, tipo.

- *HTTP_USER_AGENT*. El navegador que utiliza el cliente para realizar la petición. El formato general para esta variable es: software/versión librería/versión.

  El servidor envía al cliente:

- Un código de estado que indica si la petición fue correcta o no. Los códigos de error típicos indican que el archivo solicitado no se encontró, que la petición no se realizó de forma correcta o que se requiere autenticación para acceder al archivo.
- La información propiamente dicha. Como HTTP permite enviar documentos de todo tipo y formato, es ideal para transmitir multimedia, como gráficos, audio y video. Esta libertad es una de las mayores ventajas de HTTP.
- Información sobre el objeto que se retorna.

Se debe tener en cuenta que la lista no es una lista completa de los campos de encabezado y que algunos de ellos sólo tienen sentido en una dirección.

Ejemplo de un diálogo HTTP

Para obtener un recurso con el URL http://www.example.com/index.html se realizan los siguientes pasos:

1. Se abre una conexión al host www.example.com, con el puerto 80, que es el puerto por defecto para HTTP.
2. Se envía un mensaje en el estilo de la Figura A.2.

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: nombre-cliente
[Línea en blanco]
```

Figura A.2. Ejemplo de un diálogo HTTP – mensaje enviado

La respuesta del servidor está formada por encabezados seguidos del recurso solicitado, en el caso de una página web:

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 2003 23:59:59 GMT
Content-Type: text/html
Content-Length: 1221

<html>
<body>
<h1>Página principal de tuHost</h1>
(Contenido)
 .
 .
 .
</body>
</html>
```

Figura A.3. Ejemplo de un diálogo HTTP – respuesta del servidor

A.3 El lenguaje XML

El lenguaje de marcado extensible XML (Extensible Markup Language) [XML] es el lenguaje utilizado para definir el formato de los documentos o mensajes. Su principal característica es que describe la información intercambiada entre las aplicaciones pero no le agrega ninguna semántica, es decir, no dice nada acerca del significado de la información. XML se ha convertido en el estándar de facto para la descripción de datos ha ser intercambiados sobre la Web. A menudo se escucha decir que los servicios web no son más que el protocolo de transferencia de hipertexto HTTP (Hypertext Transfer Protocol) y XML. Esta no es realmente la respuesta completa, pero sí una buena parte de ella. Los servicios web usan mensajes XML, y HTTP es el protocolo de transporte para estos mensajes.

XML es un lenguaje de marcado, tal como el lenguaje de marcado de hipertexto HTML (Hypertext Markup Language), que consiste de marcas y texto. Las marcas, a su vez, se componen de etiquetas individuales (denominadas tags). La Figura A.4 presenta un ejemplo de documento XML mostrando las marcas resaltadas [ZTP05].

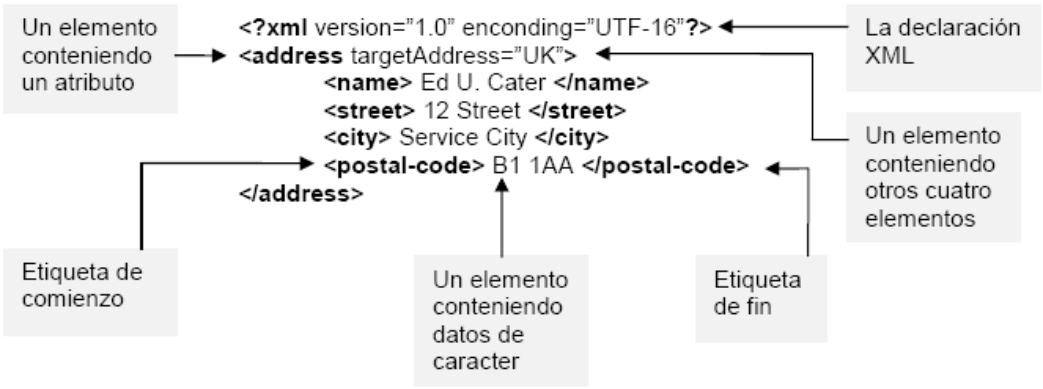


Figura A.4. Estructura de un documento XML

La diferencia principal entre un documento XML y un documento HTML es que este último no indica lo que se esta representando, mientras que XML contiene datos que se autodefinen, es decir, describen el contenido de lo que etiquetan. Por ello, XML comprende el uso de etiquetas denominadas *tags* que identifican los contenidos del documento, y al hacerlo, lo describen. Una etiqueta XML identifica información dentro del documento, como así también la estructura de dicha información.

Algunas de las reglas básicas de los documentos XML son:

- Hay dos tipos de construcciones: las *marcas* y los *datos de caracter* (todo lo demás).
- Todo documento XML se compone de *elementos*. Cada elemento está delimitado por una etiqueta de comienzo y otra de fin, a no ser que sea vacío. Los elementos vacíos constan de una única etiqueta. Los nombres de las etiquetas son arbitrarios y no pueden contener espacios.
- Siempre hay un *elemento raíz*, cuya etiqueta de inicio ha de ser la primera de todas y la de cierre la última de todas.
- Cada elemento *puede contener* datos de caracter, elementos, ambas cosas a la vez o puede estar vacío.
- No se puede mezclar la *anidación de las etiquetas*: los elementos deben abrirse y cerrarse en orden.
- Los elementos pueden tener *atributos* (propiedades) que ofrecen información sobre ellos. Los valores de los atributos deben ir entrecomillados. Tanto atributos como valores son arbitrarios.
- *Mayúsculas y minúsculas* no son intercambiables.

- El *espacio en blanco* es libre, se puede utilizar para leer mejor el documento.

Existen otras construcciones como los *comentarios*, *secciones CDATA*, que sirven para introducir texto que el analizador tendrá en cuenta como datos de carácter, sin interpretarlo como XML, y *entidades predefinidas*, que permiten incluir ciertos caracteres sin que sean tomados como XML, por ejemplo:

- & para el &
- < para el <
- > para el >
- ' para el '
- " para el "

Otro elemento importante son los *espacios de nombres* (namespaces), los cuales sirven para evitar colisiones entre elementos del mismo nombre, y en general, para distinguir los distintos grupos de elementos en un mismo documento. Cada espacio de nombres se asocia con una URL, que sólo sirve como identificador único y no tiene por qué ser válida.

Como se mencionó anteriormente, XML no agrega semántica a los documentos descriptos. En la Figura A.5 se puede observar un marcado XML que identifica cierta información como *bookshelf*. Dicho marcado también describe la estructura de *bookshelf*. La estructura de *bookshelf* incluye dos ítems subordinados denominados *book*. Cada *book* tiene tres ítems subordinados identificados como *title*, *author* y *price*.

```
<bookshelf>
     <book>
        <title>My Life and Times</title>
        <author>Felix Harrison</author>
        <price>39.95</price>
     </book>
</bookshelf>
```

Figura A.5. Documento XML

Si bien podría parecer que las etiquetas <book> y <title> le dan significado de una manera inherente a la información que ellas identifican, en realidad no lo hacen. La información asociada con las etiquetas XML sólo tiene significado si las personas asocian un significado particular con una etiqueta particular. Si las personas (1) acuerdan en el significado de una etiqueta, digamos que la etiqueta <book> se utiliza para identificar a un

libro y que las etiquetas <title>, <author> y <price> identifican el título, el autor y el precio del libro, respectivamente, y (2) utilizan dichas etiquetas de manera consistente, esto le proporciona a las personas una manera de intercambiar datos. Sus aplicaciones pueden enviarse entre sí documentos XML y procesar la información contenida en estos documentos, apoyándose en una comprensión común de los significados asociados con las etiquetas XML. Luego, las aplicaciones capaces de interpretar estas etiquetas pueden procesar la información de acuerdo con el significado de la información y su organización.

Cada documento XML posee una estructura lógica y una física. La estructura lógica del documento comprende una serie de declaraciones, elementos, comentarios, referencias e instrucciones de procesamiento, que se indican en el documento mediante marcas explícitas. La estructura física del documento comprende una serie de unidades llamadas *entidades*, las cuales indican los datos que contendrá el documento. Las estructuras lógica y física deben anidarse de forma correcta.

Los documentos XML poseen una estructura bien-formada. Según la especificación de XML de la W3C, un documento XML está bien formado si:

- Tomado como un todo, cumple la regla denominada "document".
- Respeta todas las restricciones de buena formación dadas en la especificación.
- Cada una de las entidades analizadas que se referencia directa o indirectamente en el documento está bien formada.

Cumplir la regla "document" antes mencionada significa:

- Que contiene uno o más elementos.
- Hay exactamente un elemento, llamado raíz, o elemento documento, del cual ninguna parte aparece en el contenido de ningún otro elemento.
- Para el resto de los elementos, si la etiqueta de comienzo está en el contenido de algún otro elemento, la etiqueta de fin está en el contenido del mismo elemento. Es decir, los elementos delimitados por etiquetas de principio y final se anidan adecuadamente mutuamente.

Los documentos XML deben tener una estructura jerárquica en lo que respecta a las etiquetas que delimitan sus elementos. Esto significa que los elementos deben estar correctamente anidados y que no se pueden solapar entre ellos. Además los elementos con contenido deben estar correctamente cerrados. En el ejemplo, cada etiqueta XML tiene una etiqueta de cierre, tal como se puede observar en el ejemplo con

<bookshelf>...</bookshelf> y las etiquetas que comienzan dentro de otra etiqueta terminan antes de la terminación de la misma, es decir, están completamente anidadas dentro de la misma, como se puede observar con las etiquetas </title>, </author>, y </price>, que aparecen antes de </book>, y en ese orden relativo.

Un documento XML generalmente está asociado con un *schema* que especifica sus "reglas gramaticales". En otras palabras, el *schema* especifica qué etiquetas están permitidas dentro de un documento, la estructura de esas etiquetas, y otras reglas relacionadas con las etiquetas, tales como el tipo de dato que se espera dentro de una etiqueta (o la ausencia de dato si es una etiqueta vacía). Debido a que los documentos XML válidos deben estar bien-formados y conformes al *schema* asociado, resulta relativamente fácil procesar documentos XML. Como consecuencia de esto, XML ha sido adoptado en forma generalizada como el lenguaje de datos para los servicios web.

A.4 El Protocolo SOAP para la representación de los mensajes

SOAP (Simple Object Access Protocol) [SOAP] es un protocolo simple de acceso a objetos. Es un protocolo ligero de mensajes XML que se usa para codificar la información de los mensajes de petición y respuesta de los servicios web que se envían a través de la red. Los mensajes SOAP son independientes de los sistemas operativos y de los protocolos, y pueden ser transportados usando una variedad de protocolos de Internet, incluyendo SMTP y HTTP. SOAP es un protocolo que define precisamente como realizar la comunicación de métodos, es decir, cómo codificar las llamadas a los métodos de un servicio web y cómo debe el servicio web codificar el resultado.

Si bien SOAP es un protocolo, éste no es exactamente un protocolo de comunicación entre mensajes como lo es HTTP. Básicamente, SOAP son documentos XML y se necesita otro protocolo para la transmisión de estos documentos como puede ser HTTP, o cualquier otro protocolo de comunicación capaz de transmitir textos.

A.4.1 Estructura de un mensaje SOAP

Los mensajes SOAP están compuestos por tres secciones: una etiqueta principal llamada *Envelope*, que está dividida en una cabecera o *Header* y un cuerpo o *Body*.

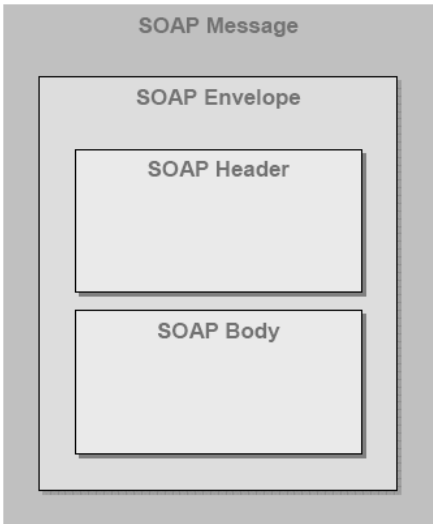


Figura A.6. Elementos de un mensaje SOAP

La Figura A.7 presenta un ejemplo de un mensaje SOAP que un cliente podría enviar para invocar un método sobre un servidor SOAP [ZTP05].

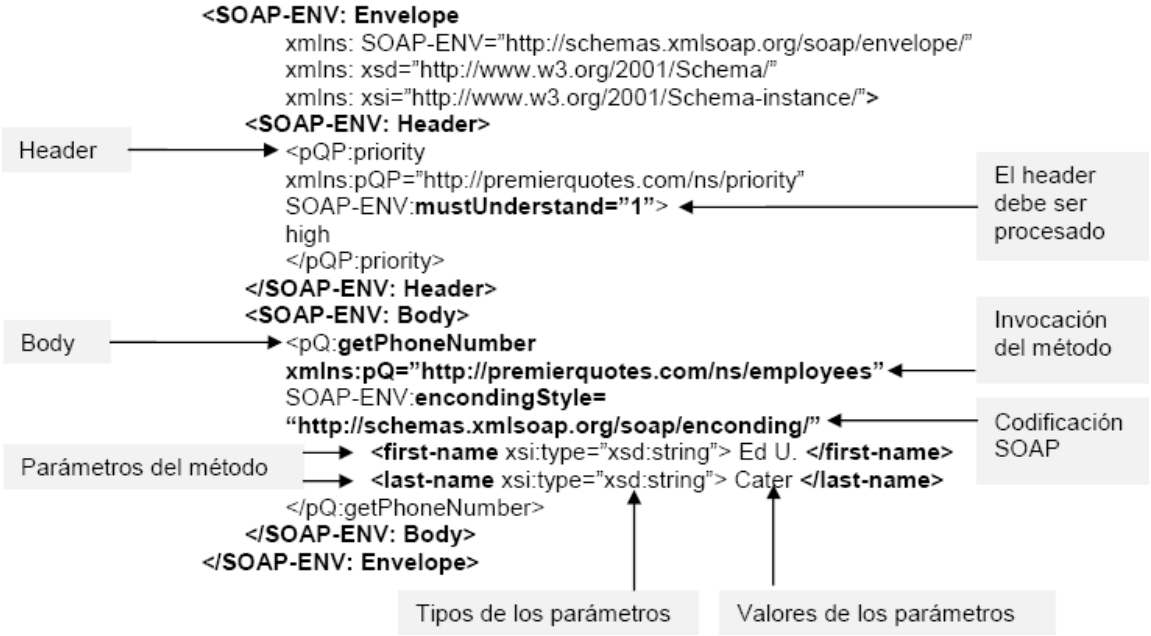


Figura A.7. Estructura de un mensaje SOAP

El elemento *envelope*

Este elemento contiene toda la información del mensaje SOAP, sin especificar una dirección. Es el elemento de mayor nivel, y está formado por grupos de notas e información separada por *tags*. Puede contener declaraciones de *namespaces* y atributos.

En el ejemplo, el elemento *envelope* declara tres espacios de nombres:

- El prefijo *SOAP-ENV* identifica los elementos definidos en SOAP, denominados *envelope*, *header* y *body*, y los atributos *mustUnderstand* y *encodingStyle*.
- El prefijo *xsd* se refiere a los esquemas XML.
- El prefijo *xsi* aparece delante de los tipos de los atributos, especificando que el tipo de los elementos first-name y last-name es el tipo *string* de XML.

El elemento *header*

Este elemento es opcional. Tiene información para manejo y control, como números de cuenta o identificación de cliente. Puede tener también información sobre el algoritmo usado para *encriptar* el body o la clave pública necesaria para leer el texto *encriptado* o un identificador de transacción o algún mecanismo para prevención de *deadlock*. Si está presente, debe ser el primer elemento hijo del elemento *envelope*. Puede usarse el atributo *encodingStyle*, y también los atributos *mustUnderstand* y SOAP actor.

En el ejemplo, el elemento *header* contiene un elemento con el nombre local *priority* y el espacio de nombres *http://premierquotes.com/ns/priority*. El valor de priority es alto. El valor del atributo mustUnderstand=”1” le dice al proveedor del servicio web que éste debe entender y procesar la semántica del elemento *header*.

El elemento *body*

En el *Body* se registra la información sobre la invocación del método del servicio. Debe ser un elemento hijo del *envelope*, y debe seguir al *header*, si este está presente. Provee un mecanismo para intercambiar información con el receptor último del mensaje.

En el ejemplo, el elemento *body* define el método *getPhoneNumber*, con los parámetros *first-name* y *last-name*, y los tipos de datos y valores asociados. Aparentemente, el solicitante del servicio web está interesado en conocer el número de teléfono de *Ed U. Cater*. El elemento que representa el nombre del método comienza con la declaración del espacio de nombres *http://premierquotes.com/ns/employees*. Este espacio de nombres representa el servicio sobre el cual el método se ejecutará.

A.4.2 Atributos de un mensaje SOAP

SOAP predefine una serie de atributos que tienen un significado especial para los clientes y servidores SOAP, ellos son:

- *encodingStyle*: indica las reglas de serialización utilizadas en el mensaje SOAP. El estilo de codificación generalmente define los tipos de datos y correspondencia de datos entre dos partes que poseen diferentes representaciones de datos. Con respecto a SOAP, la codificación (encoding) transforma la representación específica de los datos de una aplicación a un formato estándar. La decodificación convierte nuevamente esta representación estándar a la representación específica de la aplicación. La acción de transformar una representación de datos en otra, y luego volver a la representación original se denomina *serialización* y *deserialización*. Los términos *marshalling* y *unmarshalling* también pueden usarse como alternativos. El conjunto de reglas de *serialización* que se han aplicado para transformar datos específicos de la aplicación a datos del mensaje SOAP se identifica a través de uno o varios URIs. En el ejemplo de la sección anterior, el valor del atributo *encodingStyle* es *http://schemas.xmlsoap.org/soap/enconding/*, lo cual revela las reglas de serialización que se han aplicado al contenido del elemento *body*.
- *actor*: está relacionado al *header* del mensaje y permite identificar el actor destinatario del *header*, es decir, la aplicación que debería procesar el *header*. El valor de la URI (Uniform Resource Identifier) indica quién es el receptor del header. Un intermediario descubrirá si debe procesar el header chequeando el atributo actor. Si el valor de dicho atributo fuera http://schemas.xmlsoap.org/soap/actor/*next*, esto indicaría cuál es la siguiente aplicación en el camino del mensaje. Éste podría ser un intermediario o el último receptor del mensaje.
- *mustUnderstand*: también esta relacionado con el *header* del mensaje SOAP y define si el receptor del mensaje debe entender el contenido del *header*. El receptor del mensaje que debe entender el *header* se identifica por el atributo *actor*. Puede tener valores ‘0’ o ‘1’. El valor ‘1’ indica que el actor destinatario del mensaje SOAP debe entender y procesar la semántica del header. El valor ‘0’ indica que el actor destinatario puede, pero no debe, procesar la entrada del mensaje.

A.5 El lenguaje WSDL para la descripción de los Servicios Web

WSLD (Web Service Description Language) [WSDL2.0-0] [WSDL2.0-1] [WSDL2.0-2] es el lenguaje usado para describir la interfaz de un servicio web como un conjunto de métodos, capaces de intercambiar mensajes, es decir, recibir llamadas con sus

parámetros correspondientes y generar respuestas con el resultado que le corresponda. WSDL es un archivo XML que describe el conjunto de estos métodos expuestos por un servicio web. La descripción incluye el número de argumentos, y tipo de cada uno de los parámetros de los métodos, y la descripción de los elementos que retornan.

A.5.1 Estructura de un documento WSDL

WSDL es un documento XML que se utiliza para describir los mensajes SOAP y cómo estos mensajes son intercambiados. WSDL describe un servicio web en dos grupos de secciones: una abstracta y una concreta. Dentro de cada sección se utilizan un número de constructores que promueven la reusabilidad de la descripción y permiten separarla de los detalles de diseño. En la *sección abstracta* se describe el servicio web en términos de los mensajes que este envía y recibe. La *sección concreta* contiene especificaciones del transporte y detalles de formato.

En WSDL 1.1 [WSDL1.1], el documento que describe un servicio web está compuesto por un elemento raíz llamado *definitions*, que a su vez está compuesto por los siguientes elementos:

- *types*: provee la definición de los tipos de datos utilizados para describir los mensajes a intercambiar.
- *message***:** representa una definición abstracta de los datos a ser transmitidos entre el servidor y el cliente.
- *portType***:** define un conjunto de operaciones abstractas. Cada operación se refiere a un mensaje de entrada y mensajes de salida.
- *binding***:** define un protocolo concreto y las especificaciones de formato de datos para las operaciones y los mensajes definidos en un *portType* particular.
- *port:* el cual especifica una dirección de enlace, definiendo un punto final de comunicación.
- *service*: se usa para agrupar un conjunto de puertos relacionados.

Los elementos *types, messages* y *portTypes* se ubican en las secciones correspondientes a las definiciones abstractas. Estas están en la parte superior del

documento y definen el mensaje SOAP de una manera independiente del lenguaje y la plataforma. Por su parte, los elementos *binding* y *service* se ubican en las secciones correspondientes a las definiciones concretas. Aquí se encuentran las especificaciones sobre la implementación concreta del servicio web.

La nueva versión de este lenguaje, WSDL 2.0 [WSDL2.0-0], hereda la mayoría de los principios arquitecturales de WSDL 1.1, incluyendo la capas de descripción, flexibilidad en el estilo de los autores, capacidad de modularización y extensibilidad, e incorpora, además, los conceptos de interface abstracta, protocolo de *enlace* (binding*),* y *puntos finales de servicios* (service endpoints*)*.

Algunos de los cambios incorporados en WSDL 2.0 son:

- agrega más semántica al lenguaje de descripción
- remueve el elemento *message*. Este se especifica dentro del elemento types, usando *XML Schema Type System*
- no soporta la sobrecarga de operadores
- renombra el elemento *portTypes* a interfaces. La herencia de interfaces se consigue el atributo *extends* en el elemento interface.
- renombra los *ports* a *endpoints*
- incorpora patrones de mensajes abstractos.

En el campo de la computación, un lenguaje consiste de un conjunto (posiblemente infinito) de sentencias, cada una de ellas es un *string* finito de símbolos o caracteres. De esta manera, la especificación de un lenguaje debe definir el conjunto de sentencias aceptadas, e indicar el significado de cada sentencia. De hecho, este es el propósito de la especificación WSDL 2.0.

Sin embargo, para evitar la dependencia de la codificación de caracteres, WSDL 2.0 se define en términos de un *conjunto de datos abstractos de XML* (Infoset XML). Específicamente, un documento consiste de un elemento *description*, que se ajusta a las limitaciones adicionales de WSDL 2.0. Dado que un *Infoset XML* puede crearse a partir de más de un documento físico, un documento no necesariamente se corresponde con un simple documento físico, la palabra *document* se utiliza figurativamente, sólo por conveniencia. Además, dado que WSDL 2.0 provee mecanismos para *importar* e *incluir*, un documento puede referenciar otros documentos para facilitar la organización y reuso.

El diagrama de la Figura A.8 presenta el Infoset XML que describe un documento WSDL 2.0.

En esta versión los elementos principales que componen el documento que describe el servicio web, los cuales están contenidos en el elemento *description*, son:

- *types*: provee la definición de los tipos de datos utilizados para describir los mensajes a intercambiar.
- *interface*: describe la secuencia de mensajes que el servicio envía o recibe.
- *binding*: define un protocolo concreto y las especificaciones de formato de datos para las operaciones y los mensajes definidos en un *portType* particular.
- *service*: se usa para agrupar un conjunto de puertos relacionados.

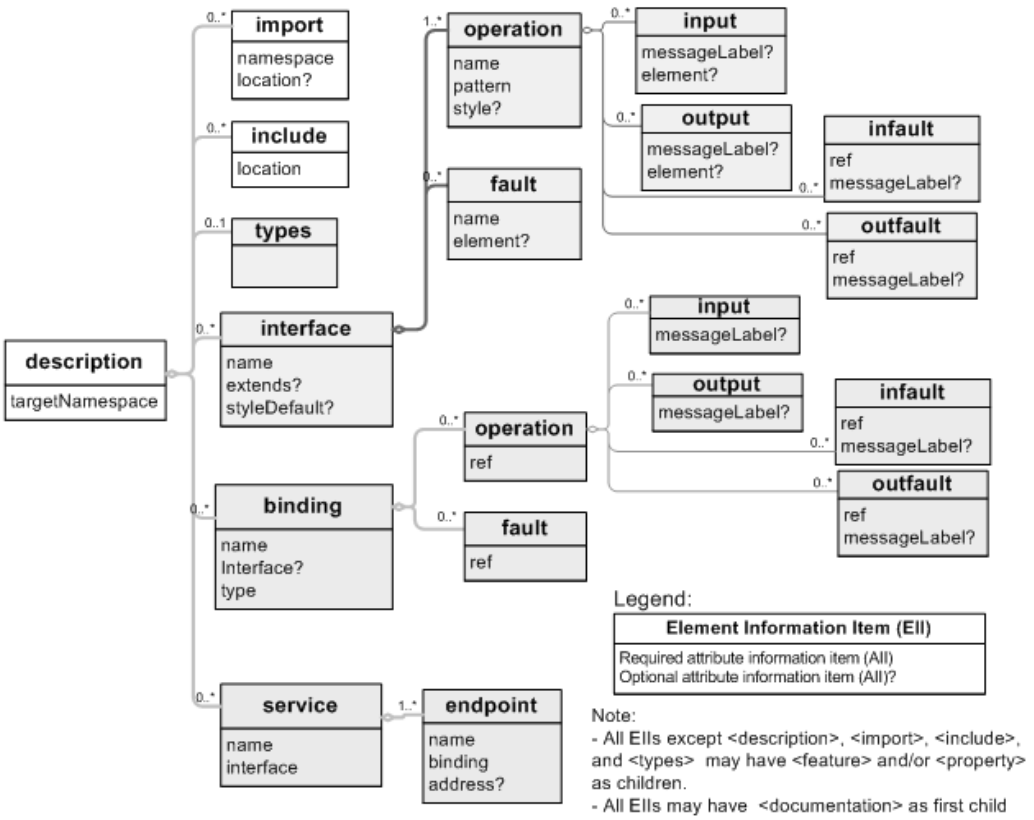


Figura A.8. Elementos de un Servicio Web

La sección abstracta contiene los elementos *types* e *interface* (*message* y *operations*), mientras que la sección concreta contiene los elementos *binding* y *service.*

```
<description
      targetNamespace="xs:anyURI" >
  <documentation />*
  [ <import /> | <include /> ]*
  <types />?
  [ <interface /> | <binding /> | <service /> ]*
</description>
```

Figura A.9. Elementos de un Servicio Web

A.5.1.1 El Elemento Types

El elemento *types* incluye la definición de los tipos de datos que son relevantes para el intercambio de mensajes y que se necesitarán posteriormente para la definición. Ellos son los parámetros de entrada y salida respectivamente.

La sintaxis para definir un elemento *types* es:

```
<description>
  <types>
    <documentation />*
    [ <xs:import namespace="xs:anyURI" schemaLocation="xs:anyURI"? /> |
      <xs:schema targetNamespace="xs:anyURI"? /> |
      other extension elements ]*
  </types>
</description>
```

Figura A.10. El elemento Types

Para definir los tipos de datos del servicio web se utiliza la sintaxis de *XML Schema*. XML Schema es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML.

A.5.1.2 El Elemento Interface

El elemento *interface* describe la secuencia de mensajes que el servicio envía o recibe. Para ello agrupa los mensajes en operaciones. Una operación es una secuencia de mensajes de entrada y salida, y una interface es un conjunto de operaciones.

Una *interface* puede, opcionalmente, extender otra interface, pero la misma no puede aparecer en el conjunto de interfaces que ella extiende, ello para evitar definiciones circulares. El conjunto de operaciones disponibles en un interface incluye todas las operaciones definidas por las interfaces que ella extiende, directa o indirectamente, más las operaciones que ella define directamente. Las propiedades de una interface son:

- nombre (name)
- interfaces que extiende (extends)
- componentes de falla (fault)
- operaciones (operation)

La sintaxis para definir un elemento *interface* es:

```
<description>
  <interface
        name="xs:NCName"
        extends="list of xs:QName"?
        styleDefault="list of xs:anyURI"? >
    <documentation />*
    [ <fault /> | <operation /> ]*
  </interface>
</description>
```

Figura A.11. El elemento Interface

El componente *fault* describe las fallas que pueden ocurrir durante la invocación de una operación de la interface. Una falla es un evento que ocurre durante la ejecución de un intercambio de mensajes que perturba el flujo normal de dichos mensajes. Típicamente, la falla surge cuando una de las partes no puede comunicar una condición de error dentro del flujo normal del mensaje, o desea terminar el intercambio de mensaje. El mensaje de falla se puede usar para comunicar cuál es la razón del error. Dentro del elemento *fault* se declara una falla, dándole un nombre e indicando el contenido del mensaje de falla. Cuándo y cómo el mensaje de falla ocurre se indica dentro del elemento *operation*.

El componente *operation* describe una operación que ocurre dentro de la interface que la soporta. Una operación es una interacción con el servicio, que consiste de un conjunto de mensajes intercambiados entre el servicio y las otras partes involucradas

en la interacción. El secuenciamiento y la cardinalidad de los mensajes involucrados en una interacción particular es gobernado por el patrón de intercambio de mensajes usado por la operación.

Los patrones de intercambio de mensajes [WSDL2.0-2] definen la secuencia y cardinalidad de los mensajes abstractos listados en el componente operación. Los patrones que propone WSDL son:

- *In-Only*: consiste de un mensaje, que define el mensaje recibido desde un nodo. No se pueden generar fallas.
- *Robust In-Only*: consiste de un mensaje, que define el mensaje recibido desde un nodo. Cualquier mensaje, incluyendo el primero, puede generar un mensaje de falla en respuesta.
- *In-Out*: consiste de exactamente dos mensajes. Uno define el mensaje recibido y otro define el mensaje enviado. Cualquier mensaje que siga al primero puede reemplazarse por una falla.
- *In-Optional-Out*: consiste de uno o dos mensajes, uno (obligatorio) que define el mensaje recibido y otro (opcional) define el mensaje enviado. Cualquier mensaje, incluyendo el primero, puede generar un mensaje de falla en respuesta.
- *Out-Only*: consiste de un mensaje, que define el mensaje enviado hacia un nodo. No se pueden generar fallas.
- *Robust Out-Only*: consiste de un mensaje, que define el mensaje enviado hacia un nodo. Cualquier mensaje, incluyendo el primero, puede reemplazarse por una falla.
- *Out-In*: consiste de exactamente dos mensajes. Uno define el mensaje enviado y otro define el mensaje recibido. Cualquier mensaje que siga al primero puede reemplazarse por una falla. Cualquier mensaje que siga al primero puede reemplazarse por una falla.
- *Out-Optional-In*: consiste de uno o dos mensajes, uno (obligatorio) que define el mensaje enviado y otro (opcional) define el mensaje recibido. Cualquier mensaje, incluyendo el primero, puede generar un mensaje de falla en respuesta.

WSDL se refiere a estas primitivas en la definición de las operaciones. Las operaciones hacen referencia a lo mensajes involucrados a través del atributo

message. Por defecto, el contenido de los mensajes se define por medio del sistema de tipos basado en XML.

A.5.1.3 El Elemento Binding

El elemento *Binding* define los detalles de implementación concretos, que serán necesarios para acceder al servicio.

La sintaxis para definir un elemento *binding* es:

```
<description>
  <binding
        name="xs:NCName"
        interface="xs:QName"?
        type="xs:anyURI" >
    <documentation />*
    [ <fault /> | <operation /> ]*
  </binding>
</description>
```

Figura A.12. El elemento Binding

El *binding* debe especificar la interface a la que se aplica y definir *bindings* para todas las fallas definidas en la interface, que son referenciadas en las operaciones. Las propiedades de este componente son:

- nombre (name)
- interface (interface), para la cual se especifica el componente binding
- tipo (type), que indica los detalles de binding concretos
- falla (binding faults)
- operaciones (binding operations)

El componente *fault* describe un binding concreto para una falla particular descripta dentro de la interface. Es importante aclarar que la falla no ocurre por sí sola, ocurre como parte de un intercambio de mensajes, definidos en el componente *operation*. Así, la información de falla en el componente binding describe cómo las fallas que ocurren dentro de un intercambio de mensajes de una operación serán formateadas y transportadas.

El componente *operations* define el formato de mensaje concreto y el protocolo de interacción asociados con una operación de interface particular.

A.5.1.4 El Elemento Service

El elemento *Service* asocia un binding con el URL donde está realmente el servicio a través de un *endpoint*. Los *endpoint*s son lugares alternativos que proveen el servicio.

La sintaxis para definir un elemento *service* es:

```
<description>
  <service
        name="xs:NCName"
        interface="xs:QName" >
    <documentation />*
    <endpoint />+
  </service>
</description>
```

Figura A.13. El elemento Service

Las propiedades de este componente son:

- nombre (name)
- interface (interface), la interface que el servicio instancia
- punto final (endpoint)

El componente *endpoint* define las particularidades de un punto final específico en el cual el servicio está disponible. El *endpoint* hace referencia al componente binding que tiene asociado (binding attribute), y a la dirección real del endpoint (endpoint attribute).

A.6 El directorio UDDI para el registro y localización de Servicios Web

El protocolo denominado Universal Description, Discovery and Integration (UDDI) [UDDI] es un estándar para la publicación y registro de los servicios web. Es el más dominante entre los mecanismos de descubrimiento de servicios web propuestos por la W3C [WSA].

El protocolo UDDI define el registro y los protocolos asociados para la búsqueda y localización de los servicios web. Tiene por objetivo crear una plataforma independiente para describir servicios, descubrir servicios e integrar servicios. UDDI tiene diferentes funciones dependiendo de la perspectiva de quien lo utilice, ya que puede verse como un proveedor de servicios para realizar la publicación de servicios web, como un registro de servicios o como un servicio de descubrimiento para localizar y enlazar un servicio web con un cliente.

Con respecto a la publicación de servicios web, un negocio puede registrar tres tipos de información en un nodo UDDI [UDDI], [ZTP05], [ACKM04]:

- *páginas blancas*: corresponde a la información básica de un negocio incluyendo el nombre del negocio, dirección, información de contacto, e identificadores únicos.
- *páginas amarillas*: contienen la información que describe las categorías a las que pertenecen los servicios web, basadas en ontologías establecidas.
- *páginas verdes*: corresponde a la información técnica que describe el comportamiento y funciones soportadas por un servicio web. Esta información incluye su descripción y ubicación.

Las especificaciones UDDI definen de qué manera publicar y descubrir información acerca de los servicios, mediante una base denominada *UDDI Business Registry*. Más específicamente, las especificaciones UDDI definen *schemas* para registrar los servicios web, basados en un Modelo de Datos de UDDI, y *APIs* para realizar la *consulta* y *publicación* de los servicios.

A.6.1 El Modelo de Datos del Registro UDDI

El Modelo de Datos de UDDI identifica los tipos de estructuras de datos XML incluidos en una entrada del registro UDDI, correspondiente a un servicio. Se debe pensar a un registro UDDI como las “Páginas Amarillas” de los servicios web. Al igual que el directorio de Página Amarillas provee información sobre números telefónicos, un registro UDDI proporciona información acerca de un servicio, tal como el nombre del servicio y una descripción de lo que éste hace, una dirección donde el servicio puede ser accedido, y una descripción de la *interface* de acceso al servicio. Usando la *UDDI interface*, una

aplicación comercial de un negocio se puede conectar dinámicamente con los servicios web provistos por una aplicación comercial externa de otro negocio.

Las cuatro principales estructuras de datos XML que define el Modelo de Datos UDDI son [UDDI], [Papazoglou08], [ZTP05], [ACKM04]:

- *BusinessEntity* (información del negocio): Información acerca de la organización que publica el servicio. Información relativa a una familia de servicios, categorías, contactos, URLs, y otros necesarios para interactuar con un determinado negocio.
- *BusinessService* (información del servicio): Información descriptiva acerca de grupos de servicios web. Describe la función del servicio en el negocio.
- *BindingTemplate* (información del enlace): Detalles técnicos necesario para invocar un servicio web, incluyendo URLs, información sobre nombres de métodos, tipos de argumentos, etc.
- *tModel* (detalles de la especificación del servicio): Metadatos de las distintas especificaciones que implementa un servicio web determinado.

Estas cuatro estructuras y sus relaciones se muestran en la Figura A.14.

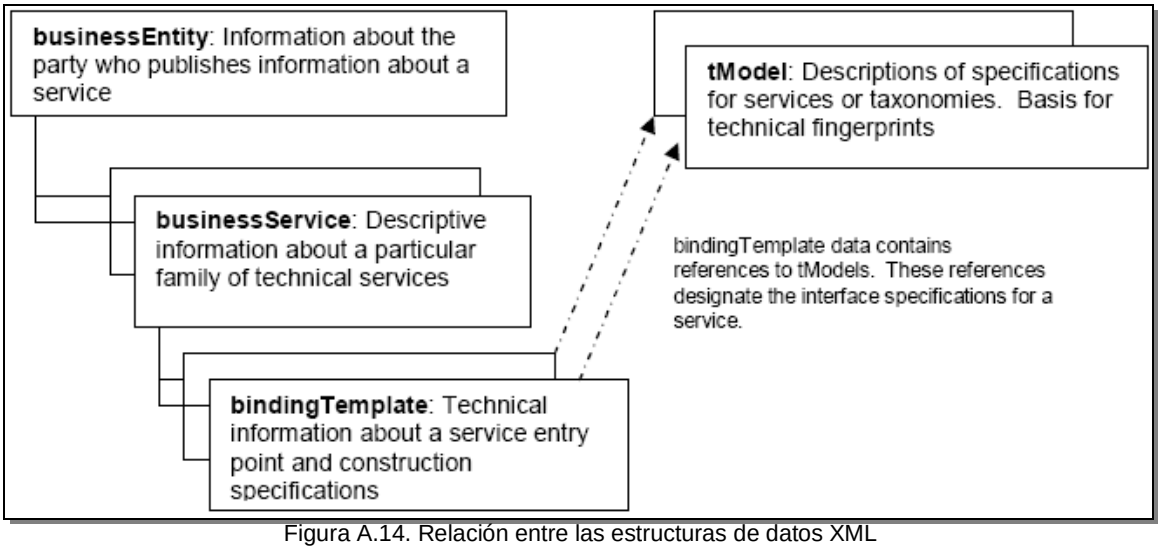


Figura A.14. Relación entre las estructuras de datos XML

A.6.2 Interfaz de Programación (API) de UDDI

Para que la base de registros UDDI funcione se utilizan las APIs de UDDI, que permiten realizar la *consulta* y *publicación* de información de los servicios. La *consulta* ubica la

información de un servicio y su descripción. La *publicación* se utiliza para crear, almacenar y actualizar la información de un nodo UDDI. Algunas de las operaciones utilizadas son *find business*, *find service*, *get businessDetail*, *get tModelDetail*, *delete business*, *delete service*, *save business* y *save service*.

Las APIs soportan tres tipos de patrones de consultas. El *browse pattern* permite buscar un servicio usando las APIs "find_xxx". Estas APIs permiten realizar una búsqueda en el UDDI Registry basada en un criterio de búsqueda. El *drill-down pattern* es soportado por las APIs de la forma "get_xxx", las cuales obtienen los detalles completos de una entrada específica del UDDI Registry. El *invocation pattern* retorna una entrada del UDDI Registry, pero sólo cuando una actualización es necesaria. Las APIs de la forma "save_xxx" o "delete_xxx" permiten la creación y destrucción de cualquiera de las estructuras XML del nivel superior, descriptas en el Modelo de Datos de UDDI.

Como se ha mencionado, las estructuras XML almacenan la información necesaria para registrar un servicio web en el *UDDI Registry*, mientras que las APIs de UDDI permiten buscar y publicar los servicios web en la base de registro de UDDI. En la siguiente tabla se puede observar la relación que existe entre los tipos de información almacenados en un nodo UDDI (páginas blancas, amarillas y verdes), las operaciones necesarias para registrar y acceder a esta información y las correspondientes estructuras u objetos XML asociados.

INFORMACIÓN	OPERACIONES	OBJETOS XML
Páginas blancas: información para contactar al negocio	Publicar: cómo el proveedor del servicio web registra al servicio.	BusinessEntity
Páginas amarillas: información que categoriza los servicios	Encontrar: cómo una aplicación de un negocio encuentra un servicio web particular.	BusinessService
Páginas verdes: información técnica del servicio	Enlazar: cómo una aplicación de un negocio se conecta e interactúa con el servicio web, después de haberlo encontrado.	BindingTemplate tModel

Tabla A.1. Componentes que participan en el registro de una especificación UDDI

UDDI asume que los requerimientos y respuestas son objetos UDDI que se envían a través de mensajes SOAP. La ventaja de usar UDDI sobre otras formas de encontrar información es que los registros y descripciones del servicio están en formato XML, es decir, se basan en un lenguaje estándar. Por otra parte, una dificultad que posee este directorio de registro y localización de servicios web es que, debido a que XML no agrega semántica a la información, las búsquedas son un tanto limitadas.

ANEXO B.

Características de Calidad de los Servicios Web

Este anexo presenta las características de calidad de los servicios web, detallando los conceptos básicos, las características principales organizadas en categorías, y los diferentes enfoques propuestos por la W3C para el soporte de las características de calidad. También se presenta una breve explicación de los acuerdos SLA, como una de las soluciones posibles para establecer la correspondencia entre las características requeridas por el cliente y las ofrecidas por el proveedor. Y en cuanto a la incorporación de las características de calidad en el lenguaje WSDL, se presentan las especificaciones *WS-Policy* y *WS-Policy Attachment.*

La sección 1 describe los conceptos básicos de las características de calidad. La sección 2 detalla las cuatro categorías consideradas y las características que incluye cada una de ellas. La sección 3 detalla los tres enfoques propuestos por la W3C para la incorporación de las características de calidad de los servicios web. La sección 4 describe los acuerdos SLA, y finalmente, la sección 5 presenta dos especificaciones que permiten incorporar al lenguaje WSDL la descripción de los aspectos no funcionales del servicio web.

B.1 Introducción

La descripción de un servicio web tiene dos componentes principales interrelacionados: sus características funcionales y no funcionales [Papazoglou08]. La descripción funcional detalla las características operacionales que definen el comportamiento general de un servicio web. La descripción no funcional se concentra en sus atributos de calidad (Quality of Service - QoS), tales como tiempo de respuesta, seguridad, autorización, autenticación, etc. Las descripciones no funcionales fuerzan al solicitante del servicio a especificar, en tiempo de ejecución, los requerimientos no funcionales que pueden influir en la elección de un servicio web ofrecido por un proveedor.

Las características de calidad de un servicio web se refieren a la habilidad del servicio de responder a las invocaciones y llevarlas a cabo en consonancia con las expectativas del proveedor y de los clientes. Diversos factores de calidad que reflejan las expectativas del cliente, como la disponibilidad, conectividad y alta respuesta, se vuelven clave para mantener un negocio competitivo y viable. Las características de calidad son criterios substanciales que determinan la usabilidad y utilidad del servicio, ambos de gran influencia en la popularidad de un servicio web particular, y que resultan un importante punto de diferenciación entre los proveedores de servicios web.

La prestación de un servicio web en Internet es un desafío crítico por su naturaleza dinámica e impredecible. Las aplicaciones con características y requerimientos bien diferenciados compiten por toda clase de recursos en la web. Los cambios en los patrones de tráfico, la seguridad en las transacciones de negocio críticas, y los efectos de una falla en la infraestructura disminuyen la performance de los protocolos web. A menudo, las características de calidad no resueltas causan que las aplicaciones con transacciones críticas sufran inaceptables niveles de degradación en su performance. Por ello surge la necesidad de definir estándares sobre las características de calidad en Internet.

B.2 Características de calidad de los servicios web

Tradicionalmente, las características de calidad se miden por el grado en que las aplicaciones, sistemas, redes y todo otro elemento en la infraestructura de las tecnologías de la información soportan la disponibilidad de los servicios al nivel de la performance requerida en todos los accesos y bajo todas las condiciones. En el contexto de los servicios web, las características de calidad se pueden ver como la capacidad de proveer garantía sobre un conjunto de características cuantitativas. Estas se pueden definir sobre la base de propiedades importantes del servicio web, tanto funcionales como no funcionales.

Para proveer al usuario un servicio web que se adecue a sus requerimientos no funcionales, primero es necesario identificar todas las características posibles, es decir, el conjunto de atributos no funcionales que pueden influenciar la calidad del servicio [W3C03-QoS]. Existen muchos aspectos de las características de calidad importantes para los servicios web. Se comenzará organizando estas características en categorías, donde cada una agrupa un conjunto de parámetros o medidas relacionadas [Ran03].

B.2.1 Características relacionadas al tiempo de ejecución

Performance

Es la velocidad para completar un servicio requerido. Se puede medir en términos de *rendimiento (throughput), tiempo de respuesta*, *latencia, tiempo de ejecución, tiempo de transacción,* etc.

- Rendimiento (throughput): representa el número de requerimientos de servicios web atendidos en un período de tiempo dado.
- Tiempo de Respuesta: representa el tiempo esperado entre que se envía un requerimiento y se recibe una respuesta. Es el tiempo requerido para completar un servicio web.
- Latencia: representa la longitud de tiempo entre que un servicio web envía un requerimiento y obtiene una repuesta.
- Tiempo de ejecución: es el tiempo que necesita un servicio web para procesar su secuencia de actividades.
- Tiempo de transacción: representa el tiempo que transcurre mientras el servicio web está completando una transacción.

En general, un servicio web de alta calidad debería proporcionar alto rendimiento, tiempo de respuesta más rápido, baja latencia, bajo tiempo de ejecución y tiempo de transacción más rápido. En el campo de los servicios web, los mensajes SOAP representan el recurso crítico consumiendo tiempo y teniendo la mayoría de los impactos significativos sobre el rendimiento subyacente.

Fiabilidad

La fiabilidad (reliability) representa la habilidad de un servicio para funcionar correcta y consistentemente y proveer la misma calidad a pesar de las fallas en la red o el sistema. Los servicios web deberían proveer alta fiabilidad. Normalmente, la fiabilidad se expresa en términos del número de fallas transaccionales por día, semana, mes o año. También está relacionada a la entrega de forma segura y ordenada de los mensajes recibidos y transmitidos por los solicitantes y proveedores de los servicios.

Escalabilidad

La escalabilidad (scalability) se refiere a la habilidad de atender consistentemente un requerimiento a pesar de las variaciones en el volumen de requerimientos. Representa la habilidad de incrementar la capacidad de computación de los sistemas de los proveedores de los servicios y la habilidad de procesar mayor cantidad de requerimientos de los usuarios, operaciones o transacciones en un intervalo de tiempo dado. Esta medida está relacionada a la performance. Los servicios web deberían proveer alta escalabilidad. Puede conseguirse una alta accesibilidad a un servicio web construyendo sistemas altamente escalables, en términos del número de operaciones o transacciones soportadas.

Capacidad

La capacidad (capacity) es el límite de requerimientos simultáneos que permita garantizar la performance. Los servicios web deberían proveer la capacidad requerida.

Robustez

Es el grado en el cual un servicio puede funcionar correctamente en presencia de entradas inválidas, incompletas o conflictivas.

Los servicios web deberían proveer alta robustez, lo cual implica que deberían funcionar aún si existen parámetros incompletos en el requerimiento del servicio.

Manejo de excepciones

Dado que no es posible para un diseñador de servicios especificar todas las posibles salidas y alternativas al ejecutar un servicio web, pueden surgir excepciones. El manejo de excepciones se refiere al manejo de este tipo de irregularidades.
Los servicios web deberían contar con la funcionalidad de manejo de excepciones.

Precisión

La precisión (accuracy) es una medida del radio de error producido por un servicio. Los servicios web deberían proveer alta robustez precisión, lo cual indica que el número de errores que genera en un intervalo de tiempo debería ser mínimo.

Accesibilidad

La accesibilidad indica si el servicio web es capaz de cumplir los requerimientos del cliente. Los servicios web deberían proveer alta accesibilidad.

Disponibilidad

La disponibilidad (availability) representa la probabilidad de que el servicio web este disponible para su consumo inmediato. Esta disponibilidad es la probabilidad de que el sistema esté activo y se relaciona con la fiabilidad. También, el tiempo consumido en la reparación (TTR) de un servicio web está asociado a esta característica. Grandes valores indican que el servicio web está siempre disponible para su uso, mientras que valores más bajos indican la incertidumbre sobre si el servicio web estará disponible o no en un momento particular.

Interoperabilidad

Los servicios web deberían poder interoperar en los diferentes ambientes de desarrollo usados para implementarlos, de manera tal que los desarrolladores de dichos servicios no deban tener en cuenta el lenguaje de programación o sistema operativo sobre el cual se aloja el servicio.

Reputación

La reputación (reputation score) es una medida de la reputación otorgada al servicio por el usuario final.

B.2.2 Características relacionadas al soporte de transacciones

Hay diversos casos en donde los servicios web requieren comportamiento transaccional y propagación de contexto. El hecho de que un servicio web particular requiera comportamiento transaccional indica que los proveedores del servicio deben mantener esta propiedad.

Integridad

La integridad transaccional se refiere a un procedimiento o conjunto de procedimientos, los cuales deben garantizar que preservan la integridad de la base de datos en una transacción. Las transacciones se pueden agrupar en una unidad para garantizar la integridad de los datos operados por esas transacciones. Esta unidad puede resultar satisfactoria, si todas sus transacciones terminan adecuadamente (terminan con un commit) o todas vuelven a su estado original (terminan con un rollback) en el caso de una falla en la transacción. Esto se describe a través de las propiedades conocidas como ACID: atomicidad (se ejecuta enteramente o no se ejecuta), consistencia (mantiene la integridad de los datos), aislamiento (transacciones individuales se ejecutan aún si no hay otras transacciones presentes), y durabilidad (los resultados son persistentes).

La capacidad de realizar el *commit* en dos fases es el mecanismo que garantiza las propiedades ACID en el caso de transacciones distribuidas ejecutándose sobre sistemas estrechamente relacionados como si se tratara de una sola transacción. Esto es más dificultoso en el ambiente de los servicios web, porque las transacciones pueden involucrar más de un socio del negocio, con la posibilidad de que las transacciones duren un largo período de tiempo (horas o días). Aunque la integridad de las transacciones aún se describe por medio de las propiedades ACID, es mucho más difícil conseguirlo en este caso, y se requieren mecanismos diferentes a los tradicionales.

B.2.3 Características relacionadas al costo y gestión de la configuración

Conformidad con las leyes

La Conformidad con las leyes (en inglés regulatory) es una medida de cómo se ajusta el servicio web a la reglamentación vigente.

Conformidad con estándares

Esta medida describe la concordancia de un servicio web con los estándares. Es necesaria una adhesión estricta a las versiones correctas de los estándares (por ejemplo, WSDL versión 2.0) por parte de los proveedores del servicio para permitir una invocación apropiada de los servicios por los solicitantes.

Estabilidad

Es una medida de la frecuencia de cambios relacionados al servicio en términos de su interfaz y/o implementación.

Costo

Es una medida del costo involucrado en el requerimiento del servicio.

Completitud

Es una medida de la diferencia entre el conjunto especificado de características y el conjunto implementado de características, es decir, cuántas de las características especificadas están disponibles.

B.2.4 Características relacionadas a la seguridad

Los servicios web deben proveer la seguridad requerida para su uso. Con el incremento del uso de servicios web, existe una creciente preocupación por la seguridad. El proveedor del servicio web puede aplicar diferentes enfoques y niveles de seguridad dependiendo del solicitante del servicio.

La seguridad del servicio web significa proveer autenticación, autorización, confidencialidad, trazabilidad/auditoría, encriptación de datos y no repudio. La seguridad tiene una importancia adicional dado que la invocación de los servicios web se da sobre Internet. Los proveedores del servicio deben mantener el nivel de seguridad del servicio web.

Autenticación

Los usuarios u otros servicios web que pueden acceder al servicio deberían ser autenticados. La autenticación es una medida de cómo autentica el servicio a estos usuarios y otros servicios web que lo acceden.

Autorización

Los usuarios u otros servicios web deberían estar autorizados, de manera tal que sólo ellos puedan acceder a los servicios protegidos. La autorización es una medida que indica cómo el servicio autoriza a estos usuarios.

Confidencialidad

Los datos se deben tratar de tal manera que sólo los usuarios autorizados puedan accederlos o modificarlos. La confidencialidad es una medida de cómo manipula los datos el servicio web, de manera tal que sólo estos usuarios autorizados puedan accederlos y/o modificarlos.

Rendición de cuentas

Es una medida que indica si el proveedor puede el proveedor rendir cuentas por sus servicios ofrecidos.

Trazabilidad y Auditoría

Es una medida que indica si es posible realizar una traza de la historia de los servicios requeridos.

Encriptación de datos

Es una medida que indica si el servicio web posee mecanismos de seguridad aplicados a los mensajes de entrada y salida cuantificados en términos de un protocolo de encriptación y tipos de claves usadas para la encriptación de mensaje.

No repudio

Esta medida indica que no se puede negar un requerimiento de un servicio o dato después de haberse aceptado. El proveedor del servicio debe asegurar estos requerimientos de seguridad.

B.3 Enfoques posibles para el soporte de las características de calidad

La W3C propone tres enfoques para permitir a los servicios web soportar las características de calidad QoS [W3C03-QoS], los mismos se describen a continuación.

B.3.1 Una extensión a UDDI

El registro UDDI se puede extender para soportar las características de calidad de los servicios web. Por ejemplo, se puede extender para permitir publicar los servicios web junto con sus QoS en este registro UDDI. Para conseguir esto, la estructura de datos UDDI existente debe extenderse para describir la información QoS del servicio web. Existen propuestas para proveer la información QoS en el registro UDDI [Ran03].

El ejemplo de la Figura B.1 describe el elemento qosInfo (Información QoS), definido como un sub elemento del elemento businessService.

```
<tModel tModelKey="uuid:1C620754-09E4-4930-AA19-709C62E52166">
  <name>uudi-org:qosInfo</name>
  <description xml:lang="en">Quality of Service Information</description>
  <overviewDoc>
    <description xml:lang="en"></description>
    <overviewURL>http://www.uddi.org/specification.html</overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference
      keyName="uddi-org:types" keyValue="categorization"
      tModelKey="uddi:1D3FCD00-0DA6-4479-ADFE-B10950C74F62"/>
  </categoryBag>
</tModel>
```

Figura B.1. Un ejemplo de tModel

B.3.2 Una extensión a SOAP

El protocolo SOAP existente en la actualidad podría extenderse para permitir encontrar los servicios web que satisfacen requerimientos QoS específicos. Esto se puede conseguir extendiendo el formato de los mensajes SOAP. Una forma sería usar una consulta para encontrar los servicios web deseados. En respuesta a la consulta, el resultado del mensaje, el cual contenga un servicio web adecuado, debería se generado. El ejemplo de la Figura B.2 muestra cómo encontrar servicios web adecuados usando un mensaje de consulta [Ran03].

```
<?xml version="1.0" encoding="UTF-8"?>
  <envelop xmlns= "http://schmas.xmlsoap.org/soap/envelop/">
    <body>
      <find_service businessKey="*" generic="1.0" xmlns="urn:uddi-org:api"
         maxRows="5">
         <name>car reservation</name>
         <qosInfo>
           <availability> 0.8 </availability>
         </qosInfo>
      </find_service>
    </body>
  </enveolop>
```

Figura B.2. Un ejemplo de un mensaje de consulta

En el mismo el mensaje de consulta se usa para encontrar un servicio web de reserva de un automóvil cuya disponibilidad es de 0,9. La Figura B.3 muestra el contenido del mensaje de resultado, generado en respuesta al mensaje de consulta.

```
<?xml version="1.0" encoding="UTF-8"?>
<envelop xmlns="http://schemas.xmlsoap.org/soap/envelop">
  <body>
    <serviceList generic="1.0" xmlns="urn:uddi-org:api"
        operator="rental.com/car/services/uddi" truncated="false">
      <serviceInfos>
         <serviceInfo
           serviceKey="F4D95E5D-F37A-4BDC-8E95-BFD45671A0F"
           businessKey="8CF8C1E3-4C39-43E1-8B6B-0137ED6BE63F">
           <name>car reservation</name>
           <qosInfo>
              <availability> 0.92 </availability>
           </qosInfo>
         </serviceInfo>
      </serviceInfos>
    </serviceList>
  </body>
</envelop>
```

Figura B.3. Un ejemplo de un mensaje de resultado

B.3.3 Certificación del servicio con características de calidad

Podría ser necesario certificar el mensaje resultado desde el registro UDDI en respuesta a una consulta sobre las características de calidad del servicio web. Para conseguir esto, puede definirse un protocolo de certificación eficiente, el cual describa cómo registrar y certificar la información QoS.

B.4 Descubrimiento automático de servicios web con QoS

Existen dos problemas principales relacionados al descubrimiento automático de los servicios web con características de calidad. El primero involucra la especificación de la información QoS, al momento de diseñar el servicio. Se refiere a cómo se debería expresar y almacenar la información QoS. Resulta obvio que se hace necesario contar con un formato estándar compartido por proveedores y consumidores de servicios web, para que esta información se pueda intercambiar e interpretar correctamente. El segundo problema está relacionado a la correspondencia entre las características requeridas por el cliente y las ofrecidas por el proveedor. A continuación se presentan algunas propuestas para solucionar estos problemas.

B.4.1 Los acuerdos SLA

Una de las propuestas para solucionar los problemas antes mencionados son los acuerdos SLA (Service Level Agreements). Los acuerdos SLA son un mecanismo que permite asegurar que el proveedor del servicio web elegido garantiza un cierto nivel de calidad. Un SLA es un acuerdo formal de contrato entre el proveedor y el cliente, formalizando lo detalles del servicio web (contenido, precio, proceso de entrega, criterio de aceptación y calidad, sanciones, etc.), por lo general en términos medibles, de forma tal de llegar a un mutuo entendimiento y conocimiento de las expectativas de ambos, proveedor y cliente. Un SLA es básicamente una garantía QoS respaldada por cobro a los usuarios y otros mecanismos diseñados para compensar a los usuarios de los servicios y para obligar a las organizaciones a cumplir los compromisos SLA. Un SLA es un instrumento muy importante y ampliamente usado en el mantenimiento de las relaciones de prestación del servicio cuando los proveedores y clientes lo están usando.

Un SLA puede contener las siguientes partes:

- Propósito: Este campo describe las razones para la creación del SLA.
- Partes: Este campo describe los participantes involucrados en el SLA y sus respectivos roles, por ejemplo, proveedor del servicio y consumidor del servicio (cliente).

- Periodo de validación: Este campo define el periodo de tiempo que el SLA cubrirá. Esta delimitado por el momento de comienzo y finalización de los términos del acuerdo.
- Alcance: Este campo define los servicios cubiertos en el acuerdo.
- Restricciones: Este campo define las medidas necesarias que deben tomarse para proveer los niveles de calidad requeridos del servicio.
- Objetivos del nivel de servicio: Este campo define los niveles del servicio que ambos, proveedor y cliente, acuerdan y usualmente incluye un conjunto de indicadores del servicio, como disponibilidad, performance, y fiabilidad. Cada uno de estos aspectos del nivel del servicio web tendrá que lograr un nivel meta.
- Sanciones: Este campo define qué sanciones deberían aplicarse en el caso de que el proveedor del servicio sea incapaz de alcanzar los objetivos especificados en el SLA.
- Servicios opcionales: Este campo especifica cualquier servicio que no es requerido normalmente por el usuario, pero podría ser requerido excepcionalmente.
- Términos de exclusión: especifican lo que no es cubierto por el SLA.
- Administración: Este campo describe los procesos y objetivos medibles en un SLA y define la autoridad organizacional que los supervisará.

Un SLA puede ser estático o dinámico. Un SLA estático es un SLA que generalmente sigue siendo el mismo en muchos intervalos de tiempo del servicio. Los intervalos de tiempo del servicio pueden ser meses calendarios o pueden ser una transacción o cualquier otro período de tiempo relevante y medible para otros procesos.
Son usados para evaluar las características de calidad QoS y se acuerdan entre el proveedor y el cliente. Un SLA dinámico es un SLA que generalmente cambia de un período del servicio a otro período del servicio, para acomodarse a los cambios en la provisión del servicio.

Para incorporar un SLA a un servicio web, hay métricas QoS que son evaluadas en un intervalo de tiempo para definir el conjunto de objetivos que deberían ser empleados. Las medidas de los niveles QoS en un SLA implicarán, en última instancia, la localización del servicio web a través de múltiples infraestructuras (geográficas, tecnológicas, de la aplicación, y el proveedor). En un escenario típico, cada servicio web puede interactuar con múltiples servicios web, cambiando los roles con que participa. Cada una de estas

interacciones debería ser gobernada por un SLA. Las métricas impuestas por los SLAs deberían correlacionarse con los objetivos del servicio ofrecido.

Para conseguir medir un SLA de una manera organizada, es conveniente agrupar las características de calidad de un servicio web bajo las siguientes tres categorías:

1. Performance y capacidad: Esta categoría considera características tales como volúmenes de transacción, tasas de rendimiento, tamaño del sistema, niveles de utilización, qué sistema subyacente ha sido diseñado y probado para hacer frente a los picos de carga, y, finalmente, qué tan importantes son los tiempos de requerimiento/respuesta.
2. Disponibilidad: Esta categoría considera características tales como tiempo medio de falla para todas las partes del sistema, mecanismos de recuperación de desastres, tiempo medio de recuperación, si el negocio puede tolerar caídas del servicio web y cuanto le cuesta, etc.
3. Seguridad/Privacidad: Esta categoría considera características tales como responder a intento sistemáticos de irrumpir en un sistema, privacidad, mecanismos de autenticación/autorización del proveedor, etc.

B.4.2 La descripción no funcional en WSDL

Con respecto a la especificación de la información QoS al momento de diseñar el servicio web, como se expresó anteriormente, el lenguaje utilizado en la actualidad para describir un servicio es WSDL. Pero, si bien WSDL especifica la sintaxis del servicio, no especifica ningún aspecto no funcional del servicio. La plataforma de los servicios web debería ser capaz de soportar una multitud de diferentes tipos de aplicaciones con diferentes requerimientos de calidad. Así, las características no funcionales de los servicios web también deben ser descriptas de alguna manera. De hecho, los programadores y las aplicaciones deben ser capaces de entender las características de calidad de los servicios web para poder invocarlos e interactuar con ellos.

Habilitar las características QoS en un servicio web requiere de un lenguaje separado para describir las características no funcionales de los servicios web. Actualmente, el enfoque más utilizado para ello es la combinación de dos especificaciones: *WS-Policy* y *WS-Policy Attachment* [Papazoglou08]**,** [ZTP05]:

- *WS-Policy:* provee un lenguaje flexible y extensible, basado en XML, para expresar las capacidades, requerimientos, y propiedades funcionales y no funcionales del servicio web, de una manera declarativa. WS-Policy define un framework y un modelo para expresar estas propiedades como políticas.
- *WS-Policy Attachment:* indica tres mecanismos de asociación específicos para usar expresiones de políticas con las tecnologías de servicios web existentes. Más específicamente, especifica cómo asociar una expresión de política con una definición WSDL, en las entidades UDDI.

Estas especificaciones se basan en el concepto de políticas de servicios web. Las políticas agrupan características adicionales de los servicios relacionadas a los requerimientos, capacidades y preferencias que hacen a la descripción de la configuración del servicio. Ejemplos típicos son las cuestiones de seguridad (incluyendo autorización y autenticación), mensajería fiable, comportamiento transaccional, características de calidad, privacidad, y opciones del servicio específicas de la aplicación o capacidades y restricciones específicas de un dominio particular. Las políticas de los servicios web son expresiones declarativas de la política requerida para la interacción con un servicio web particular. Describen en forma de aserciones una o más características que el proveedor del servicio web utiliza para darle instrucciones al consumidor del servicio.

Un framework estándar de políticas es un marco de trabajo común para la expresión, intercambio y procesamiento de las políticas que gobiernan las interacciones entre proveedores y consumidores de servicios web. El *framework WS-Policy* provee una capa adicional de descripción de los servicios y ofrece un lenguaje declarativo para expresar y programar las políticas. Empleando este lenguaje se pueden tener en cuenta las características de seguridad (incluyendo autenticación y autorización), el comportamiento transaccional, los niveles de QoS y calidad de protección ofrecidos por el proveedor, las políticas de privacidad observadas por el proveedor, las opciones específicas de la aplicación, y las capacidades y restricciones específicas de un dominio particular. Cuando se consideran las características QoS en los servicios web, la especificación de la interfaz necesita ser extendida con sentencias asociadas a la misma de manera completa o a algún atributo u operación en particular.

ANEXO C.

Reglas de Transformación de Grafos

Este anexo detalla la tecnología de transformación de grafos, ampliamente utilizada para especificar cómo deben ser construidos y cómo deben evolucionar los modelos de software. Se presentan los conceptos básicos de grafos: definición de grafo, gramáticas de grafos y reglas de transformación de grafos, y su aplicación el modelado de sistemas de software orientado a objetos.

La estructura del anexo es la siguiente. La sección 1 introduce las tecnologías de grafos y su utilidad en el campo de la ingeniería de software. La sección 2 describe las gramáticas de grafos y las reglas de transformación de grafos, detallando las áreas de computación donde tiene amplia aplicación este tipo de tecnología. La sección 3 describe los diferentes enfoques en la aplicación de las reglas de transformación de grafos, orientados a resolver los problemas que surgen al aplicar las reglas. En particular, en la sección 4 se detallan los dos enfoques algebraicos denominados *simple- pushout (SPO)* y *doble- pushout (DPO)*. Finalmente, las secciones 5 y 6 introducen la utilidad de la técnica de transformación de grafos en el modelado orientado a objetos, detallando la aplicación de los conceptos previamente descriptos al modelado y meta modelado orientado a objetos.

C.1 Introducción

Los grafos y diagramas son muy usados para describir estructuras y sistemas complejos y modelar conceptos e ideas de una forma directa e intuitiva. En particular, proveen una simple y poderosa herramienta para modelar una gran variedad de problemas típicos de la ingeniería de software. Los artefactos producidos para conceptualizar un sistema son llamados modelos, y los diagramas son usados para visualizar su estructura compleja de una forma más intuitiva y natural. A través de los grafos se pueden definir las estructuras de esos modelos, y entonces la transformación de grafos puede ser explorada para especificar cómo deben ser construidos y cómo deben evolucionar dichos modelos.

La definición e implementación de las técnicas de modelado visual presentan problemas cuando son comparadas con los lenguajes de especificación formal y de programación:

- El primer problema es que muchas de las técnicas para la definición de lenguajes, tales como la denotacional o la operacional, están intrínsecamente basadas en términos (árboles de sintaxis abstracta) de acuerdo a la representación de la estructura del lenguaje. Esto resulta del uso de las gramáticas libres de contexto (ej. BNF) para la definición de su sintaxis textual. Sin embargo, la mayoría de los lenguajes gráficos tienen una estructura de grafo, y tales técnicas no son totalmente aplicables.
- El segundo problema es el número y la diversidad de lenguajes de modelado y dialectos que están actualmente en uso, además de los todos los nuevos que surgen por necesidades particulares de áreas específicas. Para proveer definiciones e implementaciones de lenguajes para esas notaciones dentro de restricciones de recursos razonables, se requieren técnicas de meta-nivel y herramientas para generar las implementaciones de los lenguajes a partir de especificaciones de alto nivel.
- El tercer problema es la consistencia de los modelos que se componen de submodelos interrelacionados, representando diferentes aspectos y distintos niveles de abstracción. Generalmente, un modelo inconsistente no tiene una implementación correcta porque los requerimientos expresados por los diferentes submodelos están en conflicto. Para atacar este problema es necesario comprender la relación entre los

submodelos a un nivel semántico. En este sentido, además de técnicas y herramientas que definan la semántica, se requiere de dominios semánticos que capturen lo esencial de los sistemas actuales, incluyendo aspectos como orientación a objetos, arquitectura de software, concurrencia, distribución, movilidad, etc.

Las técnicas de transformación de grafos pueden contribuir a resolver el tipo de problemas planteados anteriormente. Las gramáticas de grafos y la transformación de grafos se fundan en la modificación de grafos basada en reglas, donde cada aplicación de una regla a un grafo da lugar a un paso en la transformación. Las gramáticas de grafos pueden usarse para generar lenguajes de grafos, similares a las gramáticas de Chomsky en una teoría de lenguaje formal. Además, los grafos se pueden usar para modelar los cambios de estado de los sistemas. En el desarrollo dirigido por modelos, la transformación de modelos es fundamental, por ello, existe la necesidad de un lenguaje adecuado para la manipulación de los modelos.

C.2 Gramáticas de Grafos y Transformación de Grafos

Un grafo consiste de un conjunto de vértices *V* y un conjunto de arcos *E* tal que cada arco *e* en *E* posee como origen y destino los vértices *s(e)* y *t(e)* en *V*, respectivamente. Una gramática de grafos está formada por un conjunto de reglas más un grafo inicial. Las reglas están formadas por una parte izquierda y una parte derecha, ambas conteniendo un grafo. Por su parte una transformación de grafos es una modificación de grafos basada en reglas, como se detalla más adelante.

Las gramáticas de grafos y la transformación de grafos son una forma muy natural de explicar situaciones complejas a un nivel intuitivo [BH02]. Por ello son usadas en la ciencia de la computación en todas partes, por ejemplo, para modelar diagramas de control de flujo y de datos, diagramas de entidad-relación y diagramas del Lenguaje de Modelado Unificado (Unified Modeling Language - UML) [UML], también en las Redes de Petri, para la visualización de arquitecturas de software y hardware, diagramas de evolución de procesos no determinísticos, y para muchos otros propósitos. Al igual que los *tokens* en una Red de Petri, la transformación de grafos permite modelar los aspectos dinámicos, dado que pueden describir gráficamente la evolución de las estructuras. De esta manera, la transformación de grafos se ha vuelto atractiva como un paradigma de

modelado y programación para desarrollar software estructurado complejo y sus interfaces gráficas.

La transformación de grafos tiene tres orígenes fundamentales [EEPT06]:

- Las gramáticas de Chomsky sobre cadenas
- La reescritura de términos
- Las Redes de Petri para el modelado visual

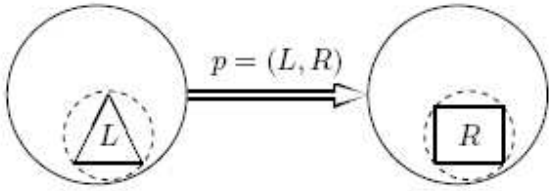


Figura C.1. Modificación de grafos basada en reglas

La Figura C.1 muestra la idea de la transformación de grafos basada en reglas. Una regla de transformación de grafos o producción p = (L, R) contiene un par de grafos (L, R), denominados la parte izquierda L y la parte derecha R. Aplicar la regla p = (L, R) significa encontrar una ocurrencia de L en el grafo anfitrión (G) y reemplazarla por R, obteniendo el grafo destino (H). El problema principal de la técnica es cómo borrar L y cómo conectar R con el contexto en el grafo destino. De hecho, existen diferentes enfoques sobre cómo manejar estos problemas, que se resumen en la sección 6.3.

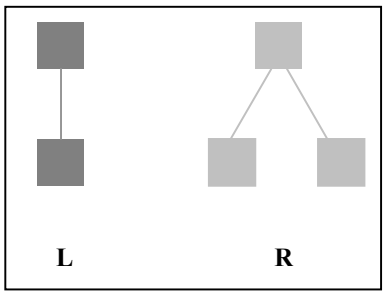


Figura C.2. Grafos L y R, la parte izquierda y parte derecha de la regla

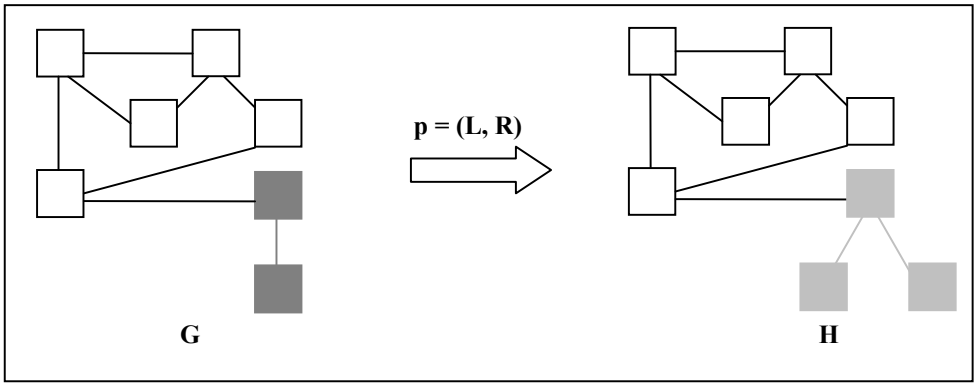


Figura C.3. Aplicación de la regla a un grafo anfitrión

El concepto de computación por transformación de grafos es fundamental en las siguientes áreas [EEPT06]:

- *Especificación y modelado visual*: los grafos son un medio bien conocido, entendido, y frecuentemente usado para representar los estados de los sistemas. Los diagramas de clases y objetos, grafos, diagramas de entidad-relación y Redes de Petri son representaciones gráficas comunes de los estados de los sistemas. Las reglas han demostrado ser extremadamente útiles para describir las computaciones por transformaciones de estados. En el modelado orientado a objetos, los grafos ocurren a dos niveles: a nivel de tipos (definidos en base a los diagramas de clases) y a nivel de instancia (dado por todos los diagramas de objetos válidos). El modelado por transformación de grafos resulta, por un lado, muy natural para usarlo como una representación visual, y por el otro, muy preciso, mostrando todo su fundamento formal. Así, la transformación de grafos se puede usar en técnicas de especificación formal para sistemas basados en estados.
- *Transformación de modelos*: en los últimos años ha evolucionado el proceso de desarrollo de software basado en modelos. En esta área los modelos ya no son una documentación pasiva, sino que pueden usarse para la generación de código, el análisis y la simulación. Una cuestión importante es cómo especificar tales transformaciones de modelos. Comenzando desde un modelo visual, como los discutidos arriba, la transformación de grafos resulta una elección natural. Sobre la base de la estructura de tales modelos visuales, es decir, los grafos de sintaxis abstracta, se define la transformación de modelos. Debido al fundamento formal de esta técnica, se puede chequear la correctitud de las transformaciones de modelos. Más precisamente, la correctitud se puede formular sobre una sólida base

matemática, y existe una buena chance de verificar la correctitud usando teoría de transformación de grafos.

- *Concurrencia y distribución*: cuando se utiliza la transformación de grafos para describir sistemas concurrentes, los grafos se usan para describir las estructuras estáticas de los sistemas. El comportamiento del sistema, expresado por los cambios de estado, se modela por medio de la manipulación de grafos basada en reglas, es decir, la transformación de grafos. Las reglas describen las pre y poscondiciones de un paso de transformación. En un sistema de transformación de grafos puro, el orden de los pasos se determinan solamente por la dependencia causal de las acciones, es decir, se pueden ejecutar reglas independientes en un orden arbitrario. El concepto de regla en la transformación de grafos provee un concepto claro para definir el comportamiento de los sistemas. En particular, para modelar la concurrencia de las acciones, las reglas de grafos proveen un medio adecuado, porque permiten explicar todas las interdependencias estructurales. En una ejecución secuencial, las transformaciones paralelas deben modelarse por medio de la intercalación arbitraria de sus acciones atómicas. La ejecución simultánea de acciones se puede modelar si la regla paralela se compone de esas acciones. La transformación de grafos distribuida y paralela permite la aplicación estructurada de las reglas, tanto en la dimensión temporal y como la espacial. Los grafos se dividen en grafos locales, y luego de las transformaciones, se vuelven a unir en un grafo global.
- *Desarrollo de software*: en el desarrollo de software, una gran variedad de estructuras diferentes ocurren a diferentes niveles, las cuales se pueden manejar por grafos. Se distinguen las estructuras arquitecturales y técnicas de las configuraciones administrativas y la integración de documentos. Toda esta información estructural evoluciona, es decir, cambia durante el proceso de desarrollo de software. Esto incluye la edición de documentos, ejecución de operaciones, modificación de programas, análisis, control de configuración y revisión, etc. Las técnicas de transformación de grafos son usadas para describir esta evolución estructural basada en reglas.

C.3 Diferentes enfoques de la transformación de Grafos

Desde un punto de vista operacional, una transformación de grafos de G a H, $G \Rightarrow H$, contiene los siguientes pasos principales [EEPT06]:

1. Elegir una producción p: L ⇒ R con una ocurrencia de L en G.
2. Chequear las condiciones de aplicación de la producción.
3. Borrar de G la parte de L que está en R. Si quedan arcos sueltos[1] después de borrar L, hay dos opciones: o la producción no se aplica o los arcos sueltos también se borran. El grafo obtenido se denomina D.
4. Agregar la parte derecha R al grafo D en la parte de L que tiene una imagen en D. La parte de R que no procede de L se agrega de forma disjunta a D, dando como resultado el grafo E.
5. Si la producción p contiene una relación adicional, entonces se integra a la parte derecha R en el grafo G de acuerdo a esta relación. El resultado es el grafo H.

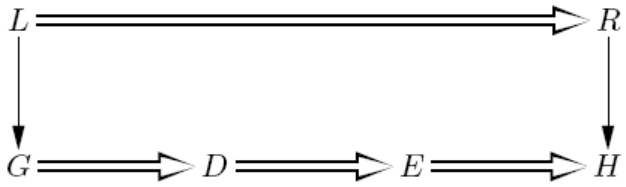


Figura C.4. La transformación de grafos desde un punto de vista operacional

Los sistemas de transformación de grafos pueden dar lugar a dos clases de no determinismo. Primero, varias producciones pueden ser aplicables y debe elegirse arbitrariamente una de ellas. Segundo, dada una producción, varias ocurrencias pueden aplicarse y tiene que elegirse una de ellas. Existen diferentes técnicas para restringir ambas clases de elecciones, como:

- *Enfoque de reemplazo de nodos etiquetados* (node label replacement approach): el cual permite que un nodo simple, como el lado izquierdo L, sea reemplazado por un grafo arbitrario R. La conexión de R con el contexto se determina por una relación embebida (*embedding relation*) que depende de los nodos etiquetados.
- *Enfoque de reemplazo de hiper arcos* (hyperedge replacement approach): que tiene como la parte izquierda L un hiper arco etiquetado, el cual se reemplaza por un hiper grafo R donde los nodos se corresponden con los nodos de L. El pegado (gluing) de R al contexto da lugar a un grafo destino sin la necesidad de usar una relación embebida adicional.

[1] Arcos que de aplicarse la regla o producción quedarían sin su nodo fuente o destino.

- *Enfoque algebraico* (algebraic approach): el mismo se basa en las construcciones *pushout*, donde los *pushouts* se usan para modelar el pegado de los grafos. De hecho, existen dos variantes de este enfoque, el enfoque algebraico *simple- pushout (SPO)* y el enfoque algebraico *doble- pushout (DPO)*. En ambos casos no existe una relación embebida adicional.
- *Enfoque lógico*: el cual permite que las transformaciones y las propiedades de los grafos se expresen en una lógica de segundo orden monádica.
- *Teoría de dos estructuras* (theory of 2-structures): es un framework para la descomposición y transformación de grafos.
- *Enfoque de reemplazo de grafos programados* (programmed graph replacement approach): el cual combina los aspectos del pegado (gluing) y el embedding de la transformación de grafos. Además, utiliza programas para controlar el no determinismo en la elección de la regla a aplicar.

En particular, el borrado de los elementos en un grafo, es decir, el paso 3 dentro de la transformación de grafos desde el punto de vista operacional, puede causar algunos problemas, por el hecho de que se debe asegurar que la estructura resultante siga siendo un grafo válido. En el marco del *Enfoque Algebraico en la Transformación de Grafos* existen dos soluciones a estos problemas, una es denominada radical y la otra conservativa. La primera le da prioridad al borrado, borrando los arcos sueltos que resultan del borrado de vértices en un grafo. Esto puede producir efectos no deseados y requiere, por lo tanto, de controles adicionales para restringir las aplicaciones posibles de la regla. Una alternativa más segura consiste en formular condiciones de aplicación estándares, las cuales excluyan las situaciones descriptas anteriormente como transformaciones válidas. Dichas condiciones de aplicación deben ser satisfechas por el grafo anfitrión para poder aplicarse la regla. Esto se consigue en el enfoque algebraico *doble- pushout* a través de las denominadas *condiciones de pegado* (gluing conditions). En la siguiente sección se presenta una breve descripción del Enfoque Algebraico para la Transformación de Grafos y sus dos alternativas para solucionar los problemas antes mencionados, el enfoque algebraico *simple- pushout (SPO)* y el enfoque algebraico *doble-pushout (DPO)*.

C.4 El enfoque algebraico en la Transformación de Grafos

El enfoque algebraico se basa en las construcciones *pushout*, donde los *pushouts* se usan para modelar la unión o pegado (*gluing*) de lo grafos. En este enfoque se usan dos construcciones de unión para modelar un paso de transformación de grafo. Por esta razón este enfoque se conoce también como enfoque *double pushout* (DPO), en contraste con el enfoque *simple pushout* (SPO).

En este enfoque un grafo se ve como una estructura algebraica [Lara04] [DBRG00], de la forma:

$G = (V,E,src,tar)$

$src: E \rightarrow V$

$tar: E \rightarrow V$

El morfismo entre los grafos establece que:

$h = (h_V: V_1 \rightarrow V_2, h_E: E_1 \rightarrow E_2): G_1 \rightarrow G_2$

La preservación de la estructura ocurre si se cumple la siguiente relación:

$h_V(src_1(e)) = src_2(h_E(e))$ y $h_V(tar_1(e)) = tar_2(h_E(e))$

es decir:

$h_V \circ src_1 = src_2 \circ h_E$ y $h_V \circ tar_1 = tar_2 \circ h_E$

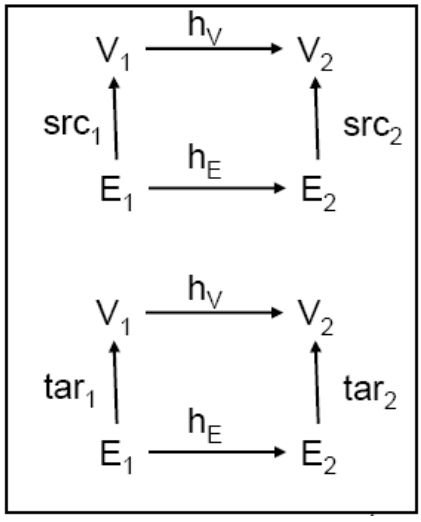


Figura C.5. Grafos como estructura algebraica

A continuación se describen brevemente los dos enfoques.

C.4.1 El enfoque DPO

En este enfoque una producción tiene la forma p: L←K→R [EEPT06]. Una regla de transformación de grafos o producción p consiste de tres grafos L, K y R. La producción es aplicable a un grafo G si G contiene un subgrafo que es una imagen de L. Una producción p es aplicable a un grafo G si L es un subgrafo de G. El grafo L constituye la parte izquierda de la producción, y formula las condiciones bajo las cuales resulta aplicable la producción. El grafo de contexto K, que generalmente es un subgrafo de L y de R (K = L $\cap$ R), describe los subobjetos de la parte izquierda que se deben preservar en el proceso de aplicación de la producción. Por lo tanto, la diferencia entre L y K: L \ K, contiene a todos aquellos subobjetos que se deben eliminar al aplicar la producción. Análogamente, la diferencia R \ K contiene a todos aquellos subobjetos que se deben agregar al aplicar la producción. Este grafo intermedio K describe el contexto en el cual se integran los subobjetos agregados.

Una transformación de grafos directa con una producción p se define encontrando **primero** una ocurrencia m de la parte izquierda L en el grafo corriente G, tal que m preserva la estructura. Cuando se realiza una transformación de grafos directa con una producción p y un grafo G, se borran de G todos los vértices y arcos que se corresponden con L \ K. En general, la parte borrada no es un grafo, pero la estructura resultante D := (G \ m(L)) $\cup$ m(K) aún continúa siendo un grafo válido, es decir, no debería haber arcos sueltos. Esto significa que la ocurrencia m debe satisfacer una condición denominada *gluing condition*, lo cual asegura que la unión de L \ K y D son iguales a G (ver (1) en la Figura C.6). En el **segundo** paso de la transformación de grafos directa, el grafo D es agregado a R \ K para obtener el grafo derivado H (ver (2) en la Figura C.6). Una transformación de grafos o, más precisamente, una secuencia de transformación de grafos consiste de cero o más transformaciones de grafos directas.

Formalmente, una transformación de grafos directa con p y m se define como sigue. Dada una producción p = (L ← K → R) y un grafo de contexto D, el cual incluye la interface K, el grafo origen G de la transformación G $\Rightarrow$ H vía p está dado por la unión de L y D vía K, escrita G = L $+_K$ D, y el grafo destino H está dado por la unión de R y D vía K, escrita H = R $+_K$ D. Más precisamente, se usan los grafos de morfismo K → L, K → R, y K → D para expresar cómo K es incluido en L, R y D, respectivamente. Esto permite definir

las construcciones de pegado G = L $+_K$ D y H = R $+_K$ D como las construcciones pushout (1) y (2) en la Figura C.6, dando lugar a un doble pushout. El morfismo de grafo resultante R → H se denomina co-ocurrencia (comatch) de la trasformación de grafos G ⇒ H.
Para poder aplicar una producción p con una ocurrencia de L en G, dado un morfismo de grafo m: L → G como el de la Figura C.6, primero se debe construir un grafo de contexto D, tal que la unión L $+_K$ D de L y D vía K sea igual a G. En el segundo paso se construye la unión R $+_K$ D de R y D vía K, dando lugar al grafo H y así a la transformación de grafos DPO de G ⇒ H vía p y m. Para la construcción del primer paso, sin embargo, debe satisfacerse una condición de unión (gluing condition), lo cual permite construir D con L $+_K$ D = G. En el caso de un morfismo m inyectivo, los estados de la condición de unión que todos los puntos sueltos de L, es decir, todos los nodos x en L tal que m(x) es origen o destino de un arco e en G \ L, deben ser puntos de unión x en K.

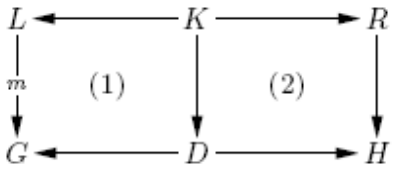


Figura C.6. Transformación de Grafos DPO

En la Figura C.7 se muestra un ejemplo simple de un paso de transformación de grafos DPO, que se corresponde con el esquema general de la Figura C.6. En el diagrama (PO1) G es la unión de los grafos L y D a través de K, donde el número de los nodos indica cómo se hace la correspondencia de nodos por el morfismo de grafos. La correspondencia de arcos puede deducirse unívocamente de la correspondencia de nodos. En la Figura C.7 se puede ver que la condición de unión se satisface ya que los puntos sueltos (1) y (2) de L son también puntos de unión. Además, H es la unión de R y D a través de K en (PO2), dando lugar a la transformación de grafos G ⇒ H vía p. Por razones técnicas, en las producciones los morfismos K → L y K → R usualmente se restringen a morfismos inyectivos. Pero también se permiten las ocurrencias no inyectivas m: L → G y las co-ocurrencias n: R → H.

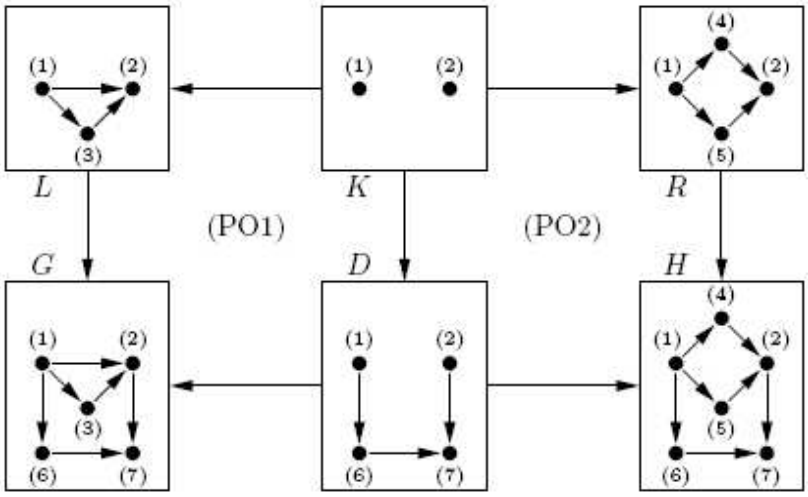


Figura C.7. Ejemplo de Transformación de Grafos DPO

C.4.2 El enfoque SPO

En el enfoque algebraico las construcciones *pushout* pertenecen a la categoría de **Grafos**, basados en morfismos de grafos totales [EEPT06].

Por otro lado, la producción p = (L←K→R) que se muestra en la Figura C.8 también puede considerarse como un morfismo de grafo parcial p: L → R con un dominio dom(p) = K. Además, el span (G ← D → H) puede considerarse un morfismo de grafos parcial s: G→ H con dom(s) = D. Esto da lugar al diagrama de la Figura C.8, donde los morfismos horizontales son parciales y los verticales son totales. De hecho, esta figura es un *pushout* en la categoría de **PGrafos** de morfismos de grafos parciales y totales y muestra que la transformación de grafos puede expresarse con un único *pushout* en la categoría de **PGrafos**.

Desde un punto de vista operacional, el enfoque SPO difiere del enfoque DPO en lo que concierne al borrado de los elementos del grafo de contexto durante un paso de transformación de grafo. En el enfoque DPO, si la ocurrencia m: L → G no satisface la condición de unión con respecto a la producción p = (L←K→R), entonces la producción no se puede aplicar. Pero sí se puede aplicar en el enfoque SPO, lo cual permite que queden arcos sueltos después de la eliminación L \ K de G. Sin embargo, los arcos sueltos en G también se borran, dando lugar a un grafo bien definido H. Es decir, la regla se puede aplicar incluso si produce arcos sueltos, simplemente se borran estos arcos.

Si, en la Figura C.7, el vértice (2) fuera borrado de K, la condición de unión no sería satisfecha en el enfoque DPO. En el enfoque SPO, esto significa que el vértice (2) no está en el dominio de p, dando lugar a un arco suelto e en G después de la eliminación L \ dom(p) en la Figura C.8. Como resultado, el arco e debería borrarse en H.

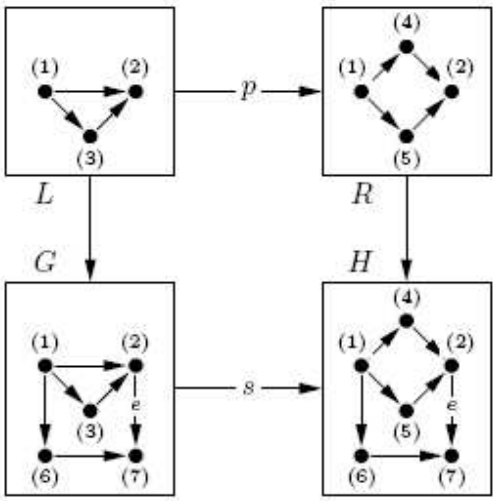


Figura C.8. Ejemplo de Transformación de Grafos SPO

C.5 Las Reglas de Transformación de Grafos en el Modelado Orientado a Objetos

Grafos

Como se expresó en el capítulo 6, en el modelado orientado a objetos los grafos ocurren a dos niveles: a nivel de tipos (dado por el diagrama de clases) y a nivel de instancia (dado por los diagramas de objetos válidos). Esta noción se puede generalizar en el concepto de grafos tipados, los cuales pueden verse como una representación abstracta de los diagramas de clases de UML. Sus diagramas de objetos son grafos equipados con una correspondencia a los grafos tipados (que preserva la estructura), formalmente expresada como un homomorfismo de grafos [BH04] [Heckel06].

La Figura C.9 muestra un ejemplo de un diagrama de objetos y un diagrama de clases en la notación UML, modelando una aplicación bancaria. El diagrama de objetos UML de la izquierda (representando un grafo instancia) se puede corresponder al diagrama de clases de la derecha (representando un grafo tipo), definiendo que type(object) = Class

para cada instancia object : Class en el diagrama de objetos. Extendiendo la idea a los enlaces, la preservación de la estructura significa que, por ejemplo, un enlace entre los objetos object1 y object2 debe corresponderse a una asociación en el diagrama de clases entre type(object1) y type(object2). Por el mismo mecanismo de compatibilidad estructural se puede asegurar que un atributo de un objeto está declarado en la clase correspondiente, etc.

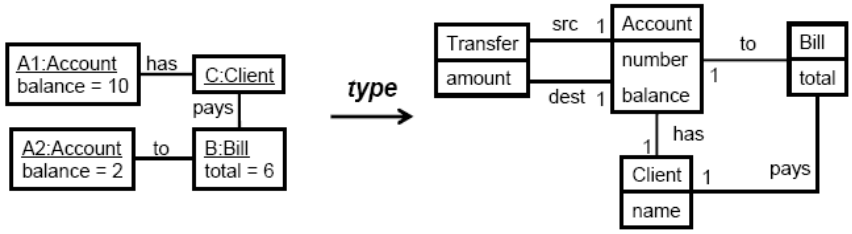


Figura C.9. Diagrama de Objetos tipado (Izquierda) sobre un Diagrama de Clases (derecha)

Reglas y Transformaciones

Una regla de transformación de grafos p: L → R consiste de un par de grafos instancia L, R tal que la unión L ∪ R está definida. Esto significa que, por ejemplo, los arcos que aparecen en L y R están conectados a los mismos vértices en ambos grafos, o que los vértices con el mismo nombre deben tener el mismo tipo, etc. La parte izquierda L, representa la precondición de la regla y la parte derecha R representa la poscondición.

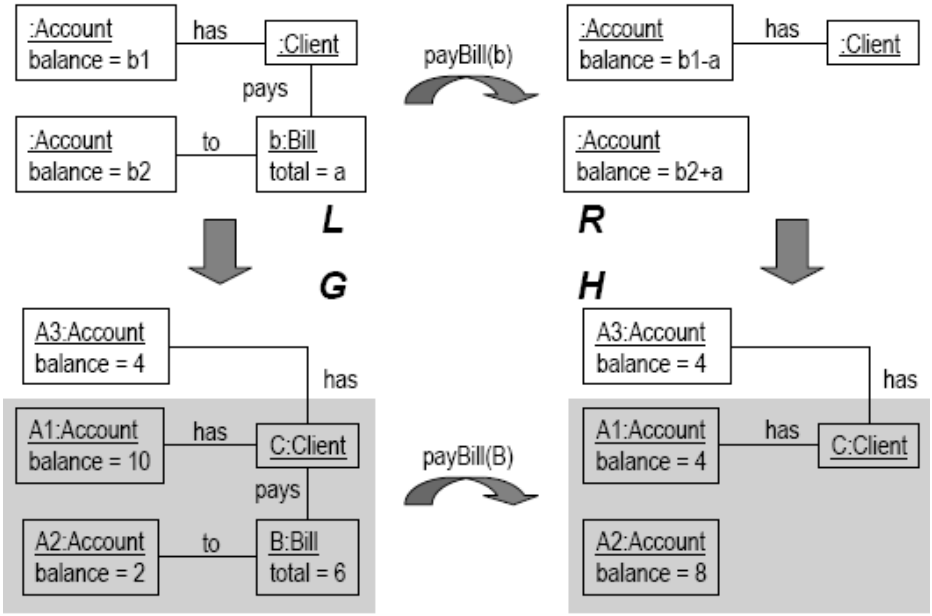


Figura C.10. Un ejemplo de transformación usando la regla payBill

Una ***transformación de grafos*** desde el pre-estado G al pos-estado H está dada por un homomorfismo o: L ∪ R → G ∪ H, llamado *ocurrencia*, tal que:

- o(L) ⊆ G y o(R) ⊆ H, es decir, la parte izquierda de la regla está incluida en el pre-estado y la parte derecha de la regla está incluida en el pos-estado,
- o(L \ R) = G \ H y o(R \ L) = H \ G, es decir, la parte de G que se borra es la que se corresponde con los elementos de L que no están incluidos en R y, simétricamente, la parte de H que se agrega es la que se corresponde con los elementos nuevos en R.

La Figura C.10 muestra la aplicación de la regla de transformación de grafos payBill que modela el pago de una factura mediante la transferencia del monto requerido desde la cuenta del cliente.

C.6 Modelado y Meta modelado

Las técnicas de transformación de grafos se pueden aplicar a en dos áreas bien diferenciadas: el modelado de sistemas individuales, y la especificación de la sintaxis y la semántica de los lenguajes de modelado visual. Sin embargo, en ambos casos, se pueden utilizar herramientas y notaciones similares. Y se utiliza el término meta modelado

con el fin de subrayar el hecho de que se puede definir un lenguaje con las mismas técnicas que se usan para modelar un sistema particular.

C.6.1 Modelado con Transformación de Grafos

En esta sección se ilustra el uso de las técnicas de grafos en el modelado de los requerimientos funcionales de un sistema de software.

Objetos dinámicos

Los requerimientos funcionales se presentan a menudo en forma de casos de uso, es decir, servicios que un sistema le provee a sus usuarios. Cada caso de uso provee una descripción más detallada de su precondición y poscondición y de la interacción requerida para realizar el servicio respectivo. La transformación de grafos tipados han sido propuestos como una forma de especificar los casos de uso de manera visual, y a la vez formal, con los beneficios adicionales de una especificación ejecutable. Un ejemplo de tal especificación se muestra en la Figura C.11, con la regla *payBill* y su aplicación. Las reglas de transformación de grafos, como *payBill*, proveen una especificación de los requerimientos funcionales de alto nivel, en términos de su pre y poscondición, abstrayéndose de las acciones y los estados intermedios. Si interesa tratar una estructura más detallada, se puede descomponer la regla en pasos más elementales.

Por ejemplo, la regla *payBill(b)* se puede descomponer secuencialmente en *payBill(b)=createTransfer(b,t); executeTransfer(t),* donde la relación entre las dos reglas elementales se especifica por los parámetros abstractos b:Hill y t:Transfer. Esta descomposición se muestra en la Figura C.11, donde createTransfer(b,t) se aplica a la parte izquierda de payBill(b) y executeTransfer(t) se aplica al grafo resultante, el resultado es la parte derecha de la regla original.

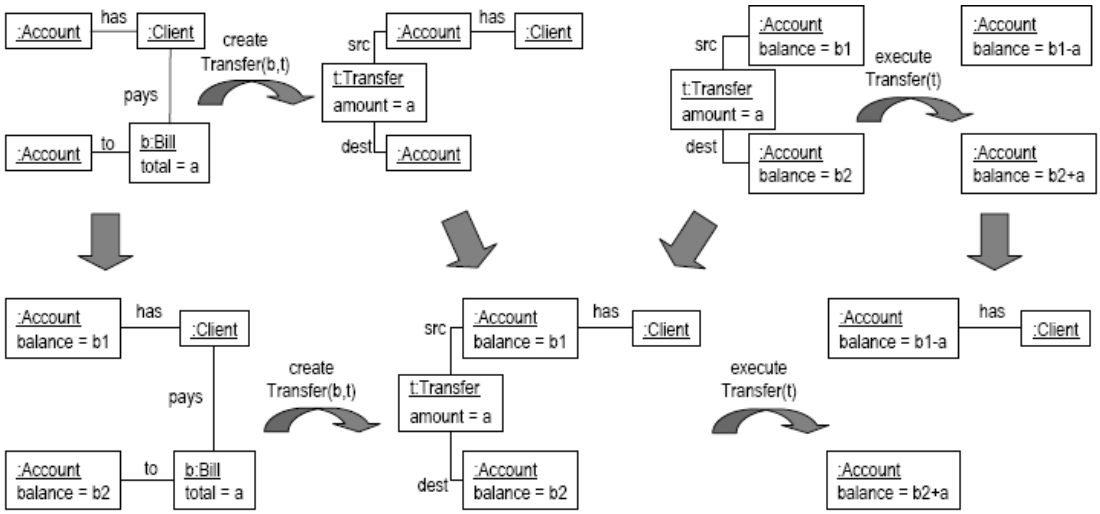


Figura C.11. Regla *payBill* derivada de *createTransfer(b,t)* y *executeTransfer(t)*

La condición de consistencia semántica para esta clase de descomposición establece que, para un grafo G dado, existe una transformación:

$$G \overset{\text{payBill(B)}}{\Longrightarrow} H$$

sí y solo sí hay un grafo X y transformaciones:

$$G \overset{\text{createTransfer(B,T)}}{\Longrightarrow} X \overset{\text{executeTransfer(T)}}{\Longrightarrow} H$$

Se puede observar que la regla compuesta *payBill* tiene dos pasos de transacción. La condición semántica deseada puede chequearse descomponiendo la regla elemental, como se muestra en la Figura C.11. Primero, *createTransfer(b,t)* se aplica a la parte izquierda de *payBill(b)*. Luego, *executeTransfer(t)* se aplica al grafo resultante. El resultado de este segundo paso debe ser isomórfico a la parte derecha de la regla original.

C.6.2 Meta modelado con Transformación de Grafos

En la sección anterior, se utilizó un meta modelo para especificar la sintaxis abstracta de diagramas estáticos, tales como los diagramas de clases y los diagramas de objetos, y las reglas de transformación de grafos fueron usadas para modelar la estructura de los

objetos y las computaciones. De manera más general, las técnicas de meta modelado pueden ser usadas para definir la sintaxis abstracta, la sintaxis concreta o la semántica de cualquier notación de modelado, tanto para los aspectos estáticos como los dinámicos. La idea clave es que estas estructuras pueden ser especificadas por medio de diagramas de clases. Entonces, la cuestión crucial es cómo relacionar los tres niveles con el fin de definir la sintaxis y la semántica de un lenguaje. En esta sección se discute el uso de sistemas de transformación de grafos para especificar la correspondencia entre la sintaxis abstracta y concreta y entre la semántica abstracta y concreta, así como para la definición de una semántica operacional basada en la interpretación directa de modelos a un nivel de sintaxis abstracta.

En la Figura C.12 se muestra un simple ejemplo con un diagrama de estados que modela el comportamiento del componente *cashBox*, como se observa a través de la interface *CardReader*.

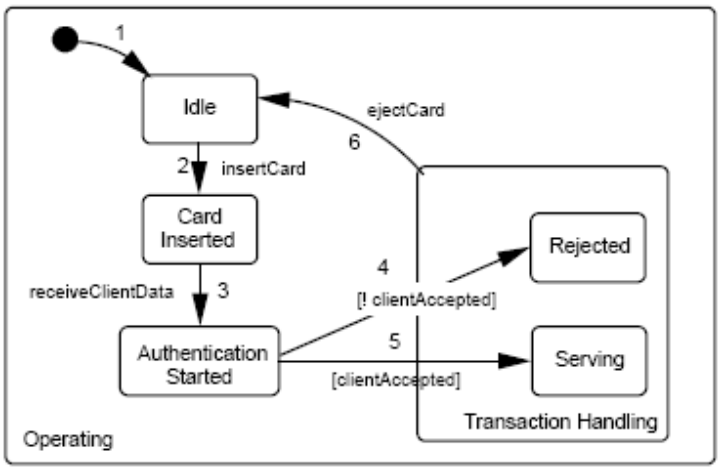


Figura C.12. Diagrama de Estados de un CardReader modelando el comportamiento de un CashBox

Si cashBox es encendido, cambia del estado inicial al estado Operating, moviéndose al estado por defecto idle. Una vez que la primera tarjeta es insertada (ocurre el evento insertCrad), el componente entra en el estado Card Inserted, y cuando recibe los datos del cliente desde la tarjeta, (evento receiveClientData), se mueve al estado Authentication Started. Aquí, ocurre una de las dos opciones dependiendo del dato recibido. Si el cliente es aceptado, el componente entra en el estado *Serving*, en otro caso entra en el estado *Rejected*. En ambos casos, luego de procesarse la transacción, el componente retorna al estado *idle*, tan pronto como la tarjeta es expulsada (evento *cardEjected*). Este diagrama,

junto con su interpretación semántica intuitiva, es suficiente para considerar todos los aspectos que define una notación visual. Más precisamente, se puede organizar en tres capas la definición del lenguaje, como se muestra en la Figura C.13.

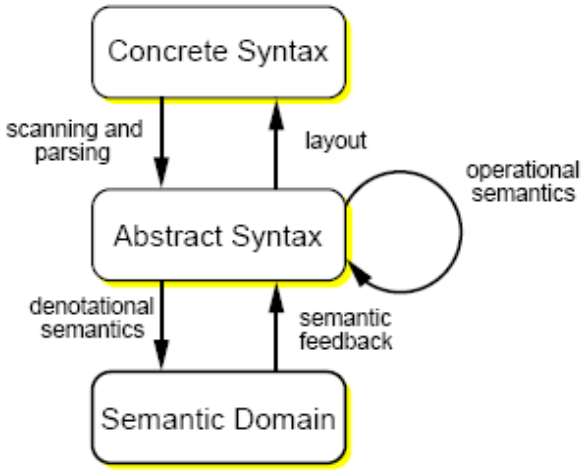


Figura C.13. Vista en capas de un lenguaje de modelado visual

A continuación se describe en detalle las transformaciones entre estas capas, representadas en la figura por las flechas.

Scanning and Parsing.

En una dirección Top-Down, cada lenguaje tiene una *sintaxis concreta*, la cual define cómo los usuarios perciben los diferentes elementos de modelado suministrados por la notación. En los lenguajes gráficos, la sintaxis concreta estará dada por burbujas, rectángulos, líneas, flechas, etc. Por ejemplo, en la Figura C.12 los estados están representados por rectángulos con las puntas redondeadas y las transiciones con flechas dirigidas. Las relaciones entre estos elementos pueden ser: una línea conectando dos rectángulos, un rectángulo conteniendo otros rectángulos, un elemento ubicado a la derecha o izquierda de otro elemento. A este nivel, una gramática de grafos define la sintaxis concreta del lenguaje, y es esta gramática la que se debe usar para explorar los modelos de usuario.

Por su parte, la *sintaxis abstracta* define los elementos de modelado suministrados por la notación y las relaciones entre ellos, pero sin detalles concretos. Los modelos de este nivel pueden ser analizados para chequear si son sintácticamente correctos. Este nivel es comparable a la representación de los modelos de UML como instancias de un meta

modelo [UML]. La principal diferencia es el estilo declarativo de la especificación en el meta modelo, el cual define la formación correcta de los elementos por medio de restricciones lógicas, como opuesto al estilo constructivo empleado al usar una gramática para generar los modelos.

La Figura C.14 muestra un grafo simplificado de la sintaxis abstracta del diagrama de estados de la Figura C.12. El camino inverso, que permite asociar la sintaxis abstracta a una sintaxis concreta, significa que se está definiendo una capa concreta de modelos.

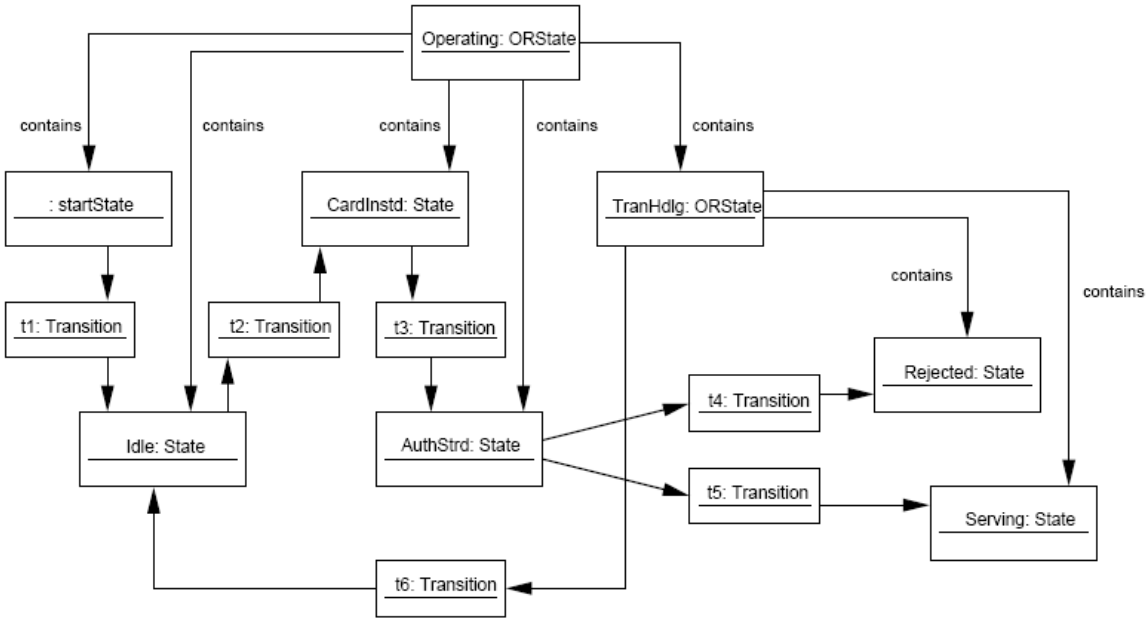


Figura C.14. Grafo de la sintaxis abstracta para el Diagrama de Estados de la Figura C.12

Semántica Operacional

Hay esencialmente dos formas de definir semántica operacional por medio de la transformación de grafos. Primero, las reglas de transformación de grafos pueden especificar un intérprete abstracto para el lenguaje completo. Segundo, cada modelo puede ser compilado en base a un conjunto de reglas. Siguiendo el segundo enfoque, los diagramas de estados de UML son traducidos a *sistemas de transformación de grafos estructurados* que satisfacen las reglas bien formadas impuestas por la notación. Los estados activos son representados como configuraciones de estados, los cuales son subgrafos en la jerarquía de estados. Los estados activos cambian cuando se disparan

transiciones, tal como se especifica con reglas de transformación de grafos. La parte izquierda identifica la configuración actual, es decir, el actual estado activo (s), la parte derecha define configuración alcanzada mediante la aplicación de la regla (disparo de la transición), es decir, el nuevo conjunto de estados activos. La Figura C.15 muestra las transiciones de la Figura C.12 como reglas de transformación de grafos.

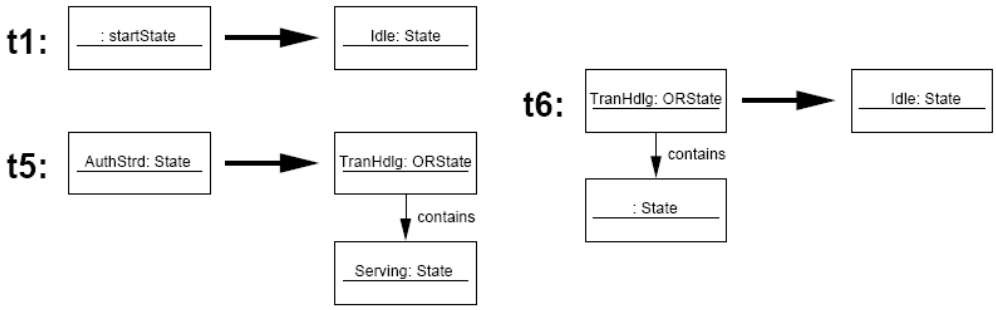


Figura C.15. Transiciones de la Figura C.12 como Reglas de Transformación de Grafos

t1: mueve el estado corriente desde el estado de comienzo al estado *Idle* (inactivo).
t5: mueve el estado corriente desde *Authentication Started* a la jerarquía *Transaction Handling/Serving*.
t6: mueve el estado corriente desde la jerarquía *Transaction Handling/cualquier estado contenido* al estado *Idle*.

Semántica Denotacional

La última capa de la Figura C.13 introduce el *dominio semántico*. La *semántica* definida como una asociación entre la sintaxis abstracta en un dominio semántico es llamada *semántica denotacional*. En el ejemplo, para definir la semántica dinámica de los diagramas de estados, el dominio semántico en sí mismo debe proveer un modelo operacional, así, la semántica operacional y la denotacional ocurren en combinación.

Usualmente, se selecciona un dominio semántico, es decir, un método formal simple cuyas reglas de ejecución están bien establecidas, y se define el comportamiento de cada elemento sintáctico abstracto a través de una adecuada asociación en el dominio semántico. En este caso, el rol jurado por la transformación de grafos depende del método formal seleccionado. Si se trata de una versión en modo texto, las producciones de la gramática que define la sintaxis abstracta pueden ser aumentadas anotaciones de texto para construir la representación semántica. Más generalmente, las producciones se pueden asociar con la gramática textual que específica el modelo semántico, y la

aplicación de una producción de la gramática de la sintaxis abstracta debería disparar automáticamente la aplicación de la producción textual asociada.

Un enfoque similar se describe en [Baresi97], donde Redes de Petri Temporizadas de alto nivel son usadas como dominio semántico y las reglas son pares de producciones de gramáticas de grafos. La primera producción define cómo modificar la representación sintáctica abstracta, y la segunda define los estados correspondientes a cambios sobre las Redes de Petri funcionalmente equivalentes. Por ejemplo, usando las reglas de transformación definidas en [Baresi97] se podría atribuir semántica dinámica formal al diagrama de estados de la Figura C.12 a través de la Red de Petri de la Figura C.16. Las hipótesis usadas para transformar un diagrama de estados en una Red de Petri funcionalmente equivalente son: los estados son directamente asociados a *places* y las transiciones de estados a *transiciones* de la Red de Petri. El estado de inicio se asocia a *places marcados,* etc.

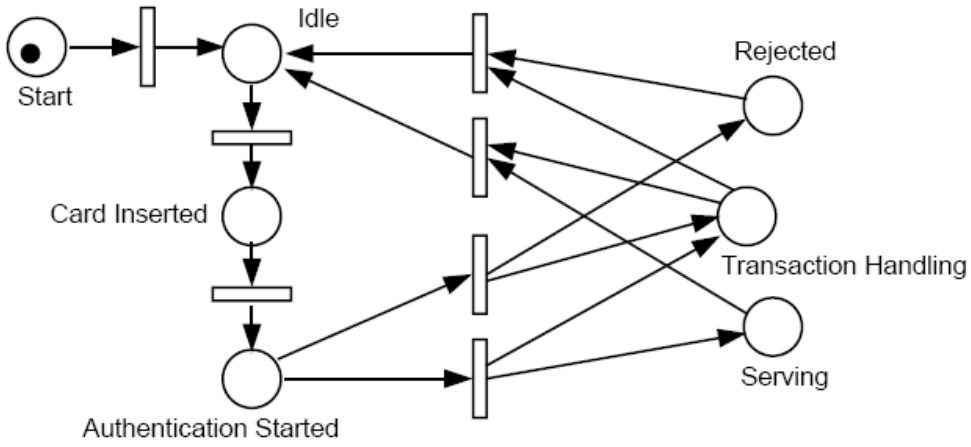


Figura C.16. La Figura C.12 en términos de una Red de Petri

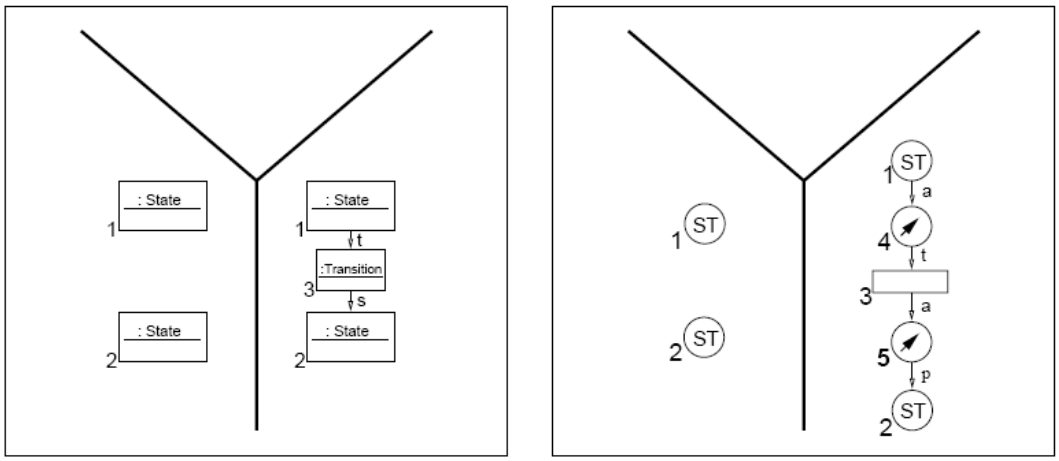


Figura C.17. Una regla de transformación simple tomada de [Baresi97]

La regla de transformación aplicada en la Figura C.17 muestra como conectar dos diagramas de estados a través de una transición. La producción izquierda modifica el grafo de la sintaxis abstracta agregando la nueva transición, mientras que la producción derecha modifica la Red de Petri agregando una transición de la Red de Petri, junto con dos arcos entre los dos *places* que corresponden a los estados del diagrama de estados.

APÉNDICE D.

Interfaz de las Aplicaciones Invocadas

El propósito de este anexo es detallar las APIs (Application Programming Interfaces) definidas en el estándar de la WfMC, las cuales deben soportar los Sistemas de Gestión de Flujos de Trabajo (SGFT). Las llamadas a las APIs proveen un método consistente de acceso a las funciones del flujo de trabajo a través de los motores de flujo de trabajo. Este conjunto de APIs se denomina Workflow Aplication Programming Interfaces (WAPIs).

La estructura del anexo es la siguiente. La sección 1 presenta una introducción a la especificación de interfaces para invocar aplicaciones. La sección 2 describe la Interfaz de las Aplicaciones Invocadas propuesta por la WfMC. La sección 3 se refiere a las funciones de conexión que permiten crear y terminar conexiones con las interfaces del *Agente de Aplicación* de la Interfaz. Finalmente, la sección 4 presenta las WAPIs para la invocación de aplicaciones. Incluye la sub-secciones D.4.1 a D.4.4, las cuales presentan la especificación y una explicación detallada de cada una de las WAPIs que componen la interfaz.

D.1 Introducción

Incorporar un Sistema de Gestión de Flujo de Trabajo en una organización implica habilitar un ambiente de interfaz de flujo de trabajo, para permitir la comunicación con el mismo. Adicionalmente, esta interfaz debe otorgar un cierto grado de protección sobre los sistemas de software existentes en la organización.

Las interfaces de la WfMC que permiten invocar aplicaciones no definen un mecanismo de control de aplicaciones directo. Actualmente, los clientes y proveedores deben confrontarse con diferentes sistemas operativos y mecanismos de comunicación de aplicaciones. Así, un Sistema de Gestión de Flujos de Trabajo (SGFT) necesita contar con una interfaz para especificar los controladores de la aplicación. Con la definición de estos controladores para invocar y controlar las aplicaciones, la WfMC ofrece una interfaz que facilita un protocolo estandarizado entre los productos de flujos de trabajo y cualquier otro sistema de software. Las WAPIs definidas en esta interfaz ofrecen un protocolo estandarizado entre los diferentes productos de gestión de flujos de trabajo y cualquier otro producto de software.

En la actualidad, una gran variedad de herramientas de gestión de flujos de trabajo soportan mecanismos especializados para integrar las aplicaciones y permitir el intercambio de información. Mientras que todos estos mecanismos son, en su mayoría, implementados individualmente según los requerimientos específicos de un cliente, las compañías dedicadas a la integración de sistemas y proveedores involucrados como terceras partes, deben re-implementar estos mecanismos a la hora de usar otra herramienta de gestión de flujos de trabajo. Consecuentemente, su interés en soportar la generación de este tipo de interfaces, es, de hecho, muy alto, ya que permite mejorar su trabajo diario.

Podría parecer muy sencillo permitir al flujo de trabajo acceder a aplicaciones comunes, no obstante, los ambientes de gestión de flujo de trabajo típicamente deben incluir una serie de aplicaciones especializadas, las cuales permiten la ejecución en ambientes heterogéneos. Tanto los sistemas de gestión de flujos de trabajo como las plataformas de integración son requeridas por el mercado actual y demandan la especificación de interfaces de aplicación generalizadas y estandarizadas.

D.2 La Interfaz de las Aplicaciones Invocadas

El conjunto de funciones de la Interfaz de Aplicaciones Invocadas provee servicios a los *Agentes de Aplicación* (*tool-agents*), para invocar y controlar las aplicaciones asociadas con ítems de trabajo específicos [WfMCb].

La Interfaz de las Aplicaciones Invocadas define un conjunto de APIs, altamente recomendadas para ser usadas por los componentes del Sistema de Gestión de Flujos de Trabajo (por las aplicaciones de cliente y el motor), para controlar estos dispositivos de aplicaciones especializadas, llamados agentes de aplicación. Estos agentes de aplicación finalmente comienzan y terminan las aplicaciones, pasan información relevante del flujo de trabajo y la aplicación "a" y "desde" la aplicación y controlan el estado a nivel de ejecución de la aplicación. Así, las WAPI's de la Interfaz de las Aplicaciones Invocadas trabajan solamente con los agentes de aplicación. Una aplicación puede requerir información adicional al SGFT vía un *Agente de Aplicación,* a través de la funciones WAPI estándar.

Como la Interfaz de Aplicaciones Invocadas debe manejar requerimientos bi-direccionales (requerimientos a y desde la aplicación), dicha interacción con los agentes de aplicación dependerá de la arquitectura de la aplicación. Se debería permitir el requerimiento y la actualización de datos de la aplicación y otras funcionalidades importantes en tiempo d ejecución.

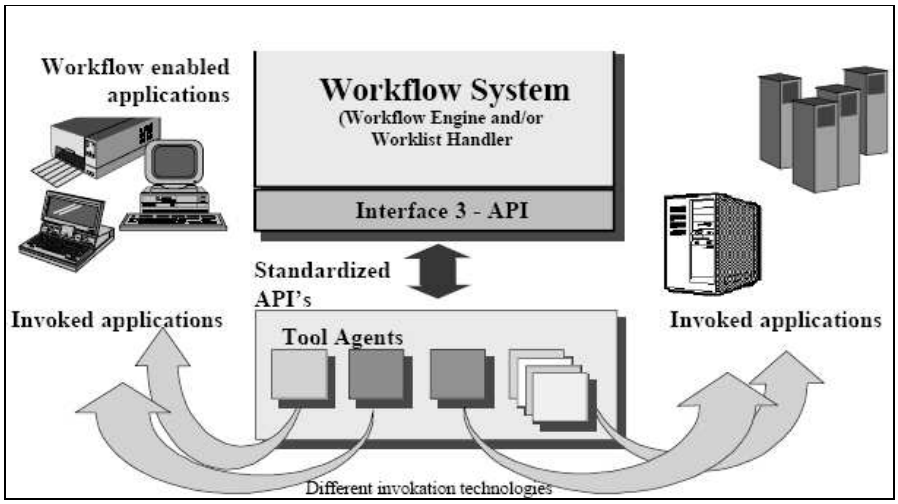


Figura D.1. Interfaz de las Aplicaciones Invocadas

El SGFT debe saber cuál es el *Agente de Aplicación* que está instalado. La arquitectura básica del *Agente de Aplicación* podría comparase con la interfaz de un controlador, como por ejemplo ODBC. Dentro de esta definición de interfaz, ningún mecanismo de comunicación entre el *Agente de Aplicación* y el SGFT es necesario.

D.3 Funciones de Conexión/Desconexión

Las funciones de conexión permiten crear y terminar conexiones con las interfaces del *Agente de Aplicación* [WfMCb]. Las aplicaciones pueden requerir procedimientos de identificación del usuario ante el sistema (login), por lo que la autenticación del usuario debería pasarse al *Agente de Aplicación* para proveer mecanismos simples de identificación de usuarios.

D.3.1 Descripción general de la conexión/desconexión

Las APIs de la WfMC denominadas *WMConnect/WMDisconnect* están destinadas a ligar un conjunto de trabajos relacionados a una aplicación. Cuando se usa, *WMConnect* retorna un valor, el cual es único y se usa en todas las llamadas a las APIs de la WfMC. El comando *WMConnect* que la información sea suministrada una sola vez y continúe siendo válida hasta que ocurra la desconexión. La información provista durante la conexión incluye información de identificación relacionada a quién/qué esta requiriendo un servicio de autenticación al motor del flujo de trabajo. La estructura de manejo de sesión que retorna la función de conexión contiene un puntero a una estructura que contiene el identificador de la sesión (session ID) y otro puntero a una estructura con información de conexión provista por el usuario. En aquellos servidores de flujos de trabajo que establecen una conexión, el motor del flujo de trabajo debería ser el encargado de retornar el identificador de la sesión y la información específica del proveedor, al momento de establecer la misma. En aquellos servidores de flujos de trabajo que no establecen una conexión, el identificador de la sesión debería establecerse en 0, y el puntero a la información de conexión provista por el usuario debería almacenarse en una estructura privada contenida en la estructura de manejo de sesión.

D.3.2 Operaciones entre las APIs y el motor del flujo de trabajo

La construcción de las APIs de la WfMC está destinada a tener un impacto mínimo sobre la estructura operacional del producto que las soporta. Las llamadas a APIs son consideradas un protocolo neutral, una vez que se cruzan sus límites. Se pueden emplear diferentes tipos de mecanismos para entregar la solicitud al motor del flujo de trabajo. El método particular del SGFT que permite la interacción entre las llamadas a APIs y el motor del flujo de trabajo, puede implementarse por medio de diferentes tecnologías como *Remote Procedure Call* (*RPC*), mecanismos conversacionales, envío de mensajes en un modo de transmisión *connectionless*, etc. Un producto de gestión de flujo de trabajo particular puede elegir entre aceptar el comando *WMConnect*, retornar un manejador, o ignorarlo. Los anteriores son ejemplos de las posibles operaciones desarrolladas por los productos de gestión de flujo de trabajo para soportar el comando de conexión. Obviamente, también hay otras posibles. En algunos casos, un producto requerirá conectar una estación de trabajo simple con múltiples motores de flujos de trabajo. Es posible que múltiples comandos de conexión estén activos concurrentemente y los subsecuentes comandos APIs deben estar dirigidos al motor correcto.

Por su parte, el resultado del comando de desconexión *WMDisconnect* es muy variado, y depende de la implementación particular del producto de flujo de trabajo. Su propósito es indicar que la aplicación ya no accederá a las funciones del motor del flujo de trabajo, en el contexto previamente establecido. En algunos productos, previa recepción del comando de desconexión, se pueden liberar las comunicaciones y otro tipo de recursos.

D.3.3 Estructura operacional de una aplicación cuando se usan las llamadas a APIs

La estructura operacional de una aplicación en cuanto al uso de las llamadas a las APIs de la WfMC se ve afectada por la forma en que se construyeron estas llamadas a APIs. La construcción corriente de las llamadas a APIs resulta en un segmento de código de la aplicación donde se hace la llamada a la API, para correr en modo bloqueado. Esto es, la aplicación invoca un comando API y espera una respuesta del motor del flujo de trabajo. Cuando se hace la llamada API, el segmento de código de la aplicación le da el control a la API y no lo recupera hasta que el comando API es satisfecho.

Durante gran parte del tiempo, los comandos API serán utilizados por los participantes del flujo de trabajo, vía una interfaz de usuario final. La mayoría de los comandos API actuales no suponen que un participante del flujo de trabajo esté interesado en hacer un requerimiento, hacer algo más, y un tiempo más tarde ver la respuesta real al requerimiento. Con los tipos de requerimientos soportados por el conjunto de las APIs, podría ser normal el caso de que un participante del flujo de trabajo quiera ver la respuesta a su requerimiento tan pronto como sea posible.

Las llamadas APIs pueden construirse de forma tal de permitir al segmento de código de la aplicación hacer la llamada API para ejecutarse en modo desbloqueado. Esto es, hacer la llamada API con un "retorno inmediato", en vez de esperar por la respuesta actual a la acción requerida. Si esto fuera así, la WfMC necesitará definir funciones adicionales para soportar el modo de transmisión *connectionless* de una operación (de alguna manera, obtener la respuesta asincrónica cuando esta llega y otorgársela al participante).

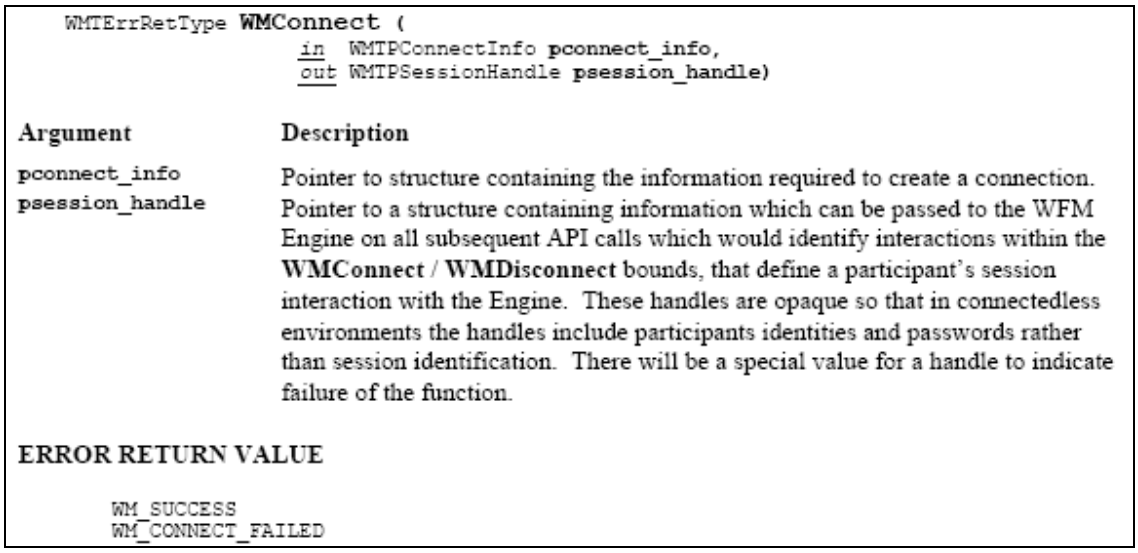
Los comandos para la conexión/desconexión no tienen nada que ver con la habilidad de la aplicación de correr conectada o en modo *connectionless*.

D.4 WAPIs de la invocación de aplicaciones

D.4.1 WAPIs de conexión/desconexión

WMConnect

Este comando permite conectarse al motor del flujo de trabajo, para realizar una serie de interacciones. El comando le informa al motor que otros comandos serán originarios de esta fuente [WfMCb]. La Figura D.2 presenta la especificación de la WAPI asociada a este comando.



```
WMTErrRetType WMConnect (
                    in  WMTPConnectInfo pconnect_info,
                    out WMTPSessionHandle psession_handle)
```

Argument	Description
pconnect_info	Pointer to structure containing the information required to create a connection.
psession_handle	Pointer to a structure containing information which can be passed to the WFM Engine on all subsequent API calls which would identify interactions within the **WMConnect / WMDisconnect** bounds, that define a participant's session interaction with the Engine. These handles are opaque so that in connectedless environments the handles include participants identities and passwords rather than session identification. There will be a special value for a handle to indicate failure of the function.

ERROR RETURN VALUE

```
WM_SUCCESS
WM_CONNECT_FAILED
```

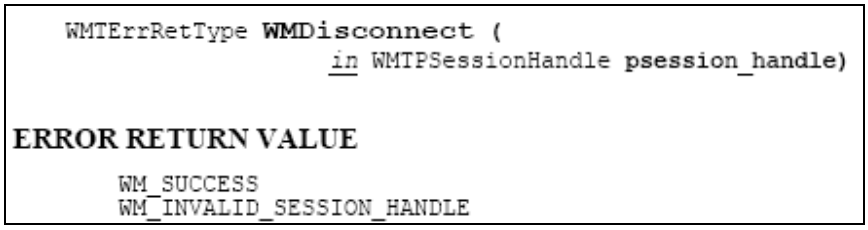
Figura D.2. WAPI del comando WMConnect

Los argumentos son:

- *pconnect_info*: es un puntero a una estructura que contiene la información requerida para crear la conexión.
- *psession_handle*: es un puntero a una estructura que contiene la información que será pasada al motor del flujo de trabajo y a todas las subsecuentes llamadas APIs, y servirá para identificar las interacciones dentro de los límites de la conexión establecida.

WMDisconnect

Este comando permite la desconexión con el motor del flujo de trabajo. El comando le indica al motor que no habrá más llamadas APIs originarias de la fuente para la que se creó la conexión. El motor puede descartar los datos del estado de la conexión o tomar otras acciones. La Figura D.3 presenta la especificación de la WAPI WMDisconnect.



```
WMTErrRetType WMDisconnect (
                    in WMTPSessionHandle psession_handle)
```

ERROR RETURN VALUE

```
WM_SUCCESS
WM_INVALID_SESSION_HANDLE
```

Figura D.3. WAPI del comando WMDisconnect

D.4.2 WAPI para invocar una aplicación

WMTAInvokeApplication()

Este comando fuerza al *Agente de Aplicación* a comenzar o activar una aplicación específica [WfMCb]. El motor del flujo de trabajo activa una aplicación específica asociada con un ítem de trabajo invocando la API del *Agente de Aplicación*. Las aplicaciones ya podrían estar activadas o tener que ser activadas por el agente. Invocar una aplicación siempre incluye el pasaje de información adicional tal como parámetros de la aplicación y modos. La Figura D.4 presenta la especificación de la WAPI asociada a este comando.

```
int WMTAInvokeApplication (
                in int tool_agent_handle,
                in string application_name,
                in WMTPProcInstID pproc_inst_id,
                in  WMTPWorkItemID pwork_item_id,
                in WMTPAttributeList pattribute_list,
                in int app_mode)
```

Argument Name	Description
tool_agent_handle	This handle represents one connection to a specific Tool Agent
application_name	This parameter represents the name of the executable file or component. The application name must be passed without the path name. (The Tool Agent implemetation and configuration has to handle the local configuration.)
pproc_inst_id	Process instance, to identify the relation between the application and a process instance. This ID allows the System to reference to a specific application handle of the Tool Agent.
pwork_item_id	Work Item associated with invoked application
pattribute_list	Pointer to a list of parameters and attributes which are required by the application. These parameters could be either application relevant, or dynamic, or workflow relevant data. (e.g. filename, record identifer, etc.)
app_mode	Represents a possible application mode like "CREATE", "UPDATE", "READ-ONLY", "PRINT", etc..

ERROR RETURN VALUE

```
WM_SUCCESS
WM_APPLICATION_NOT_STARTED
WM_APPLICATION_NOT_DEFINED
WM_APPLICATION_BUSY
```

Figura D.4. WAPI del comando WMTAInvokeApplication

El comando permite invocar una aplicación específica asociada a un ítem de trabajo. El *Agente de Aplicación* debería controlar una o múltiples aplicaciones, las cuales deberían ser activadas o invocadas. También, una aplicación debe comenzarse en un modo específico tal como "open" o "update".

D.4.3 WAPI para solicitar el estado de una aplicación

WMTARequestAppStatus()

Este comando le permite al SGFT chequear las aplicaciones activas y su estado (corriendo, pendiente, etc.) [WfMCb]. El mismo define cómo el sistema de flujo de trabajo tiene que chequear el estado de una aplicación y recupera los datos relevantes del flujo de trabajo desde la aplicación. Para recuperar los datos relevantes del flujo de trabajo desde una aplicación invocada, el motor del flujo de trabajo debe requerir el estado de la aplicación y demás información adicional al *Agente de Aplicación*. Debido a algunos requerimientos asincrónicos ente aplicaciones integradas, el *Agente de Aplicación* puede requerir información adicional que usarán otras interfaces de WAPIs. La Figura D.5 presenta la especificación de la WAPI asociada a este comando.

```
int WMTARequestAppStatus (
                in WMTInt32 tool_agent_handle,
                in WMTPProcInstID proc_inst_id,
                in  WMTPWorkItemID pwork_item_id,
                out WMTInt32 app_status,
                out WMTPAttributeList WFRelevantData)
```

Argument Name	Description
tool_agent_handle	This handle represents one specific Tool Agent
Pproc_inst_id	Workflow relevant data belong to this process instance and should be updated after the application is finished.
Pwork_item_id	Work Item associated with invoked application.
app_status	Information about the invoked application. (I.e. "RUNNING", "ACTIVE", "WAITING", "TERMINATED", "FINISHED", etc.)
WFRelevantData	A list or structure of workflow relevant data, which could be accessed by the Tool Agent mechanisms.

ERROR RETURN VALUE

```
WM_SUCCESS
WM_APPLICATION_BUSY
WM_INVALID_TOOL_AGENT_HANDLE
WM_INVALID_WORKITEM
WM_INVALID_PROCESS_INSTANCE
```

Figura D.5. WAPI del comando WMTARequestAppStatus

Para chequear el estado de un ítem de trabajo activo se debería usar este comando para controlar el estado de una aplicación invocada.

D.4.4 WAPI para terminar una aplicación

WMTATerminateApplication()

Este comando fuerza al *Agente de Aplicación* a terminar una aplicación [WfMCb]. La API correspondiente permite al sistema de flujo de trabajo terminar una aplicación, que esta relacionada a una instancia de proceso específica. También, una aplicación puede ser terminada por algún otro evento. De esta manera, esta API no es obligatoria dentro de las APIs de control de aplicaciones, pero le permite al *Agente de Aplicación* liberar información relevante de la aplicación. La Figura D.6 presenta la especificación de la WAPI asociada a este comando.

```
int WMTATerminateApp (
                in WMTInt32 tool_agent_handle,
                in WMTPProcInstID pproc_inst_id,
                in  WMTPWorkItemID pwork_item_id)
```

Argument Name	Description
tool_agent_handle	This handle represents one specific Tool Agent
pproc_inst_id	Workflow relevant data belong to this process instance and should be updated after the application is finished.
pwork_item_id	Work Item associated with invoked application.

ERROR RETURN VALUE

```
WM_SUCCESS
WM_APPLICATION_NOT STOPPED
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_WORKITEM
WM_APPLICATION_BUSY
```

Figura D.6. WAPI del comando WMTATerminateApplication